DataGeneral

Software Documentation

Advanced Operating System (AOS) Programmer's Manual

Advanced Operating System (AOS) Programmer's Manual

093-000120-05

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000120

Printed in the United States of America
Revision 05, October 1984

NOTICE

Advanced Operating
System (AOS)
Programmer's Manual
093-000120

Revision History:	Effective with:
Original Release - June 1976	
First Revision - April 1977	
Second Revision - June 1978	
Third Revision - June 1979	
Fourth Revision - May 1982	
Fifth Revision - October 1984	AOS Rev. 4.0

CONTENT UNCHANGED

The content and change indicators in this revision are unchanged from 093-000120-04. This revision changes only printing and binding details.

Preface

This manual is intended for use by experienced Assembly Language programmers. It details all of the system calls (i.e., macros) available to AOS users and the many considerations pertaining to their use.

If you are not experienced in programming ECLIPSE® Assembly Language, you should read the following manuals:

- *Fundamentals of Small Computer Programming,* 093-000090; a general introduction to Data General computers.
- *Programmer's Reference Manual, ECLIPSE® -Line Computers,* 015-000024; a complete description of the ECLIPSE computers' instruction set.
- *AOS Macroassembler Reference Manual,* 093-000192; this manual contains information about the AOS Macroassembler, its macro facility, and the pseudo-ops necessary to assemble and bind a program under AOS. The preface to this manual has a more detailed list of other AOS features you need to be familiar with in order to understand Macroassembler programming and where to find this information.
- *AOS Software Documentation Guide,* 093-000202; this manual describes other AOS manuals which you may want to use.

The following manuals are referred to in text:

093-000198	*AOS Library File Editor User's Manual*
093-000254	*AOS Link User's Manual*
093-000194	*AOS Operator's Guide*
093-000193	*AOS System's Manager's Guide*
093-000249	*AOS/VS SED Text Editor User's Manual*
093-000169	*Array Processor Software (APS) User's Manual*
093-000122	*Command Line Interpreter (CLI) (AOS and AOS/VS) User's Manual*
093-000158	*HASP Workstation Emulator User's Manual (AOS and AOS/VS)*
093-000217	*How to Load and Generate Your AOS System*
069-000018	*Learning to Use Your Advanced Operating System*
014-000626	*Programmer's Reference Manual, ECLIPSE® -Line Computers*
093-000178	*XODIAC™ Network Management System User's Manual*

As we describe all of the AOS system calls in detail, we will also introduce you to a number of fundamental AOS topics. Process and task concepts, memory management techniques, intertask and interprocess communications, file structure and management, and file and peripheral I/O are among the concepts this manual covers. The following list summarizes the contents of each chapter.

Chapter 1	Introduces AOS basic features, such as multiprogramming and multitasking, file structures, storing and retrieving files, and the syntax and use of system calls.
Chapter 2	Discusses process creation and management: the hierarchy AOS uses to execute processes, how the system identifies them, how you can create, access and monitor processes, and finally, how you can recover process states, if need be.
Chapter 3	Describes the structure of memory, and how to manage memory effectively using AOS. Topics within this chapter include shared page concepts, resource calls, and overlaying techniques.
Chapter 4	Explains how to initiate and maintain communications among various processes running under AOS.
Chapter 5	Describes the AOS file structure: how to create and access AOS files and directories of files.
Chapter 6	Details file input and output methods, including the use of parameter packets and file specifications. This chapter also describes the use of devices such as magnetic tapes, character devices, card readers, MCAs (Multiprocessor Communications Adapters), and full duplex modems.
Chapter 7	Introduces task concepts, and gives specifics on managing a multitasking environment within AOS.
Chapter 8	Describes how AOS handles user defined devices, how the operating system services interrupts, and how you can recover from a temporary powerfail.
Chapter 9	Lists a number of miscellaneous system calls, used for such purposes as examining the system's error message file, issuing requests to the EXEC utility, using the floating point unit, and obtaining current AOS, clock, and calendar information.
Chapter 10	Explains how to use binary synchronous communications, a method of sending bits of information over switched or dedicated communications lines.
Chapter 11	Describes the AOS connection management facility and the connection management system calls.
Chapter 12	Explains how array processing system calls can be used on AOS.
Chapter 13	Describes block I/O techniques; a method of bypassing the standard I/O calls and moving blocks of data directly.
Appendix A	Lists the error messages AOS generates. We group the messages into generic categories, and list the categories in the order they are most likely to appear.
Appendix B	Shows real-time programming in practice. The sample programs in this appendix demonstrate program initialization, error processing, and the use of offset addressing in parameter packets.
Appendix C	Gives detailed information on the system tables the Binder utility builds to relocate and manage object modules.

Appendix D	Shows the system's PARU.LS file, which lists all of the system call parameters and error codes. This appendix also provides information on how to generate PARU.LS.
Appendix E	Lists the Powers of 2 Table.
Appendix F	Provides ancillary information on multitasking and multiprogramming techniques; information to help you choose the best method to meet your needs.

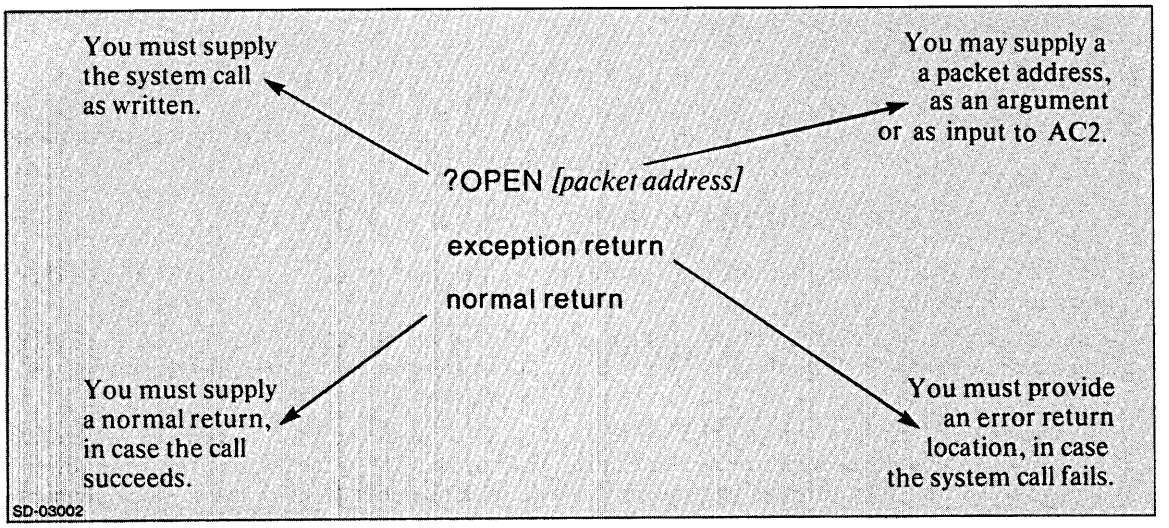
Conventions Followed In This Manual

- Unless the text specifies an alternate radix, we express all memory addresses in octal, and all other numbers in decimal.
- Mnemonics whose first character is a question mark (?) are symbols defined by Data General for use in user programs. Such symbols include system calls, offsets (words) in parameter packets, and bits within offsets.
- We use a single asterisk to indicate the multiplication of two values; i.e., 2 times 15 is written as 2*15. We use a double asterisk to indicate exponentiation; i.e., 2**15 means 2 to the 15th power. When we use scalar quantities as parameters for system calls, we use them in two's complement notation.
- *Main memory* and *core memory* refer to any type of memory you can address directly, such as magnetic core memory or semiconductor memory.
- When we refer to *magnetic tape*, we mean both 7- and 9-track magnetic tape.
- When we enclose a location or offset in parentheses (), we mean the *contents* of the entity, not the entity itself.

In describing system calls, we use the following conventions:

- Angle brackets <> surround required 8-bit quantities. Thus, <NEW LINE> indicates that a NEW LINE character, octal 12, is required.
- Italics indicate the paraphrase of a required argument. For example, *filename* means that the name of some file is required.
- An elipsis (...) means that you may repeat the preceding argument or group of arguments if you choose. Thus, *filename*... means that you have the option of supplying one or more filenames.
- Braces { } appear when you must supply one of a group of arguments. Thus, arg1, arg2 means that you may choose either arg1 or arg2, but you must choose at least one of them.
- Brackets [ ] appear when you have the option of using an argument, for a system call or value, or passing the information to an accumulator.

We show the syntax of each system call graphically, as follows:



In the figures illustrating parameter packets, some areas are shaded grey; these areas must be set to 0, since they are not used by the particular call.

End of Preface

Contents

Chapter 1 - Introduction to AOS

Chapter 2 - Process Creation and Management

Chapter 3 - Memory Management

Chapter 4 - Interprocess Communications

Chapter 5 - File Creation and Management

Chapter 6 - File Input/Output

Chapter 7 - Creating and Managing a Multitask Process

Chapter 8 - User Device Support

Chapter 9 - Miscellaneous System Calls

Chapter 10 - Binary Synchronous Communications

Chapter 11 - Connection Management

Chapter 12 - Array Processing

Chapter 13 - Block I/O

Appendix A - Exceptional Condition Codes

Appendix B - Real-Time Programming Example

Appendix C - System Tables Built in the User Context

Appendix D - Obtaining Current AOS Information

Appendix E - Powers of 2 Table

Appendix F - Selecting Multiprogramming or Multitasking

Illustrations

Tables

Chapter 1
Introduction to AOS

Data General's Advanced Operating System (AOS) combines the power found in large central computer installations with the low cost and modularity of the ECLIPSE® computers.

You can apply this operating system to manage and control a wide variety of environments. AOS is a real-time system suitable for the most demanding real-world process control application as well as a multi-user system suitable for commercial and educational applications. Data General provides high-level language support with the system so you can implement a wide range of applications.

In addition to high-level language support, several system utilities are also provided to complete your collection of program development tools. Included are a macroassembler, a binder, symbolic editors, and library builders. The Command Line Interpreter (CLI) is a commonly used interface between the operating system and users working at a console.

Typical applications of this system are overseen by a System Manager who assigns programming privileges to individual users. The System Manager can assign memory resources, execution priority, and file access on a unique basis to each user. The *AOS System Manager's Guide* and *AOS Operator's Guide* describe procedures for starting up and maintaining a running AOS system.

You may be permitted to execute only a restricted set of programs, or you may be given wider privileges such as the ability to examine and modify programs and data files. The system will ensure that appropriate access privileges are enforced and will maintain the integrity of all files.

The operating system is simple to use. Applications programmers using high-level languages need learn only a few CLI command sequences to compile and execute their programs. Operation of the CLI is described in the *AOS Command Line Interpreter User's Manual.* Programmers wishing to use more advanced operating system facilities must acquire a deeper understanding of the system. They should be familiar with topics such as process management, file structure, input/output processing, and task management; and we present these topics completely in this manual.

Multiprogramming

An ECLIPSE® computer has only one Central Processing Unit (CPU), with its accumulators, carry bit, and program counter.* The CPU can execute only one instruction at a time. However, under Data General's Advanced Operating System, you can obtain many of the features and advantages of multiple CPU's. AOS has two types of programming entities which can be used to achieve the illusion of parallel processing: tasks and processes.

The system representation of a program in some phase of execution is a *process.* Each process may be tailored by the system manager to have only those privileges and resources which it requires. This resource allocation specified by the manager is maintained by the system to promote a high degree of system efficiency and response.

The system provides a means for processes to intercommunicate and synchronize their activities. This general-purpose Interprocess Communications tool, IPC, routes messages of arbitrary length between processes.

*The ECLIPSE® M/600 computer has an additional I/O processor, but its operation is invisible to AOS users.

Two categories of memory pages (shared and unshared) are allocated to each process, and each is 2048 bytes long. The total amount of main memory (shared and unshared) allocatable to each process is 32 pages or 65,536 bytes. Unshared pages contain information which is used only by one process, while shared pages contain information that may be shared among processes according to their privileges.

The operating system maintains, in main memory, a cache-like collection of shared pages no longer in use. So if they should be required again soon, and the system does not need the memory space for other use, they can be used again without accessing the disk. This enhances system efficiency by doing away with rereading the information from disk.

The system also provides a program overlay facility, permitting a process to increase its total effective address space. Overlaying is a technique you can use for bringing files, or portions of files, from disk into selective areas of main memory. Distinct overlays can occupy the same main memory at different times. A virtually unlimited amount of information can be placed into overlay files.

Multitasking

A *task* is an asynchronous flow of code execution through a process's logical address space. Each task has its own Task Control Block, which is a block of data maintained by the operating system with a memory image of the CPU registers and other context data for each task. Since each task has its own program counter, several tasks can appear to run simultaneously, either through a section of re-entrant code (i.e., code which is not modified during execution), or through their own unique code paths. What in fact occurs is that the system schedules tasks and allows them to run alternately on the basis of their states and priorities.

A given task may be ready for execution, suspended and awaiting the occurrence of an event to make it ready, or it may actually be in control of the CPU and executing. At any moment, only one task in one process may ever be executing. The system switches control from process to process, and in the selected process the highest priority ready task will run.

The system provides an elaborate body of task management calls to permit the control of a dynamically varying environment. Tasks are not only distinguished by their states; they are also differentiated by priority and optionally by identification numbers. Although task priority is defined when each task is initiated (that is, made known to the system), task priority can be changed as conditions change. Task states can also be modified at will, and tasks can be queued for periodic execution based upon elapsed time. Finally, a mechanism is provided for transmitting and receiving messages between tasks.

File Structures

A *file* is any collection of related data treated as a unit. Devices are the means for physically storing and retrieving this information. Names assigned to files are independent of the device used to access them. This device independence permits file references to be made in programs without knowing the name of the device that will be used to access the file when the program is run.

The system organizes files into a hierarchical structure resembling a tree. The nodes of the file tree are directories, which contain information used to catalog subordinate directories and files. Certain directories are designated as Control Point Directories (CPDs), and these control the allocation of disk space for all subordinate directories.

Disk storage found in separate disk units can be associated to form a logical disk (LD) of nearly any size. Alternatively, each disk unit can be accessed as a single file within the tree structure. Accessibility to each file in the hierarchy is controlled by a mechanism called the Access Control List (ACL). The ACL lists the name of each user who can access the file, and it describes the kinds of access to which the user is entitled.

Storing and Retrieving File Data

The operating system supports and manages a wide variety of devices such as the following:

- card readers
- diskettes
- moving-head disks
- fixed-head disks
- line printers
- magnetic tape units
- digital plotters
- paper tape readers and punches
- synchronous communications lines
- asynchronous communications lines
- multiprocessor communications adapters

The IPC facility can be treated like a device for communications between processes.

There are two types of I/O which can be used in file accesses: record I/O and block I/O. Record I/O causes transfers to occur in data units of one of four types:

- dynamic
- fixed-length
- data sensitive
- variable-length

We elaborate on each of these types later in this manual. Block I/O, by contrast, is applied only to magnetic tape or disk units. This type of I/O causes data to be transferred in physical blocks, and does not require system buffering. Block I/O is described in Chapter 13.

System Calls

AOS executes system calls either in operating system space or in the user context. Calls executed in the user context require code modules in user space. These modules are all found in the user runtime library, URT.LB, and are described in Appendix D.

You express system calls in the source program as macros that begin with a question mark. Each macro is expanded at assembly time. Calls which will not be executed in the user context are expanded into two words: an indirect call to the system command processor (JSR @17), followed by a system command word. SYSID.SR defines system commands.

Except where noted, two return locations must be reserved after each call: an exception return, followed by a normal return. After system action is complete, control goes either to the exception return or to the normal return. If an exceptional condition, such as an error or end-of-file condition, was detected in the attempted execution of the command, then control returns to the exception return. Otherwise control goes to the normal return. Thus the standard instruction sequence for system calls in a source program is the following:

```
?macro name
exception return
normal return
```

The exception return must be reserved even though no exceptional condition may be currently defined. Except in rare instances, the normal return is always reserved; the text will always indicate the proper call format.

Upon either an exception return or a normal return, AC3 contains the current contents of the frame pointer, FP, location 41 octal. On an exception return, an exceptional condition code is placed in AC0. Unless the text indicates otherwise, all other accumulators will contain the values that they had upon input to the call.

All symbols (e.g., error codes and offsets in parameter tables) are referred to by their defined mnemonic instead of their numeric value. Numeric equivalents are found in appropriate parameter files; see Appendix E. By assembling programs with these parameter files, you can write programs which refer to symbolic values mnemonically.

Exception codes which can be received from each system and task call may be listed by category and specifically by mnemonic. For example, in the description of the call to assign a device, error ERDAI (device already assigned) is listed specifically, and the category "FILE SYSTEM codes" is also listed. Appendix A describes each of the conditions in detail and enumerates the separate conditions which are listed by category.

Byte Pointers

Some system calls require byte pointers. An ECLIPSE computer word is 16 bits in length, and its bit positions are numbered left to right, from 0 to 15 inclusive. A byte is 8 bits in length. A byte string consists of a sequence of bytes, packed left to right in a series of one or more words. A byte pointer consists of a single word with two fields. The left field consists of bit positions 0 through 14, and it contains the address of the word containing the byte which was selected. The right field consists of the bit position 15. When the state of this bit equals 1, the pointer selects the right half or byte of a word; when the state of this bit equals 0, the pointer selects the left byte of a word.

If you need to point to a byte in a word whose address you have defined as a variable V, the value V*2 will serve as a byte pointer to the left byte, and V*2+1 will point to the right byte. See Figure 1-1 for an illustration of byte pointer structure.

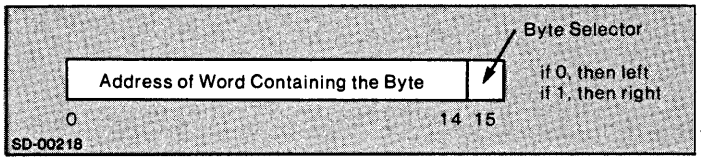


Figure 1-1. Byte Pointer Structure

Parameter Packets

Some system calls require parameter packets, which are blocks of data in the user's address space, or context, in order to execute. You may supply the packet address in AC2 when you make the call, or you may provide it as an argument to the macro call itself; the assembler will then provide code in your program which will load the packet address into AC2. This is indicated throughout the manual in the following manner:

?macro name *[packet address]*

The system uses the data you supply in a parameter packet to decide how to execute a call. The location of data in a packet determines how it will be interpreted. There are two ways to set up a parameter packet: with *absolute addressing* or *offset addressing*. These are illustrated in Figures 1-2 and 1-3, respectively. The examples are taken from the ?GTMES call, documented in Chapter 9. The purpose of this call is not relevant to these examples; we use it here only to illustrate parameter packets.

```
 0001 ABSOL AOS ASSEMBLER
01
02                    ;SOMEWHERE IN A PROGRAM THE ?GTMES CALL IS MADE.
03
04                SUBRTN: ?GTMES PKT        ;CALL IS MADE WITH ARGUMENT "PKT"
05 00004 000012           JMP ERROR         ;ERROR RETURN
06 00005 000013           JMP CONTINUE      ;NORMAL RETURN
07
08                    ;LATER IN THE PROGRAM, OUTSIDE OF THE
09                    ;EXECUTION PATH, THE PACKET IS LOCATED,
10                    ;STARING AT ADDRESS PKT.
11
12 00006 000002     PKT: ?GCNT              ;FIRST WORD OF PACKET
13 00007 000001          1                  ;SECOND WORD OF PACKET
14 00010 000000          0                  ;THIRD WORD OF PACKET
15 00011 177777          -1                 ;LAST WORD OF PACKET
16

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure 1-2. Absolute Addressing in a Parameter Packet

The ?GTMES call has a four-word packet currently defined. In Figure 1-2, we have given the packet address as PKT, and then provided four contiguous words of data. The variable ?GCNT is a system-defined parameter, listed in PARU.SR, which we use instead of the value currently assigned to it. This ensures that even if the value of ?GCNT is redefined, the example shown will still execute correctly, provided that it is reassembled. Note, however, that if, in a future revision of AOS, the packet is redefined, the example will be obsolete.

In Figure 1-3, we have made use of *word offsets*. These are system-defined parameters, also listed in PARU.SR, which reference the words of the parameter packet. Presently, ?GREQ is word zero, ?GNUM is word one, ?GSW is word two and ?GRES is word three. Note that in this example, if the packet is redefined in a future release of AOS, our program will still run correctly if reassembled, because Data General will insure that the word offsets still correspond to the appropriate data, regardless of their actual physical location in the packet.

```
 0001 OFFST AOS ASSEMBLER
01
02                      ;SOMEWHERE IN A PROGRAM THE ?GTMES CALL IS MADE.
03
04                          ?GTMES PKT      ;CALL IS MADE WITH ARGUMENT "PKT"
05 00004 000012             JMP ERROR       ;ERROR RETURN
06 00005 000013             JMP CONTINUE    ;NORMAL RETURN
07
08                      ;LATER IN THE PROGRAM, OUTSIDE THE
09                      ;EXECUTION PATH, THE PACKET IS LOCATED,
10                      ;STARTING AT ADDRESS PKT.
11
12          000006      PKT:.LOC PKT+?GREQ          ;WORD "?GREQ" OF PACKET
13 00006 000002             ?GCNT
14          000007          .LOC PKT+?GNUM          ;WORD "?GNUM" OF PACKET
15 00007 000001             1
16          000010          .LOC PKT+?GSW           ;WORD "?GSW" OF PACKET
17 00010 000000             0
18          000011          .LOC PKT+?GRES          ;WORD "?GRES" OF PACKET
19 00011 177777             -1
20

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure 1-3. Offset Addressing in Parameter Packet

Some parameter packets contain *flag words,* in which each bit has a special meaning. These bits are set with bit masks, which are system-defined parameters listed in PARU.SR, each equal to a single set bit. For example, bit mask ?PVEX in offset ?PPRV of the ?PROC call (documented in Chapter 2) is currently defined as 0000000000100000 binary. Adding this mask to a word will set bit 10, provided that the word was zero originally.

Note that you must set all reserved words in a parameter packet to zero.

A few I/O calls allow you to use extended parameter packets; these are explained in Chapter 6, "File Input/Output."

End of Chapter

Chapter 2
Process Creation and Management

Process Concepts

A *task* is an asynchronous flow of code execution through a process's logical address space. You may assign many tasks to operate asynchronously in a single re-entrant sequence of instructions, or you may assign each task a distinct instruction path. In either case control always goes to the highest priority task currently ready for execution.

A *process* is a collection of tasks initiated by the execution of one or more programs with a unique ID, competing for system resources such as memory, I/O devices, file space, and processor time. The system supports a multiprocess environment. That is, the system will permit a dynamically varying number of independent processes to compete for system resources.

The current contents of any process's primary address space is a *program*. A program is an executable core image, produced typically by assembling or compiling one or more source files to produce one or more object files. The Link utility links object files to produce the executable program file. A program is a static entity; it contains code paths which are executed by tasks, the dynamic entities.

Processes compete for memory and the CPU. The operating system allocates memory to certain processes, and CPU control to processes which have obtained memory. The system allocates memory and CPU control to each process according to several parameters, one of which is *priority*.

Memory Contention

Each process is one of three types:

- Resident
- Pre-emptible
- Swappable

Resident processes always remain in main memory; the other two types may be written out to disk. The resident type should be reserved for critical processes. A typical application for resident processes is the line interface process in a communications system and the alarm process in a real-time process control system.

Pre-emptible processes do not have the absolute control over main memory that resident processes possess. A pre-emptible process can be deprived of its main memory and be written out to disk in one of two situations: when either a resident process or higher priority pre-emptible process requires memory, or when the process has become blocked and any other process requires memory. (The blocked state is described in the next section.)

The lowest priority process is the *swappable* process. Swappable processes receive main memory only when it is available after both higher types have received what they need. Swappable processes, as their name implies, may be written out to disk at the discretion of the system scheduler based on its time-slicing policy. The system derives a priority for each swappable process (and only this type) by adjusting the user-assigned priority with other factors such as process behavior in the past. The system assures an optimal response time for all processes and efficient throughput for the total system.

Resident and pre-emptible processes are assigned priorities ranging from 1 (the highest) through 255 (the lowest). Swappable processes have one of three priorities: 1 (high), 2 (normal), and 3 (low). See the *AOS System Manager's Guide* for instructions on selecting process priorities.

In summary, memory contention between processes is resolved in the following way: first, all resident processes receive main memory and retain that memory. Secondly, a pre-emptible process receives memory by priority and keeps that memory, as long as the process remains unblocked, unless a resident process or a higher priority pre-emptible process requires the memory. When a pre-emptible process becomes blocked, other processes compete for its memory. Finally, the system gives the remaining memory to swappable processes based on their derived priority; these processes retain memory for periods of time that are determined by the system's time-slicing policy.

CPU Contention

Every process will be in one of three states at any given moment:

- Eligible
- Ineligible
- Blocked

A process is *eligible* to gain CPU control if it has been allocated main memory and is ready to run. A process is *ineligible* to receive CPU control if it has not been allocated main memory but in all other ways is ready to run. When it is allocated main memory, it becomes an eligible process. Each newly created process is initially ineligible. A *blocked* process is one which is suspended until a specific external event occurs (the event may never occur). A process can become blocked either voluntarily or involuntarily.

The system gives CPU control to processes in this manner: among processes that are eligible to run, control is given first to the highest priority resident or pre-emptible process (both types are treated as a single group), and then it is given to the highest priority swappable process. The priority of a swappable process is a composite factor including both a user-assigned priority and adjustment factors derived by the system based on the past behavior of the process.

Blocking Rules

Resident, pre-emptible, and swappable processes can become blocked if:

- the process is blocked when it is created; or
- the process is the object of a block command (?BLKPR); or
- the process creates a son process and blocks until that son terminates; or
- the process issues a system call that suspends its only active task. This condition implies that the process has only one task, or that all of its other tasks are suspended. ?IREC (receive an IPC message, Chapter 4) and ?DELAY (suspend a task for an interval of time, Chapter 9) are two examples of calls that can cause such a block in a process. The system waits briefly for the calls to be completed, but if they are lengthy, and if all other tasks in the process are suspended, AOS will block the process that issued the call.

A process which is blocked for any reason can be unblocked by one of the following events:

- by being the target of an unblock call (?UBLPR);
- by a task within a process becoming ready for run (e.g., ?DELAY interval expired or IPC message received);
- by a son process's terminating (if the father process is waiting for that termination).

If a process is blocked because it is waiting for a son process to become terminated, then no other event can cause that process to become unblocked. If a process is blocked by ?BLKPR, then only ?UBLPR can unblock it.

Creating Processes

When you initialize AOS, the system creates a primary logical node called the *root*. From this root the system creates a system process (the PMGR, the peripheral process which manages character-oriented I/O) and a user process called the *initial process* (a CLI process). The initial process is highly privileged so that it can create other processes of any type with any priority.

All processes in existence at any moment are related in a tree-like structure. Figure 2-1 depicts one possible tree structure of processes.

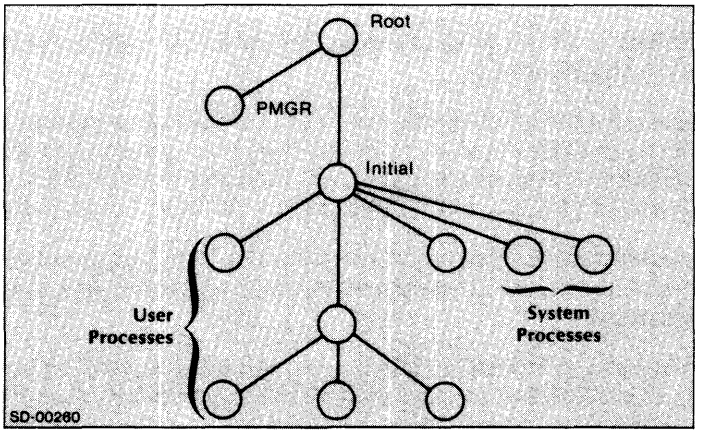


Figure 2-1. Process Tree

Each created process has an independent existence and will exist until it terminates itself, or is terminated by another process, or exceeds the maximum execution time (if any) assigned to it when it was created. Process creation can be performed using the CLI whenever an operator issues a request from the console to create a new process; with such a request the CLI issues a ?PROC system call.

Any process bearing a filial relationship to another process which is higher on the tree is called a *subordinate* process. Thus son, grandson, and great-grandson processes are all subordinate to a single, *superior* process.

There is an elaborate body of privileges attached to each process which describes, among other things, the limits of its process-creation capabilities.

Privileges sharply define certain capabilities of a process. You should exercise caution whenever you pass privileges to an untested program. Thus the system permits each caller to give none, all, or some subset of his privileges to a son process (see the discussion of ?PFPM in the ?PROC packet). No caller can pass privileges he lacks to a son process.

The following list describes each of the privileges. We describe the symbolic names given to each privilege at length later in this chapter with the system call to create a process, ?PROC.

?PVPC — A process with this privilege may create an unlimited number of son processes.

?PVEX — This privilege permits the caller to create a process and remain unblocked (i.e., not select ?PFEX in the ?PROC packet). Without this privilege, a process must become blocked when it creates a son process.

?PVPR — This privilege permits a process to change its own priority to a value higher than its initial priority, or, when it creates another process, to create it at a higher priority.

?PVTY — This privilege permits a process to create another process of any type (e.g., resident). This privilege also permits a process to change the type that it was assigned when it was created. Without this privilege, a process cannot change its own type or that of any of its sons.

?PVIP This privilege permits a process to issue the primitive IPC calls ?ISEND and ?IS.R.

?PVUI This privilege permits a process to create another process which has a username that differs from that of its father. Without this privilege, a process cannot specify a username for its son, and all its sons will have the father's username.

?PVDV This privilege permits a process to access user devices (i.e., to issue system calls ?IDEF, ?DEBL, and ?STMAP).

?PVSP This privilege permits a process to issue ?SUPROC and enter superprocess mode (see "Superprocess Mode" below).

?PVSU This privilege permits a process to issue ?SUSER and to enter the Superuser mode (See "Superuser Mode" below).

The system determines the number of offspring a process can create by checking its ?PROC packet for the ?PVPC and ?PVEX privileges. It also checks for two other parameters: offset ?PPCR, which specifies the maximum number of offspring allowed in the process; and ?PFEX, a mask within offset ?PFLG, which determines whether the process will be blocked while its sons execute.

Offset ?PPCR is a cumulative value. That is, if a process with a ?PPCR value of 10 creates 2 sons, each with a ?PPCR setting of 4, the original process cannot create any other sons, since 2 sons plus 2*4 (8 potential grandsons) equals 10.

The ?PVPC privilege enables a process to ?PROC an unlimited number of sons. When a process without this privilege tries to create a son, the system checks the ?PPCR count. If the number of sons and their combined ?PPCR count exceed the caller's ?PPCR value, the system signals an error.

The ?PVEX privilege enables a process to create a son without being blocked. If bit ?PFEX in the ?PROC packet is not set, indicating that the process should remain unblocked while creating a son, then the system checks for the ?PVEX privilege. If the process has the privilege, the system will allow it to create the son without being blocked. If not, the system takes the error return from ?PROC.

A process may request that a son process be created in the blocked state by specifying ?PFBS in the ?PROC packet. In this case, the system creates the son, but blocks it before it begins executing. The son remains blocked until a superior process unblocks it by issuing a ?UBLPR system call.

Identifying Processes

Two means are provided for identifying each process: a *process name* and a *process ID (PID)*. The PID is a number, unique for each process, which ranges from 1 through 64. The system assigns the number when each process is created. Each process name consists of two parts: a *username* and a *simple process name,* separated by a colon.

Both the username and simple process name may be up to 15 characters long. You can use all valid filename characters. The username is analogous to a surname, since by default all sons bear the username of the process that created them. Only by privilege ?PVUI can one process create another and give to that process a username which differs from its own. Usernames are employed to define file access rights for processes (see Chapter 5, "Controlling Access to Files").

Each process bearing the same username must have a unique simple process name. Thus three processes with simple process names PROC1, PROC2, and PROC3 may all have the same username: SMITH. Three other processes might also have the simple process names PROC1, PROC2, and PROC3 and have another username: JONES. In this case, the full process names and a possible tree representation of those processes are shown in Figure 2-2.

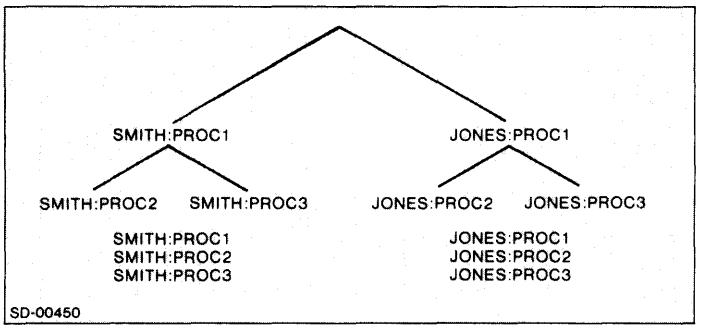


Figure 2-2. A Sample Process Tree

Each process in the two sets is uniquely distinguished from its brothers by a simple process name (PROC1, PROC2, or PROC3), and each process in the whole system has a unique full process name.

Whenever a system call requires a process name as an input parameter, you can supply either a full process name or a simple process name. Likewise, when we say *process name,* both full and simple process names apply. If a simple process name is shown as a call parameter, the system will supply the username of the calling process. Thus if SMITH:PROC1 supplied the name PROC2, the system would understand the full name to be SMITH:PROC2.

Three system calls permit each process to obtain its own name or PID or those of another process. ?PNAME returns the full process name associated with a PID, the PID associated with a process name, or the calling process's full process name and/or PID. ?GUNM returns a username for a given PID or process name, or the calling process's username. ?DADID returns a father process's PID, either the father of the calling process or the father of some other process.

Process Management Calls

?PROC creates a process. In Figure 2-3, process A is the creator of processes B, C, and D. Finally, D created G and E, while B created son F.

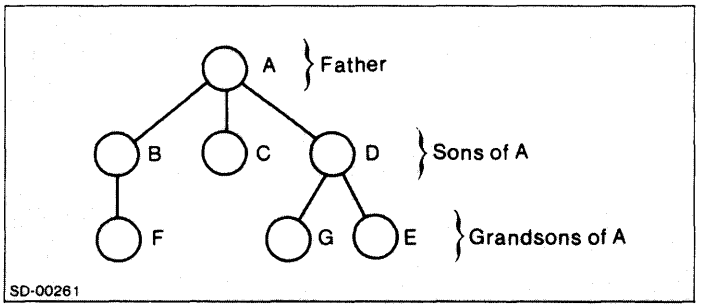


Figure 2-3. Differing Relationships in a Process Tree

It is common for one process to create a son who in turn creates another son. In the above illustration, A might be the system administrator process which creates processes B and C so that two users can perform source program editing. If the user in process B wishes to assemble his program, he can create a son, F, and then becomes blocked until the assembly is completed in F.

In Figure 2-3, processes B through G are all subordinate to A, but only F is subordinate to B, its superior. Any superior process can selectively block (?BLKPR) a subordinate process, or unblock it (?UBLPR). A process can change its own priority or the priority of any of its subordinates by issuing system call ?PRIPR. However, only processes with ?PVPR can establish a priority for a target process higher than the priority the target process had when it was created. Similarly, a process can change its own type or the type of any subordinate process with system call ?CTYPE (if the superior process has privilege ?PVTY).

If a user wants to change the program that a process is running but retain the process's states and privileges; e.g., keep the same working and default directories and the same console; then he issues the system call ?CHAIN from within the process. This procedure is called *chaining* to a new program.

?TERM terminates a process. If any process which is terminated has any sons (grandsons, etc.), they too are terminated. A process can be terminated regardless of its current state (i.e., eligible, ineligible, or blocked). Thus, if in the tree structure shown in Figure 2-3 process B is terminated, process F would also be terminated and the new tree structure would appear as shown in Figure 2-4.

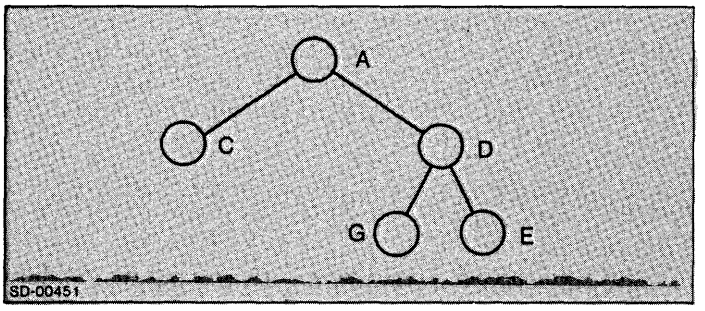


Figure 2-4. New Process Tree After Terminating B

A process terminating itself by ?TERM can transmit a final message of any length via the IPC to its father.

?RETURN provides a convenient way for any process to terminate itself and to return a message to its father. If the father process is the CLI, ?RETURN passes the message in CLI format. Procedures for a father process to receive IPC termination messages are described in Chapter 4, "Receiving Process Termination Messages."

Process Traps

A variety of different conditions will cause a hardware fault (called a *trap*) and terminate a process. These conditions include the following:

- An attempt to write into a write-protected shared area.
- An attempt to refer to an address outside the user context.
- An attempt to use more than 14 levels of indirect addressing in a memory reference instruction.
- An attempt to issue a machine-level I/O instruction without having first issued system call ?DEBL (see Chapter 8).
- A system-detected fatal process error.

The system returns an appropriate termination message describing the trap to the father.

Break Files

When you issue the system call ?BRKFL (documented later in this chapter), when one of your processes traps, or when you issue a CTRL-C, CTRL-E sequence from a process's console, the operating system attempts to create a break file in the current working directory. If no pathname is assigned by the process creating the file, it is given the default name ?pid.time.BRK where pid is the three-digit PID of the currently running process and time is in the form hours_minutes_seconds.

The break file contains memory images of the currently running process, and of certain portions of the operating system's memory. All unused memory words are zeroed in the break file.

If there is not enough disk space in the current working directory, or if the current process does not have write or append access to the current working directory, no break file will be created and, in the case of a ?BRKFL, the process will not be terminated.

If the termination of a process was caused by a validity trap, the system will also record five words from the User Status Table (UST) in the break file. These words, which describe the state of the accumulators and program counters at the time of the trap, are as follows:

Location	Label	Value After Trap
402	?USTSS	AC0 at time of the trap
403	?USTSE	AC1 at time of the trap
404	?USTS1	AC2 at time of the trap
405	?USTS2	AC3 at time of the trap
406	?USTDA	Bit 0 = carry at time of trap Bits 1-15 = program counter at time of the trap

Note that the system retains status information from the UST *only* when there is a validity trap. If you interrupt a process by issuing a CTRL-C, CTRL-E sequence or the ?BRKFL call, the system saves the process state from the task control block. (See "Task Control Block," Chapter 7.)

Superuser Mode

Any process with privilege ?PVSU can issue system call ?SUSER and enter the superuser mode. When it is in superuser mode, a process can access any file regardless of file access controls (see the section in Chapter 5, "Controlling Access to Files"). Superuser mode also allows you to determine a user's access privileges for a specific file. Any process that has the ?PVSU privilege can give this privilege to its son processes. Any process created with ?PVSU, whose father was in superuser mode at the time of the creation, will be in superuser mode initially. Any process in superuser mode remains in this mode until it issues ?SUSER to leave this mode, or is terminated.

Superprocess Mode

Any process with privilege ?PVSP can issue system call ?SUPROC and enter superprocess mode. While in superprocess, the calling process can change the state of any other process running on the system, instead of being restricted to its subordinates. This means that a process in superprocess mode can affect any other process by issuing the following system calls: ?BLKPR, ?BRKFL, ?CTYPE, ?PRIPR, ?TERM, and ?UBLPR. Any process with this privilege can give it to its sons. A son created while the father is in superprocess mode will not initially be in that mode. Use this call with extreme caution, since it enables a process to affect any other process in the system.

Get the Runtime Statistics of a Process

System call ?RUNTM provides you with a number of current statistics which describe resource utilization by the calling process or one of its sons. These include:

- real-time elapsed since the process was created.
- CPU time used by the process.
- number of blocks read or written via the data channel.
- page usage over time.

The time elapsed since the process's creation is given in seconds, with a range of from 0 to (2**32)-1
* seconds. Elapsed CPU time is given in milliseconds. Page usage is given in page-milliseconds.

Process Histograms

Each resident process can monitor its use or another process's use of the CPU. A *histogram* shows how often the target process receives control and contrasts this with how often other processes or the system receive control. Moreover, each histogram shows how often the system was in an idle loop waiting for some process to become ready to run. You may optionally request a more detailed histogram showing how often control was in specific intervals over a stated range within the target process. Thus each histogram provides a global view of CPU activity, showing how often a target process receives control, optionally where within the process it received control, and how often other processes and the system received control. Histogram information is updated after each real-time clock pulse.

Any resident process can issue ?IHIST to collect information about itself or any other process, and to build a histogram. Alternatively, a resident process can issue ?IXHIST to collect information about one process, or about the system as a whole. No process can initiate more than one concurrent histogram. ?IHIST and ?IXHIST name a target process (which need not be resident) and specify a histogram parameter packet. The packet also names the array where the histogram will be recorded. Histogram monitoring will be terminated when the process that initiated it issues a ?KHIST or is terminated for any reason. ?IXHIST initiated histograms are terminated when the target process terminates.

?IHIST and IXHIST zero out the entire data array before data collection begins.

System Call Summary

The following list summarizes the process creation and management calls:

?BLKPR	Block a process.
?BRKFL	Terminate a process and create a break file.
?CHAIN	Run a new program within a process.
?CTYPE	Change the type of a process.
?DADID	Get a father process's PID.
?GUNM	Get a process's username.
?IHIST	Start a histogram.
?IXHIST	Start a histogram (extended functionality).
?KHIST	Stop a histogram.
?PNAME	Get a full process name.
?PRIPR	Change a process priority.
?PROC	Create a process.
?PSTAT	Get process status information.
?RETURN	Terminate a process and pass a message to the father.
?RUNTM	Get process runtime statistics.
?SUPROC	Enter or leave superprocess mode.
?SUSER	Enter or leave superuser mode.
?TERM	Terminate a process.
?UBLPR	Unblock a process.

These calls are described in detail in the following pages.

?BLKPR

Block a process.

?BLKPR
exception return
normal return

Input/Output

Input:

AC0 byte pointer to the name of a process to be blocked, process ID (PID) of process to be blocked, or -1 if the process is to block itself.

AC1 -1 means AC0 contains a byte pointer, 0 means AC0 contains a PID, or contents are ignored if AC0 contains -1.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

ERMPR System call parameter address error.

ERPNM Illegal process name.

ERPRH Attempt to access a process which is not in the process tree.

Description

You may issue this call to block a subordinate process or yourself or, if you are in superprocess mode, any process in the hierarchy. When a process becomes blocked as a result of this call, it can only be unblocked by a call to ?UBLPR.

?BRKFL

Terminate a process and create a break file.

?BRKFL
exception return
normal return

Input/Output

Input:

AC0	byte pointer to name of process to be terminated, PID of process to be terminated, or -1 to terminate the process issuing this call.
AC1	-1 if AC0 contains a byte pointer, 0 if AC0 contains a PID, or ignored if AC0 contains -1.
AC2	byte pointer to break file's pathname, or -1 for default pathname.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	destroyed.

Exceptional Condition Codes in AC0

ERPRH Attempt to access a process which is not in the process tree.

FILE SYSTEM codes

Description

?BRKFL terminates a subordinate process, or the caller's process, or, if the calling process is in superprocess mode, any process in the hierarchy. In addition, it creates a break file in the current working directory, consisting of the memory image of the terminated process. To create a break file in the current working directory, you must have the access that you would need to create any other file in that directory. If you don't supply a break file pathname, the break file receives the name *?pid.time.BRK*, where *pid* is the three-digit PID of the caller and *time* is in the form hours_minutes_seconds.

If for any reason the break file cannot be created (e.g., the maximum disk space of the control point directory is exceeded), the calling process will be given the exception return and the target process will not be terminated. In this case, you may choose to issue the ?TERM call.

?CHAIN

Chain the current program.

?CHAIN
exception return

Input/Output

Input:

AC0	byte pointer to name of program file which will be executed.
AC1	bit 14=0, don't flush outstanding IPC messages. bit 14=1, flush outstanding IPC messages. bit 15=0, pass control to the program at its entry point. bit 15=1, pass control to the Debugger.
AC2	address of IPC message header to be passed to new program (0 for no message header).

Output:

AC0	undefined.
AC1	undefined.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERNSW	Out of swap file space. There must be enough space in the swap file to permit the new program file to be written out to disk.
ERMEM	Insufficient amount of memory available. Possibly the new program file requires more main memory than was allocated for use by the process.

FILE SYSTEM codes

Description

This call performs an orderly release of the system resources used by a process (while retaining all characteristics of the process and state variables such as the current console and working directory), and executes a new program file. If the debugger bit is set, or if the program is already being debugged, then the new program is brought into main memory, but the debugger receives control. The program whose execution was terminated will not be saved, i.e., its state variables at the time of the ?CHAIN call will not be saved; nor can they be determined. Contents of accumulators passed to the new program are indeterminate.

There is no normal return for this call.

For a discussion of the contents of the initial IPC message header, see the ?PROC system call in this chapter.

?CTYPE

Change a process type.

?CTYPE
exception return
normal return

Input/Output

Input:

AC0	byte pointer to name of process whose type is to be changed, PID of process whose type is to be changed, or -1 to change the calling process's type.
AC1	-1 if AC0 contains a byte pointer, 0 if AC0 contains a PID; otherwise it is not meaningful.
AC2	new type. Type parameters are: 0, swappable; ?PFRP, pre-emptible; ?PFRS, resident.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERMPR	System call parameter address error.
ERPNM	Illegal process name.
ERPRH	Attempt to refer to a process that is not in the process tree.
ERPTY	Illegal process type.
ERPRP	Illegal process priority.

Description

This system call permits the caller to change its own process type or that of a subordinate process, or, if the caller is in superprocess mode, any process. The calling process must have privilege ?PVTY.

?DADID

Get father process's ID.

?DADID
exception return
normal return

Input/Output

Input:

AC0 ID of process whose father's PID is to be obtained (-1 to get the caller's father's PID).

Output:

AC0 unchanged.

AC1 PID of the process's father.

Exceptional Condition Codes in AC0

ERPRH Attempt to access a process which is not in the process tree.

Description

This call returns a father process's ID. The target process may be the father of the calling process or the father of some other process.

?GUNM

Get a process's username.

?GUNM
exception return
normal return

Input/Output

Input:

AC0	-1 (calling process), PID, or byte pointer to process name.
AC1	-1 if AC0 contains a byte pointer; 0 if AC0 contains a PID.
AC2	byte pointer to a 16-byte area.

Output:

AC0	1 if target process is a superuser; otherwise, 0.
AC1	target process's privileges word.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERMPR	System call parameter address error.
ERPNM	Illegal process name.
ERPRH	Attempt to access a process which is not in the process tree.

Description

This call returns a process's username, terminated by a null, in the area pointed to by AC2. If -1 is input in AC0, the contents of AC1 are ignored and the calling process's username is returned. Otherwise, AC1 indicates whether AC0 contains a PID or byte pointer to a process name.

?IHIST

Start a histogram.

?IHIST
exception return
normal return

Input/Output

Input:

AC0	PID, -1 (for this process), or byte pointer to process name.
AC1	-1 if AC0 contains a byte pointer; otherwise AC0 contains a PID or -1.
AC2	address of histogram parameter packet.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERHIS	Caller has attempted to define more than one active histogram, or there is an illogical parameter in the parameter packet.
ERMPR	Histogram array is outside the caller's address space.
ERPNM	Illegal process name.
ERPRH	Attempt to refer to a process not in the process tree.
ERPRV	Calling process is not a resident process.

Description

This call starts the collection of information about a process, and builds a histogram. The caller must
* be a resident process. ?IHIST zeros out existing histograms in the data array. The parameter packet passed to ?IHIST is as follows:

Offset	Contents
?HIST	Starting address within the process where monitoring is to occur.
?HIEND	Ending address within process where monitoring is to occur.
?HIWDS	Size of intervals to be monitored within the process, 0 or 1 to *n*, where *n* is the number of words in the range to be monitored.
?HIBUF	Address of array to receive histogram information.

If ?HIWDS contains 0, then a simple histogram records merely how often the target process gains control; the histogram gathers no range statistics within the process and the contents of ?HIST/?HIEND are ignored.

Figure 2-5 illustrates the use of these parameters.

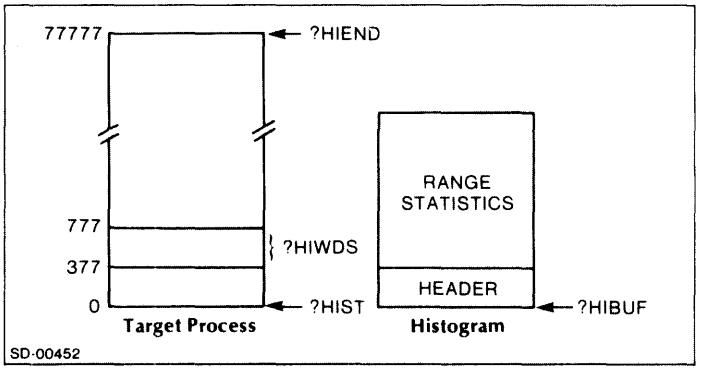


Figure 2-5. Sample Histogram Parameters

The histogram in Figure 2-5 will monitor the entire 64 Kbyte context of the target process, and will compile statistics for 256-word intervals through the context.

Each histogram array has two parts: a fixed length header, followed by double precision array elements corresponding to each interval. Table 2-1 shows how arrays are structured.

Table 2-1. Histogram Structure

Array Offset	Contents
?HTTH ?HTTL	Total number of real-time clock pulses (ticks) counted in this histogram.
?HPRH ?HPRL	Total number of ticks when PC was in the target process but outside the specified range.
?HAPH ?HAPL	Total number of ticks in some other process.
?HSBH ?HSBL	Total number of ticks in the operating system proper (but not in an idle loop).
?HSIH ?HSIL	Total number of ticks in a system idle loop.
?HARAY ?HARAY+1 . . .	Total number of ticks in the first inverval.
?HARAY+n*2-2 ?HARAY+n*2-1	Total number of ticks in the nth interval.

?IXHIST

Start a histogram (extended functionality).

?IXHIST *[packet address]*
exception return
normal return

Input/Output

Input

AC0	PID, -1 (for this process), or byte pointer to process name.
AC1	Bit 0=0, if AC0 contains a PID or -1. Bit 0=1, if AC0 contains a byte pointer. bits 1-15 contain histogram type.
AC2	address of parameter packet.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

Same as for ?IHIST.

Description

This call directs the system to build a histogram in the caller's address space. ?IXHIST differs from ?IHIST in the information it collects. A histogram initiated by the ?IXHIST system call can either collect information about a specific process, or collect statistics about the system as a whole.

?IXHIST will zero the entire data array before the data collection begins. A process can have only one histogram active at a time.

The system will stop data collection for histograms started by ?IXHIST if any of the following occurs:

- Issuance of a ?KHIST system call by the initiating process.
- Termination of the initiating process.
- Termination of the target process.
- Detection of a histogram fatal error condition (e.g., a page of the histogram data area is removed from the initiator's address space).

Although ?IXHIST allows several types of histograms to be build, the current release of AOS permits only two: ?HTTP, ?HTAP. Bits 1-15 of AC1 specify the type. The types differ in the parameter packet and the data collected.

The parameter packet for ?IXHIST follows. Parameters marked with an asterisk (*) are not required in the packet for ?HTAP histograms.

Offset	Contents
?HDAT	Word address of data area.
?HRE1	Reserved. Must be zero.
?HRE2	Reserved. Must be zero.
?HRDC*	Count of range descriptors. Must be between zero and ?HMXR.
?HRD*	Start of range descriptors (if any).

Histogram Types

?HTTP is a one process histogram, similar to histograms initiated by ?IHIST. The caller specifies the target process along with several "range descriptors." Each range descriptor specifies:

- An interval size (which dictates how the range is divided).
- A starting address within the address space of the target process.
- An ending address within the address space of the target process.

Two restrictions apply to the range descriptors. First, they must not reside in a write-protected area of the caller's address space. Second, the contents of the range descriptors must not be changed during the entire time the histogram is in progress. Violation of this second restriction could cause premature termination of the data collection, unreliable data, process traps, or other unpredictable results.

The format of the range descriptors is as follows. Each range descriptor is of length ?HRSZ.

Offset	Contents
?HRST	Starting address in target process for this range.
?HREN	Ending address in target process for this range.
?HRIS	Interval size for this range.
?HRRE	Reserved. Changed by the system.

In the data array, one double word counter is associated with each interval in the range. If the real-time clock ticks while the process is executing within a specified range, the system increments the appropriate counter. Ranges may overlap as desired.

The size in words of the ?HTTP histogram data array is ?HCNT plus two times the number of counters associated with each range descriptor. The number of counters associated with a range descriptor is given by:

(ending address - starting address + 1) / interval size

with the result rounded up.

Table 2-2 gives the data array structure for ?HTTP histograms.

Table 2-2. Data Array Structure of ?HTTP Histograms

Offset	Contents
?HTOH/?HTOL	Total ticks (real-time clock pulses) in this histogram.
?HIDH/?HIDL	Ticks in a system idle loop.
?HPMH/?HPML	Ticks in the target process.
?HGHH/?HGHL	Ticks in the ghost of the target process.
?HATH/?HATL	Ticks in the system when the system is acting on behalf of the target process.
?HOPH/?HOPL	Ticks in a process other than the target.
?HOGH/?HOGL	Ticks in the ghost of a process other than the target.
?HAOH/?HAOL	Ticks in the system when the system is acting on behalf of a process other than the target.
?HSYH/?HSYL	Ticks in the system not in an idle loop and not when the system is acting on behalf of a specific process.
?HRNH/?HRNL	Ticks within the target process when the PC was within one or more specified range.
?HCNT	Start of counters for specified ranges. All counters are double words. Groups of counters are in same order as range descriptors. Within each group, the first counter is for the first interval in the range, the second counter is for the second interval, etc.

?HTAP builds "all process" histograms. This type of histogram is a collection of data on all 64 possible processes in the system. Three double word counters in the data array are associated with each possible PID (1 to 64). For each PID, one counter shows the number of ticks occurring in that process, one counter shows ticks in that process's ghost, and one counter shows ticks in the system when the system is acting on behalf of that process (e.g., running a system call).

For ?HTAP histograms, specify a -1 in AC0 and a zero in bit zero of AC1 on the ?IXHIST call.

Table 2-3 shows the data array structure for ?HTAP histograms. The data array size is ?HAPS words.

Table 2-3. Data Array Structure for ?HTAP Histograms

Offset	Contents
?HTOH / ?HTOL	Total ticks (real-time clock pulses) in this histogram.
?HIDH / ?HIDL	Ticks in a system idle loop.
?HPMH / ?HPML	Not used.
?HGHH / ?HGHL	Not used.
?HATH / ?HATL	Not used.
?HOPH / ?HOPL	Ticks in some process.
?HOGH / ?HOGL	Ticks in the ghost of some process.
?HAOH / ?HAOL	Ticks in the system when the system is acting on behalf of some process.
?HSYH / ?HSYL	Ticks in the system not in an idle loop and not when the system is acting on behalf of a specific process.
?HRNH / ?HRNL	Not used.
?HCNT / ?HCNT + 1	Not used.
?HCNT + 2 / ?HCNT + 3	Not used.
?HPID / ?HPID + 1	Ticks in PID 1.
?HPID + 2 / ?HPID + 3	Ticks in ghost of PID 1.
?HPID + 4 / ?HPID + 5	Ticks in system when system is acting on behalf of PID 1.
?HPID + (N-1) * 6 ?HPID + (N-1) * 6 + 1	Ticks in PID N.
?HPID + (N-1) * 6 + 2 ?HPID + (N-1) * 6 + 3	Ticks in ghost of PID N.
?HPID + (N-1) * 6 + 4 ?HPID + (N-1) * 6 + 5	Ticks in system when system is acting on behalf of PID N.

?KHIST

Stop a histogram.

?KHIST
exception return
normal return

Input/Output

Input:

none.

Output:

none.

Exceptional Condition Codes in AC0

ERHIS	Caller has no active histogram.
ERHTT	Termination of histogram target process.
ERHIP	Error detected while histogram in progress.

Description

This call stops the histogram started up by a previous ?IHIST or ?IXHIST system call. The system will also stop a histogram if the initiating process terminates.

For ?IXHIST initiated histograms, the system will stop the histogram prior to the ?KHIST call if the target process terminates or if certain error conditions (such as a page containing the data area or the ?IXHIST packet being removed from the histogram's address space) occur while the histogram is being built. If the system stopped a histogram for any of these reasons before the ?KHIST call, then ?KHIST takes the exception return with error codes ERHTT or ERHIP, respectively.

?PNAME

Get process name or PID.

?PNAME
exception return
normal return

Input/Output

Input:

AC0	byte pointer to process name or buffer.
AC1	PID; zero (returns PID in AC1); or -1 (returns caller's PID in AC1).

Output:

AC0	unchanged.
AC1	requested PID.

Exceptional Condition Codes in AC0

ERMPR	System call parameter address error.
ERPNM	Illegal process name.
ERPRH	Attempt to refer to a process not in the process tree.

Description

This call returns the process name associated with a given PID, the PID associated with a process name, or as an option, the calling process's PID and name.

If AC1 contains zero upon input, then AC0 must contain a byte pointer to a process name; the PID associated with this process name is returned in AC1.

If AC1 contains -1 on input, then the calling process's PID is returned in AC1. Additionally, if AC0 contains a nonzero byte pointer on input, then the associated current process's name is returned in the specified buffer.

If AC1 contains a PID on input, then the associated process name is returned in the buffer specified by AC0.

?PRIPR

Change the priority of a process.

?PRIPR
exception return
normal return

Input/Output

Input:

AC0	-1 to change the calling process's priority, or a byte pointer either to the process name of the process whose priority is to be changed, or to the PID of the process whose priority is to be changed.
AC1	-1 if AC0 contains a byte pointer, 0 if AC0 contains a PID; otherwise it is not meaningful.
AC2	new priority. Range: 1-255 for resident or pre-emptible processes, 1-3 for swappable processes.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERMPR	System call parameter address error.
ERPNM	Illegal process name.
ERPRP	Illegal priority. Caller attempted to specify a priority greater than its initial priority, and caller lacks privilege ?PVPR, or priority was out of range.
ERPRH	Attempt to refer to a process that is not in the process tree.

Description

This system call permits the calling process to change its own priority or that of any subordinate process, or, if the caller is in superprocess mode, any process in the hierarchy. The caller may raise any process's priority up to, but not greater than, the priority the target process had when it was created, unless the caller has privilege ?PVPR or is in superprocess mode. The range of priorities extends from 1 (highest) through 3 (lowest) for swappable processes, or from 1 through 255 for other types. See the beginning of this chapter for a description of how a process's priority affects its execution.

?PROC

Create a process.

?PROC *[parameter packet]*
exception return
normal return

Input/Output

Input:

AC2 starting address of the ?PROC parameter packet.

Output:

AC1 PID of the created process.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERPRV Caller not privileged to make this call.

ERPRP Illegal priority. Caller attempted to specify a process priority greater than his own, and he was not privileged to do so.

ERBMX Attempt to create a process with an illegal maximum size. The total amount of allocated memory in the process tree cannot exceed the limit of the initial process.

ERPTY Illegal process type. Attempt to create a process which is of a different type than the caller, when the caller lacks privilege ?PVTY.

ERPDF Error in process's User Status Table definition.

ERPRN Attempt to create more than the maximum number of processes.

ERSMX Error in setting maximum CPU time limit.

ERTLM CPU time limit was exceeded.

FILE SYSTEM codes

Description

General Usage Considerations

Except for the processes which are created by the system when it is first bootstrapped, all processes are created by a call to ?PROC.

Each process is limited in its creation capabilities by the privileges which it was assigned when it was created. A process may create another process and compete with it for system resources, or it may become blocked until its offspring process is terminated. All offspring of a terminated process also are terminated.

All the specifications of a process to be created are passed in the parameter packet whose address is in AC2 when the call ?PROC is issued. If this packet contains specifications which the caller is not privileged to make, then the system signals an error condition and control returns to the call's exception return. The structure of that parameter packet is shown in Figure 2-6, and offsets are described in Table 2-4. Settings indicated by parentheses obtain default values.

?PROC (continued)

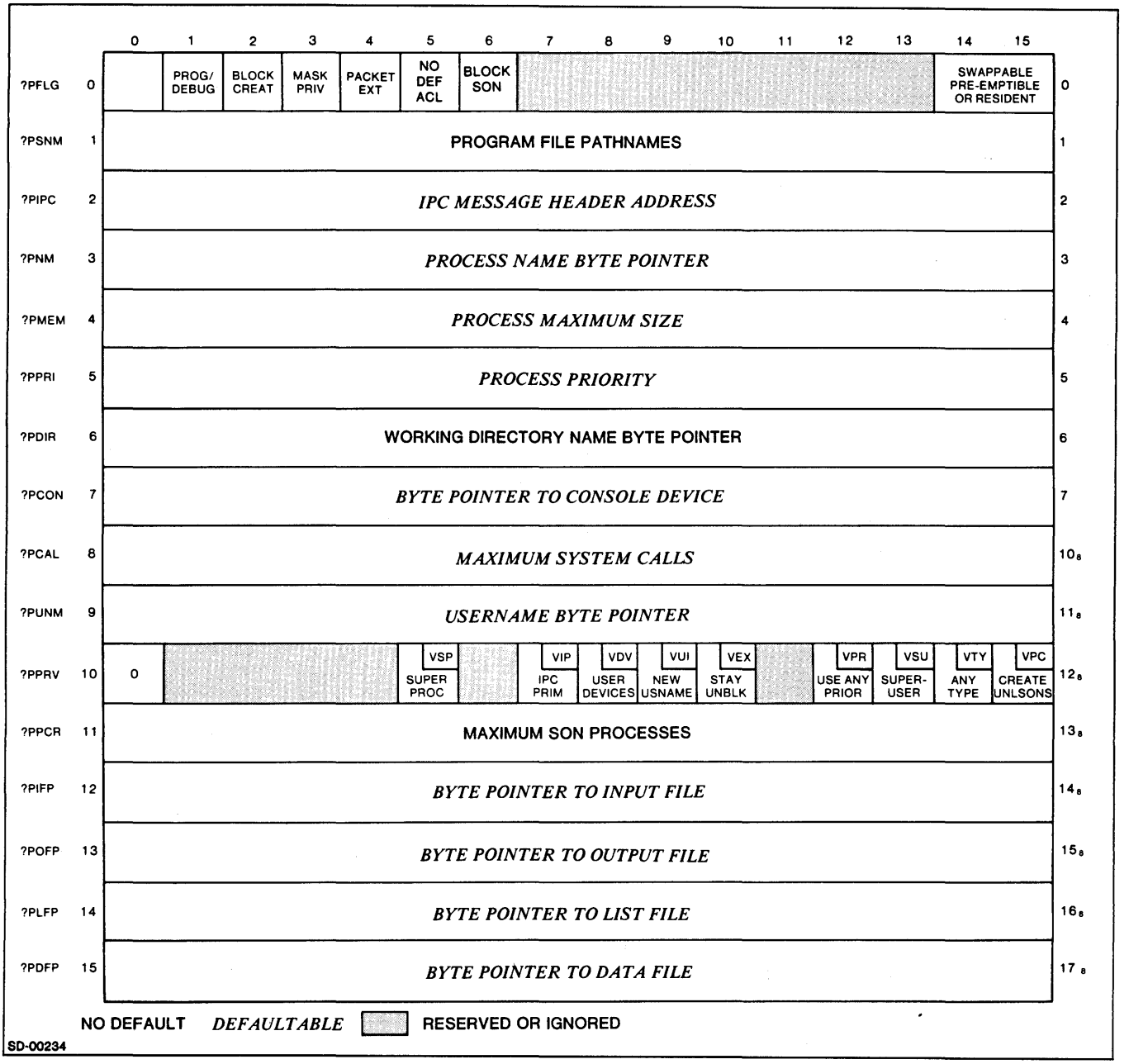


Figure 2-6. ?PROC Parameter Packet

Table 2-4. ?PROC Parameter Packet

Offset	Contents	Default
?PFLG	Miscellaneous process creation specifications:	
	packet extension ?PFPX - indicates the existence of a packet extension.	(0) No packet extension.
	privileges mask ?PFPM - mask off son's privilege.	(0) The son process receives all privileges selected in ?PPRV.
	control directive ?PFDB - control goes to the debugger.	(0) Control goes to the starting address of the program file.
	concurrency ?PFEX - caller in blocked util son is terminated. ?PFBS - son process is blocked immediately upon creation.	(0) Son runs concurrently with caller.
	process types ?PFRP - pre-emptible process ?PFRS - resident process	(0) Son is a swappable process.
	access privileges ?PFDA - do not pass default ACL to son.	(0) Son's default ACL will be same as caller's.
?PSNM	Byte pointer to pathname of program file to be executed in the created process.	No default.
?PIPC	Address of an IPC message header. This message is sent to the created process.	(-1) No message header.
?PNM	Byte pointer to the simple name of the process.	(-1) The son's name will be the ASCII representation of its PID. 3 digits including leading zeros, if necessary.
?PMEM	Maximum size of the new process in 2048-byte pages. This size includes the sizes of the shared and unshared user context and dynamic system overhead.	(-1) Same maximum size as the caller's.
?PPRI	Priority of the created process.	(-1) Same as caller's priority.
?PDIR	Byte pointer to the name of the offspring's working directory; or -1 to indicate the same as the caller's initial working directory, or 0 to indicate the caller's current working directory.	(-1) Same as caller's initial working directory.
?PCON	Byte pointer to the name of the device to be associated with the son process's CONSOLE. If 0 is specified in this offset, then no device will be associated with CONSOLE. See Chapter 6 for a discussion of CONSOLE and other generic filenames.	(-1) The caller's CONSOLE file will be given to the son.
?PCAL	Maximum number of system calls that the created process can issue concurrently.	(-1) Two calls.
?PUNM	Byte pointer to the son's username.	(-1) Son bears the username of the caller. (You must default this if you do not have privilege ?PVUI.)

(continues)

Table 2-4. ?PROC Parameter Packet

Offset	Contents	Default
?PPRV	Son process's privileges word; each bit of this word specifies a unique privilege to which the process can be entitled. (Setting each bit entitles the son process to that particular privilege unless ?PFPM is selected.) Privileges are described in detail following this summary description of the ?PROC parameter packet.	(-1) Son process has no privileges, if ?PFPM is also selected. Son process has all of caller's privileges, if ?PFPM is not selected.
?PPCR	Maximum number of son processes that can be created. 0 implies that no subordinate process can be created. -1 implies that the son may create the maximum number of processes which the father is currently entitled to create (after deducting one from this count for the newly created son). This value is overridden by privilege ?PVPC.	No default.
?PIFP	Byte pointer to the pathname of the file to be associated with the created process's INPUT generic filename. 0 specifies that no file will be associated with INPUT.	(-1) The father's INPUT file will be given to the son.
?POFP	Byte pointer to the pathname of the file to be associated with the created process's OUTPUT generic filename. 0 specifies that no file will be associated with OUTPUT.	(-1) The father's OUTPUT file will be given to the son.
?PLFP	Byte pointer to the pathname of the file to be associated with the created process's LIST generic filename. 0 specifies that no file will be associated with LIST.	(-1) The father's LIST file will be given to the son.
?PDFP	Byte pointer to the pathname of the file to be associated with the created process's DATA generic filename. 0 specifies that no file will be associated with DATA.	(-1) The father's DATA files will be given to the son.

(concluded)

Contents of accumulators passed to the newly created process are indeterminate; the father process can pass an IPC message to its son.

A process's initial search list is the same as its father's at the time of creation.

A process must specify in ?PFLG that its offspring has a process type which is the same as its own unless he has been privileged to do otherwise (privilege ?PVTY). ?PFPM, the mask bit in ?PFLG, causes the son process to receive only those privileges which the caller has and which were not selected (i.e., set to 1) in the privileges word, ?PPRV. Thus setting ?PFPM while placing -1 in ?PPRV will give no privileges to the son. (Not selecting ?PFPM and setting ?PPRV to 0 has the same effect.) For example, to send some subset of the caller's privileges, if ?PFPM is set, the caller's binary privileges word is 0...1011, and the privileges word in the ?PROC packet is 0...1, then the son receives the privileges word 0...1010.

The system passes the father's default access privileges to the son (set with the ?DACL call) except under the following conditions:

1. The ?PROC packet gives the son a different username than the father (offset ?PUNM is *not* -1, the default),
2. Bit ?PFDA in ?PFLG is set (do not pass default access privileges), or
3. The father process does not have the default access privileges (no ?DACL).

Given any one of these conditions, the system assigns the son the system default access privileges: <username> <0> <OWARE> <0> <0>. The father's access privileges remain unchanged.

The flags ?PFRP and ?PFRS in ?PFLG specify whether the son process will be pre-emptible or resident, respectively. These two flags are mutually exclusive. If you set both, the system returns the error code ERPTY (illegal process type). When you create a resident process, take care that its maximum memory requirements do not exceed physical memory, since this could cause a system deadlock.

If you send an IPC message to the created process, the new son will be able to retrieve the message with the ?GTMES call (documented in Chapter 9). You can use request type ?GMES to retrieve a message that is not in CLI format. In order to use other request types of the ?GTMES call, the message should be in CLI format. This means the following conditions must be met:

1. All arguments must be separated by single commas only.
2. The last character must be a null, and the message must be word aligned, i.e., filled out with a second null if necessary to insure an even number of bytes.
3. The following characters must not occur in the message: Space < > [ ] ( ); NEW LINE, carriage return, form feed, or any imbedded nulls.
4. The high order bit of every byte must be zero, i.e., not used for parity. (This will be done automatically if your message was coded with the .TXT pseudo-op or read from a system-controlled peripheral device.)

To send an initial IPC message, set the contents of the IPC header as follows:

?IUFL — Set flag ?GFCF of this word to tell the system that the message is in CLI format and that the first argument is argument zero, the program name.

?ILTH — Message length in words.

?IPTR — Word address of message.

Then, set all other words in this packet to zero.

For a discussion of IPC headers relevant to offset ?PIPC, consult Chapter 4.

Parameter ?PCAL specifies the maximum number of system stacks that can be potentially allocated for call processing at any one time. (Many calls, e.g., ?RDB/?WRB, discussed in Chapter 13, do not require stacks.) A fixed number of system stacks are available for use by all processes. Generally speaking, more critical processes will specify a greater number of possible concurrent calls. However, since the maximum number of concurrent system calls is a finite resource, processes should be allowed to have no more concurrent calls than is necessary. All specifications should be proportioned to actual needs as closely as possible; to set this parameter to a value that exceeds the number of tasks which will concurrently run in the process has no meaning.

?PROC (continued)

The following privilege bits and their symbolic names are defined for the privileges word, ?PPRV. We describe these earlier in this chapter, and repeat them here for your convenience.

Name	Meaning
?PVPC	A process with this privilege may create an unlimited number of son processes. The default value for ?PVPC is zero.
?PVEX	This privilege permits the caller to create a process and remain unblocked (i.e., not select ?PFEX in the ?PROC packet). Without this privilege, a process must become blocked when it creates a son process.
?PVPR	This privilege permits a process to change its own priority to a value higher than its initial priority, or, when it creates another process, to create it with a higher priority. It also allows a process to change the priority of a subordinate.
?PVSP	This privilege permits a process to issue ?SUPROC and enter superprocess mode.
?PVTY	This privilege permits a process to create another process of any type (e.g., resident). This privilege also permits a process to change the type that it was assigned when it was created. Without this privilege, it cannot change its own type or that of any of its sons.
?PVUI	This privilege permits a process to create another process which has a username differing from that of its father. Without this privilege, a process must have the same username as its father.
?PVDV	This privilege permits a process to access user devices (i.e., to issue system calls ?IDEF, ?DEBL, and ?STMAP).
?PVIP	This privilege permits a process to issue the primitive IPC calls ?ISEND and ?IS.R.
?PVSU	This privilege permits a process to issue ?SUSER and to enter the superuser mode.

Note again that a caller can pass along only those privileges which he possesses. Thus, setting ?PPRV to -1 and not selecting ?PFPM would give to the son all of the caller's privileges, but *only* those privileges.

Creating a Process with a Maximum CPU Usage Limit

A father process may assign a maximum number of CPU seconds to a son process and to all of its subordinate processes. If a son or subordinate process runs longer than the allocated time, it is terminated as a system error. The father process receives the error code ERTLM, indicating that the process has used up its CPU time limit.

You specify the time allocated to the son process as a double precision unsigned integer representing the maximum number of milliseconds a process may run before the system terminates it. You define the number of CPU milliseconds in an extension to the ?PROC parameter packet, and you define the existence of this extension in ?PROC packet offset ?PFLG by the bit ?PFPX. The ?PROC packet, plus extension, is ?PXLT words long; the last two words are ?SMCH and ?SMCL, respectively. (The standard ?PROC packet is ?PLTH words long.) In order to assign a maximum CPU runtime to a process, you must set the extender bit (?PFPX) at the same time you define the maximum number of milliseconds in the ?PROC packet extension. Offsets ?SMCH and ?SMCL in the extension define the high- and low-order parts, respectively, of the double precision integer which represents CPU time in milliseconds. Table 2-4 shows the ?PROC parameter packet. Offsets to the ?PROC parameter packet extension are shown in Table 2-5.

Table 2-5. ?PROC Parameter Packet Extension

Offset	Contents	Default
?SMCH	Maximum CPU time (in milliseconds) allotted this process (high order bits).	(-1) The son receives remainder of the father's CPU time.
?SCML	Maximum CPU time (in milliseconds) allotted this process (low order bits).	(0) If the father has no time limit, set no time limit for the son; if father has a time limit, set no time limit for son.

Guidelines for Using the CPU Usage Limit Capability

If a process has a time limit and creates a son process, the son process is limited to the time the father process has left. If the father process creates a son process with no time limit, the son process is given the maximum CPU time which the father process has left (i.e., the father process's maximum time limit minus the time it has used so far).

If the father process tries to assign more time to a son process than the father process has left, the ?PROC system call fails and returns the error code ERSMX to indicate an error in setting the maximum CPU time.

If the father process with a time limit is creating a son process with the maximum CPU time possible, then you should set the requested maximum CPU time to -1 (i.e., set ?SMCH and ?SMCL to -1). All the remaining time allocated to the father process is allowed for the son process.

If ?SMCH and ?SMCL are set to zero, then if the father process has no time limit, no time limit is set for the son process; if the father process (with no time limit) creates a son process with no time limit, no special action is taken.

When a father process has a time limit and creates a son process, the father must block when the son terminates (i.e., must select ?PFEX in ?PROC packet). When the son terminates, the CPU time used by the son is credited to the maximum time the father has left and the father is unblocked.

?PSTAT

Return status information on a process.

?PSTAT *[packet address]*
exception return
normal return

Input/Output

Input:

AC0	PID, -1 for calling process, or byte pointer to process name.
AC1	-1, if AC0 contains a byte pointer.
AC2	address of the area to receive status information.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERPRH Attempt to access a process which is not subordinate in the process tree.
FILE SYSTEM codes

Description

This system call returns status information for any process. The information is supplied in a packet, the location of which the calling process specifies. The packet size is ?PSLTH. Its structure is as follows:

Offset	Contents
?PSFP	PID of the father process.
?PSSN	Bit array indicating the PIDs of all son processes. Array length equals ?PSAL words. The system sets 1 bit in the array for each of the process's sons. The positions of these bits correspond to the PIDs of the sons. For example, if bit 13 in the array is set, then PID 13 is a son of the calling process.
?PSNR	Number of tasks in this process currently suspended on an ?IREC.
?PSNS*	Reserved.
?PSST*	Reserved.
?PSQF*	Priority queue factor.
?PSFL*	Reserved.
?PSF2*	
?PSF3*	
?PSPR	The process's priority.

?PSBK	Number of pages in the process's unshared area.
?PSPS	Number of pages in the process's shared area.
?PSSF	Starting page of the process's shared area.
?PSMB*	Number of pages in the process's system unshared area.
?PSYS*	Number of pages in the process's system shared area.
?PSYT*	Starting page of the process's system shared area.
?PSSW*	Number of shared pages loaded when the process image was last loaded.
?PSPV*	Process's privilege bits.
?PSEX*	Current time slice exponent for target process.
?PSF4*	Reserved.
?PSPD	Target process's PID.
?PSMX*	Maximum number of pages allowed this process, including system pages.
?PSSL*	Number of sub-slices in the process's time slice.
?PSRH ?PSRL	Time elapsed since process creation (in seconds).
?PSCH ?PSCL	CPU time used by this process (in milliseconds).
?PSIH ?PSIL	Number of blocks read or written.
?PSPH ?PSPL	Page usage over CPU time.

*These offsets in the ?PSTAT packet contain values taken from internal AOS databases; they are not expected to be useful to programmers external to Data General Corporation.

Note that words ?PSRH, ?PSRL, ?PSCH, ?PSCL, ?PSIH, ?PSIL, ?PSPH, and ?PSPL contain the same information as returned by the ?RUNTM system calls in words ?GRRH, ?GRRL, ?GRCH, ?GRCL, ?GRIH, ?GRIL, ?GRPH, and ?GRPL, respectively.

?RETURN

Terminate a process and return a message.

?RETURN
exception return

Input/Output

Input:

AC0 error code to CLI (optional).

AC1 byte pointer to message (optional).

AC2 high-order byte flags:

?RFCF Message is in CLI format

?RFEC Error code in AC0

Severity	Bits
?RFWA	Warning
?RFER	Error
?RFAB	Abort

low-order byte:

length (in bytes) of message.

Output:

None, since there is no normal return from this call.

Exceptional Condition Codes in AC0

ERMPR System call parameter address error.

Description

The ?RETURN call terminates the calling process and transfers control to its father. The caller may also send a message to its father by placing a byte pointer to the message in AC1 and the length of the message in the low-order byte of AC2. If the CLI or EXEC is the father, ?RFCF is usually set in AC2 and the message must be in CLI format. (See the *AOS CLI User's Manual* for a description of CLI format.) If the father is not the CLI or EXEC, then the message may be in any format agreed upon by the father and son. If no message is sent, the low-order byte of AC2 must be set to 0. In this case, AC1 is ignored. The termination message is sent to the father as an IPC on global port ?SPTM (10_8). (See Chapter 4 for a description of the IPC facility.)

If the caller is terminating due to an exceptional condition and its father is the CLI or EXEC, then the caller may also specify the severity of the exceptional condition via high-order byte flags in AC2 and place an error code in AC0. (See example in Figure 2-11 at the end of this chapter.) The severity field, ?RFWA, ?RFER, or ?RFAB, directs the father to process the outstanding exceptional condition as a warning, error, or abort, respectively. The actual severity level (0, 1, 2, or 3) at which the CLI and EXEC is currently processing warnings, errors, and aborts is not affected by this call. (See the *AOS Command Line Interpreter User's Manual* for a description of severity levels.)

If the calling process's father is the CLI and the caller asserts an exceptional condition flag, then the CLI will display on the user console the message *WARNING*, *ERROR*, or *ABORT*. If the caller passes an error code in AC0, the CLI will display a system generated error message on the console. Finally, if the caller passes a user-message to the CLI, it will be displayed on the console.

In all cases, if the calling process is returning under normal conditions, AC2 must be set to either 0 or ?RFCF.

?RUNTM

Get process runtime statistics.

?RUNTM *[packet address]*
exception return
normal return

Input/Output

Input:

AC0	PID, -1 (for this process), or byte pointer to process name.
AC1	-1 if AC0 contains a byte pointer; otherwise AC0 contains a PID or -1.
AC2	address of area to receive statistics.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERPRH	Attempt to access a process which is not in the process tree.

FILE SYSTEM codes

Description

This system call returns certain runtime statistics for any process. These statistics are provided in a packet whose location is specified by the caller. The size of this packet is ?GRLTH, and its structure is shown in Figure 2-7.

Time elapsed since the process's creation is given in seconds, with a range from 0 to (2**32)-1. Elapsed CPU time is expressed in milliseconds. Page usage is given in page-milliseconds. The blocks recorded in ?GRIH/?GRIL are those blocks read or written via the data channel.

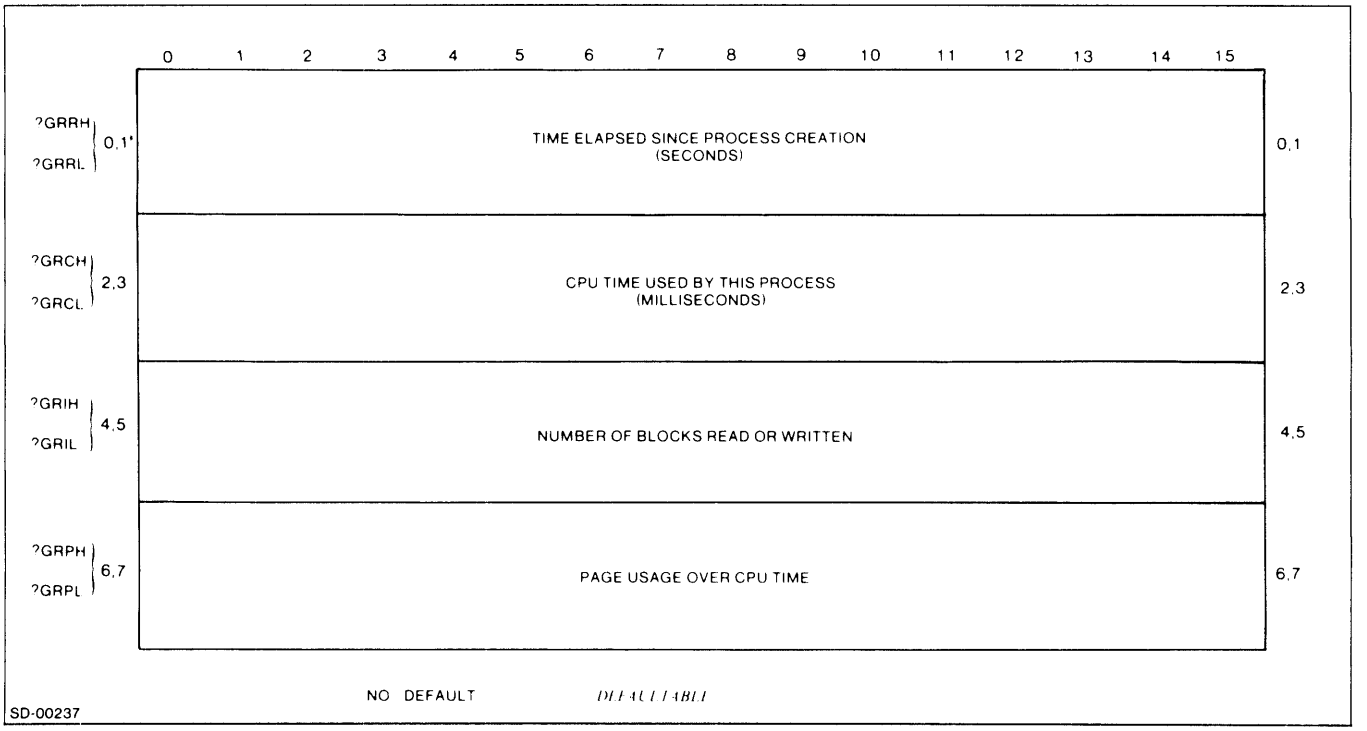


Figure 2-7. ?RUNTM Statistics Packet

?SUPROC

Change superprocess mode.

?SUPROC
exception return
normal return

Input/Output

Input:

AC0 -1 to turn on superprocess mode, +1 to turn off superprocess mode, 0 to return current superprocess mode.

Output:

AC0 the current (new, if changed) superprocess mode: -1, mode is on; +1, mode is off.

Exceptional Condition Codes in AC0

ERPRV Attempt to issue this call without having the privilege ?PVSP.

Description

This system call permits a process with the privilege ?PVSP to enter or leave the superprocess mode or to determine the mode the process is currently in.

A process in superprocess mode can issue a ?BLKPR, ?BRKFL, ?CYTPE, ?PRIPR, ?TERM, or ?UBLPR call for any process in the hierarchy, instead of merely those below it in its process tree. Superprocess mode also permits a process to create processes with any type or priority, and to change the type or priority of any process regardless of whether the calling process has privilege ?PVPR or ?PVTY.

?SUSER

Change superuser mode.

?SUSER
exception return
normal return

Input/Output

Input:

AC0 -1, turn on superuser mode; +1, turn off superuser mode; 0, return current superuser mode.

Output:

AC0 the current (new, if changed) superuser mode: -1, mode is on; +1, mode is off.

Exceptional Condition Codes in AC0

ERPRV Attempt to issue this call without having privilege ?PVSU.

Description

This call permits a process with privilege ?PVSU to enter or leave superuser mode. While in this mode, a process can access any file, bypassing the system's access control mechanism (see "Controlling Access to Files" in Chapter 5).

Any process in superuser mode remains in this mode until it issues ?SUSER to leave this mode or is terminated.

If access controls are disabled (an AOSGEN option), everyone is in superuser mode all the time. If ?SUSER is issued in an attempt to turn superuser mode off, the normal return is taken with -1 in AC0.

?TERM

Terminate a process.

?TERM
exception return
normal return

Input/Output

Input:

AC0	byte pointer to name of process to be terminated, PID of process to be terminated, or -1 to terminate the process issuing this call (self-termination).
AC1	-1 if AC0 contains a byte pointer, 0 if AC0 contains a PID; otherwise it is not meaningful.
AC2	address of IPC message header (0 for no header). The input contents of AC2 are valid only upon a self-termination.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERPRH	Attempt to refer to a process which is not in the process tree.
ERMPR	System call address error.
ERPNM	Illegal process name.

Description

?TERM can terminate the calling process, any subordinate process, or, if the caller is in superprocess mode, any process in the hierarchy. Moreover, the termination of any process also terminates any descendants of that process. If the ?TERM terminates the calling process (self-termination), it can also send a termination IPC message of any length to the caller's father.

It does not matter that any son may be created at a higher priority or a higher process type than its father. Any process can terminate any other process which occupies a position subordinate to it in the tree.

If the caller issues a ?TERM to terminate a process other than itself, control may return to the calling process before termination is complete. That is, the PID and the process name of the terminating process may still be in use when the caller regains control.

Upon self-termination, the system sends the IPC message specified in AC2 to the terminated process's father (thus the ports specified in the header are ignored). If no IPC header is specified in AC2, then the system sends a standard IPC message to the terminated process's father. In Chaper 4, see "Receiving Process Termination Messages" for procedures to receive and interpret these IPC messages.

?UBLPR

Unblock a process.

?UBLPR
exception return
normal return

Input/Output

Input:

AC0 byte pointer to name of process to be unblocked, or PID of process to be unblocked.

AC1 -1 if AC0 contains a byte pointer; 0 if AC0 contains a PID.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

ERPRH Attempt to refer to a process which is not in the process tree.

ERMPR System call address error.

ERPNM Illegal process name.

Description

This system call unblocks a process blocked by a previous ?BLKPR. If the process which is the target of this call has no tasks that are ready to run, then the process will revert to the blocked state. The target process must be subordinate to the process issuing this call, unless the caller is in superprocess mode.

Examples

Four example routines/programs follow to illustrate some of the more common process management calls. Figure 2-8 illustrates the first of these, a module entitled ERROR. This is a general-purpose routine called by a main program whenever it receives control from a system call's exception return.

```
 0001 ERROR AOS ASSEMBLER
                        .TITL ERROR
02                      .ENT ERROR
03
04              ; GENERAL PURPOSE ERROR HANDLER. CALLER INPUTS SYSTEM ERROR CODE
05              ; IN AC0.
06                              .NREL
07 00000'030404 ERROR:  LDA     2,FLAGS ;GET ?RETURN FLAGS
08                      ?RETURN         ;TERMINATE PROCESS
09 00003'000775         JMP     ERROR   ;REPORT ?RETURN ERROR INSTEAD
10 00004'150000 FLAGS:  ?RFCF+?RFER+?RFEC       ;REPORT ERROR IN AC0 TO CLI
11                      .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS


  0002 ERROR

ERROR 000000' EN     1/02    1/07    1/09
FLAGS 000004'        1/07    1/10
?RETU 002415  MC     1/08
?XCAL 000001         1/09
```

Figure 2-8. ERROR Routine

An exception code is passed in AC0 to this routine, which passes this code along with suitable flags to the process's father via ?RETURN. Sample routines in the following chapters will often call ERROR.

Figure 2-9 illustrates a routine entitled SON. This routine creates a swappable son process, defaulting nearly all entries in the ?PROC packet. IPC privileges are specified since we will use this routine in an example program found in Chapter 4.

```
 0001  SON       AOS ASSEMBLER REV 01.09          14:43:19 12/07/78
01
02                ;0001 SON       AOS ASSEMBLER
03
                                  .TITL SON
05                                .ENT  SON
06                                .EXTN ERROR
07
08                ;THIS ROUTINE CREATES A SWAPPABLE SON PROCESS. INPUT IN AC0
09                ;A BYTE POINTER TO THE PATHNAME OF THE PROGRAM WHICH IT WILL
10                ;EXECUTE (?PSNM). ALL DEFAULTABLE CHARACTERISTICS ARE
11                ;DEFAULTED: NO IPC HEADER, PROCESS NAME IS ASCII REPRESEN-
12                ;TATION OF ITS PID, SAME MAXIMUM SIZE AS THE CALLING PROCESS,
13                ;SAME WORKING DIRECTORY AS THE CALLING PROCESS, SAME GENERIC
14                ;FILES AND USERNAME AS THE FATHER, SAME PRIVILEGES AS THE
15                ;FATHER, NO SONS.
16
17                                .NREL
18 00000'163710 SON:    SAVE 0
19       000000
20
21 00002'040411          STA     0, PKT+?PSNM     ;STORE IN PROCESS PKT.
22                       ?PROC   PKT
23 00007'002402          JMP     @.ERROR
24 00010'127710          RTN
25 00011'000000$.ERROR: ERROR
26 00012'000020 PKT:     .BLK    ?PLTH
27       000012'         .LOC    PKT+?PFLG
28 00012'000000          0
29       000013'         .LOC    PKT+?PSNM
30 00013'000000          0
31       000014'         .LOC    PKT+?PIPC
32 00014'177777          -1
33       000015'         .LOC    PKT+?PNM
34 00015'177777          -1
35       000016'         .LOC    PKT+?PMEM
36 00016'177777          -1
37       000017'         .LOC    PKT+?PPRI
38 00017'177777          -1
39       000020'         .LOC    PKT+?PDIR
40 00020'177777          -1
41       000021'         .LOC    PKT+?PCON
42 00021'000000          0
43       000022'         .LOC    PKT+?PCAL
44 00022'177777          -1
45       000023'         .LOC    PKT+?PUNM
46 00023'177777          -1
47       000024'         .LOC    PKT+?PPRV
48 00024'000400          ?PVIP
49       000025'         .LOC    PKT+?PPCR
50 00025'000000          0
51       000026'         .LOC    PKT+?PIFP
52 00026'000000          0
53       000027'         .LOC    PKT+?POFP
54 00027'000000          0
```

Figure 2-9. SON Routine (continues)

```
55        000030'          .LOC    PKT+?PLFP
56 00030'000000            0
57        000031'          .LOC    PKT+?PDFP
58 00031'000000            0
59        000032'          .LOC    PKT+?PLTH
60                 ;END OF PACKET


 0002  SON
01                         .END

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS


 0003 SON

ERROR 000001  XN     1/06    1/25
PKT   000012'        1/21    1/23    1/26    1/27    1/29    1/31    1/33
                     1/35    1/37    1/39    1/41    1/43    1/45    1/47
                     1/49    1/51    1/53    1/55    1/57    1/59
SON   000000' EN     1/05    1/18
.ERRO 000011'        1/23    1/25
?PROC 005147  MC     1/22
?XCAL 000001         1/23
```

Figure 2-9. SON Routine (concluded)

TSTPRC and TERMINATOR, shown in Figures 2-10 and 2-11, are two programs (each using ERROR shown in Figure 2-8) which form a single example.

```
 0001 TSTPR AOS ASSEMBLER
                            .TITL   TSTPRC
02                          .EXTN   ERROR
03
04                  ;THIS PROGRAM CHAINS TO A NEW PROGRAM, "TERMINATOR.PR",
05                  ; WHICH TERMINATES ITSELF AND OUTPUTS A CLI MESSAGE
06                              .NREL
07
08 00000'020405 START:  LDA     0,.TERM
09 00001'126400         SUB     1,1             ;PASS CONTROL TO TERM, NOT DEBUGGER
10                      ?CHAIN
11 00004'002411         JMP     @.ERROR
12 00005'000014".TERM:  .+1*2
13 00006'052105         .TXT "TERMINATOR.PR"
14       051115
15       044516
16       040524
17       047522
18       027120
19       051000
20 00015'000000$.ERROR: ERROR
21                      .END    START

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS



  0002 TSTPR

ERROR 000001  XN     1/02     1/20
START 000000'        1/08     1/21
.ERRO 000015'        1/11     1/20
.TERM 000005'        1/08     1/12
?CHAI 002336  MC     1/10
?XCAL 000001         1/11
```

Figure 2-10. TSTPRC Routine

```
 0001 TERMI AOS ASSEMBLER
                        .TITL   TERMINATOR
02                      .EXTN   ERROR
03                      .ENT    TERM
04
05              ; THIS PROGRAM IS CHAINED TO BY "TSTPRC.PR".
06              ;THIS PROGRAM SENDS A CLI MESSAGE AND TERMINATES ITSELF.
07
08                              .NREL
09
10 00000'024405 TERM:   LDA     1,MSG
11 00001'030430          LDA     2,FLAG
12                      ?RETURN
13 00004'002426          JMP     @.ERROR
14 00005'000014"MSG:    .+1*2
15 00006'052110          .TXT    "THIS IS YOUR TEST PROCESS SIGNING OFF"
16       044523
17       020111
18       051440
19       054517
20       052522
21       020124
22       042523
23       052040
24       050122
25       047503
26       042523
27       051440
28       051511
29       043516
30       044516
31       043440
32       047506
33       043000
34 00031'100046 FLAG:   ?RFCF+46        ;MESSAGE IS IN CLI FORMAT, 23 CHARS. LONG
35 00032'000000$.ERROR: ERROR
36                      .END    TERM

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS


  0002 TERMI

ERROR 000001  XN     1/02    1/35
FLAG  000031'        1/11    1/34
MSG   000005'        1/10    1/14
TERM  000000' EN     1/03    1/10    1/36
.ERRO 000032'        1/13    1/35
?RETU 002415  MC     1/12
?XCAL 000001         1/13
```

Figure 2-11. TERMINATOR Routine

In these figures, TSTPRC chains to a new program, TERMINATOR.PR. TERMINATOR, in turn, simply terminates itself (?RETURN) and returns a CLI message to indicate a successful termination.

End of Chapter

Chapter 3
Memory Management

The maximum main memory size of AOS is 2 megabytes, structured in fixed pages of 2048 bytes. The system analyzes global memory requirements and dynamically allocates memory to itself and all other user processes in the most efficient way. When each process is created by the ?PROC system call, one of the parameters specified is the number of pages (up to 32_{10}) of main memory that will be available to the process.

A process's memory space, called its *context,* is divided into two categories: shared and unshared. Shared pages, which are usually write-protected, contain code or data available concurrently to this and other processes. Unshared pages can be used only by the process into whose context they are linked. Each context consists of at least one unshared page containing lower page zero (ZREL) and several system databases. The unshared area always starts at the first word of the process's context. If a shared area exists, Link will place it at the top of shared address space (octal 77777) unless an alternate area is specified.

Figure 3-1 is a simplified diagram of a complete user context. In a typical dynamic user context, the shared area will grow downwards and the unshared area will grow upwards.

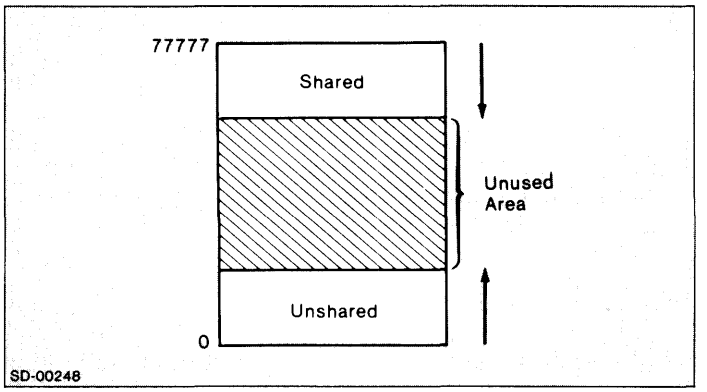


Figure 3-1. Used and Unused Areas in a Context

The operating system provides calls for managing the allocation of memory pages to a process. The following calls are used for memory management of nonshared pages:

Call Name	Purpose
?MEM	List the maximum number of unshared pages available, the number of unshared pages used, and the highest currently used unshared address for a context.
?MEMI	Increase or decrease the number of unshared pages in a context.

You may use the ?ZMAX and ?NMAX symbols, produced by Link, to determine initial usage of ZREL and NREL memory.

The following additional system calls manage shared pages:

Call Name	Purpose
?GSHPT	List the current shared partition size for a context.
?RPAGE	Release a shared page.
?SOPEN	Open a file for shared page reading.
?SPAGE	Perform a shared page read.
?SSHPT	Establish a new shared partition size.

Shared Page Concepts

Pages of memory can be shared among processes, depending on their access privileges (see Chapter 5). The manipulation of shared memory pages is important in memory management.

The operating system constructs a cache-like collection of shared pages which are not currently in use. If a released page is required again soon, and the system does not need the page's memory for other uses, then it can be remapped into a process's shared area without any disk I/O.

There are five methods of utilizing shared memory in the operating system.

- By defining a shared area to Link.
- By explicit shared page management, through system calls ?SPAGE and ?RPAGE.
- By implicit shared page management, using a special form of the ?OPEN system call.
- With shared overlays.
- With shared routines.

Shared routines and shared overlays are described later in this chapter.

One convenient means for designating shared areas in an assembly language program file is the .NREL pseudo-op. When .NREL is given a nonzero argument, code and data which follow will be bound into shared areas of a program file. Shared areas of this file will be mapped into the context of each process using the file. No prologue is required for the program to achieve this sharing; no system call such as ?OPEN is required.

If the process uses explicit calls ?SPAGE and ?RPAGE for memory management, then it can read a continuous 1024-word (2048-byte) section of a disk file into one of its shared pages. The system maintains a list of all pages for each file and also records the current use count of each page. The use count is incremented each time a process performs a shared read of a page. The use count will be decremented when the page is released.

The system also maintains a chain that lists, in least-recently-used (LRU) order, all formerly shared pages (each with a use count of zero) still residing in main memory. These pages are candidates for reuse should main memory be required.

The system call to read a page into shared memory is ?SPAGE; the call to release such a shared page (without reading a new page) is ?RPAGE.

System call ?FLUSH writes the contents of a shared page out to disk (but it does not release the page).

To clarify these concepts, study Figure 3-2, *The Dynamics of Page Sharing*. This illustration depicts the effect of shared page reading and releasing upon free memory, use counts, the file page list, and the LRU chain.

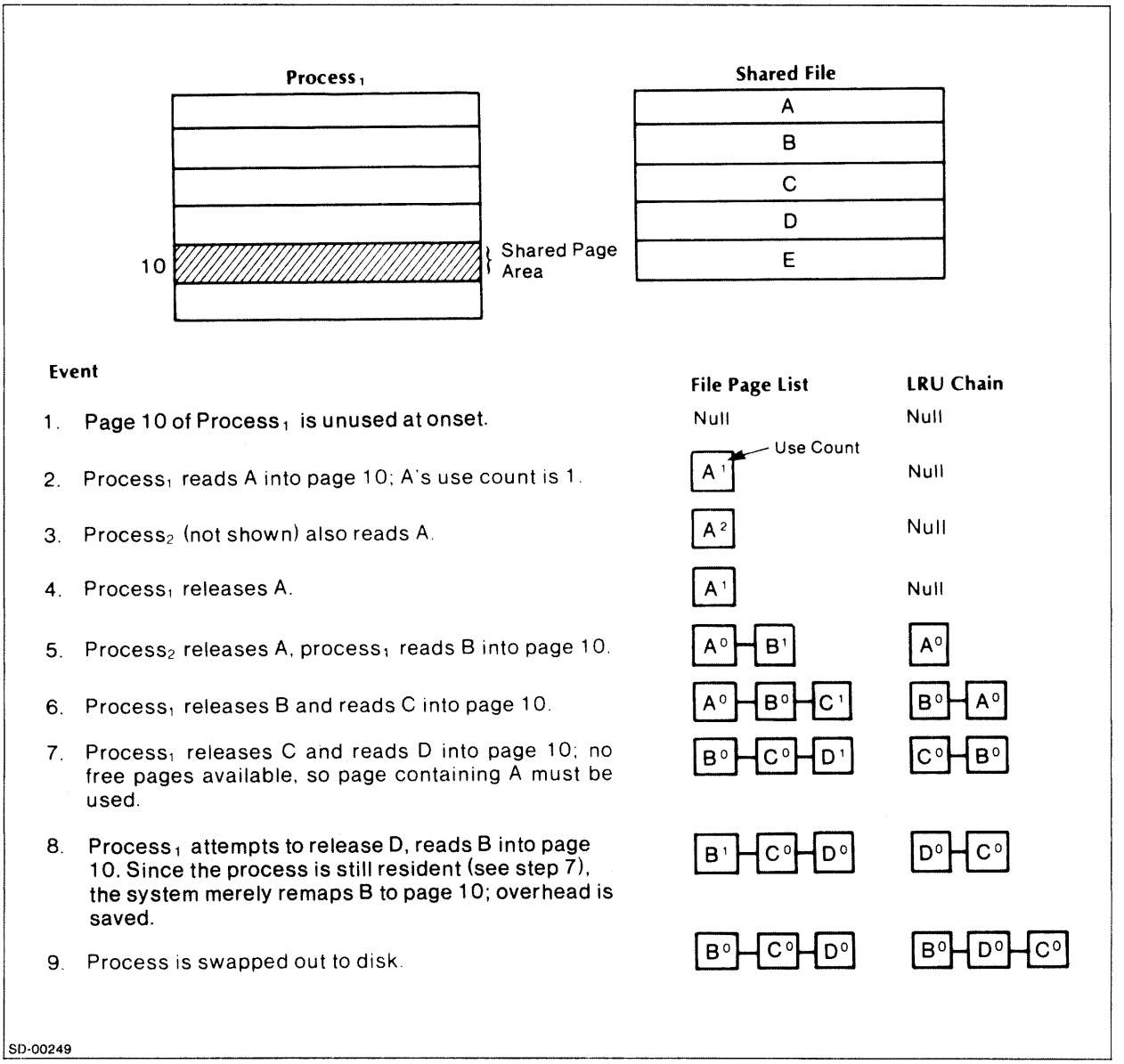


Figure 3-2. The Dynamics of Page Sharing

Shared File Facility

As an alternative, you can cause the system to read portions of a file into a shared page implicitly by setting the ?SHOP bit in the ?OPEN packet, issuing the ?OPEN system call, and then accessing the file via the ?READ and ?WRITE system calls. These implicitly created shared pages are kept within system address space. Thus, shared file accesses using ?OPEN/?READ/?WRITE do not affect the shared area of the calling process in any way. The system maintains these shared pages by automatically performing ?SOPEN, ?SPAGE, ?RPAGE, and ?SCLOSE system calls on behalf of the user process.

This facility allows several processes to share data in a file while accessing that data on a record by record basis. For details on using this facility, along with restrictions on its use, see Chapter 6.

The Effect of Shared Pages on System Overhead

Clearly, the judicious use of shared page calls (i.e., timing them to occur most frequently when the desired pages are resident in main memory) results in the least system overhead, since only a remap operation--as opposed to disk I/O--is required to effect the shared page transfer. Also, the more free memory pages available in a system, the greater the probability of a page being resident. If a page desired by a shared read is already resident in main memory, then system overhead is reduced. Instead of performing a disk read of some multiple of four disk blocks, the system merely remaps one or more pages of memory into the shared space.

If a process needs to release a shared page's contents and perform a new shared read into its partition, then only the shared read call, ?SPAGE, need be issued. If the process wants to release a shared page's contents and not read in a new page, then ?RPAGE alone is issued. ?RPAGE decrements the use count for the released page. ?SPAGE increments the use count of the acquired page and decrements the use count for the released page.

Swappable processes are read in and out to disk transparently to the user. When a process is swapped out to disk, the use count of all its shared pages are decremented (as when a shared page is released) and those pages with a use count of zero are placed at the end of the LRU chain. Shared pages in a partition for which read-only access was declared will not be swapped out to disk, however. Thus, specifying write accessibility only when it is needed will decrease system overhead.

Modifying Memory with Disk Images

The operating system provides two ways for modifying the contents of a context: overlays and shared routines. Each of these constructs can be read into a context to provide more efficient use of memory. Overlay files are built by Link when the program file is formed. The Link utility links shared routines and then the Shared Library Builder collects them into libraries.

There are two types of calls for using and managing shared routines and overlays: resource calls and primitive calls. Most applications use resource calls because of their automatic operation and modular resource management. *Resource calls* automatically load and release procedures, thereby simplifying the management of a potentially complex environment. They also permit you to defer deciding where to place procedures (into overlays, shared routines, or the permanent root context) until Link time. In contrast, *primitive calls* provide greater control over overlays, but require explicit loading and release.

NOTE: You should only use primitive overlays if you are willing to properly control and manage them.

Overlay Concepts

Overlaying is a technique you use for bringing modules of code, called *overlays,* into main memory from disk. Overlaying permits several overlays to occupy the same contiguous area in main memory at different times, thus enhancing the efficient use of that memory.

The overlay file is divided into *overlay areas;* each area contains the entire set of overlays associated with one overlay area in main memory. Each overlay is padded out to be as large as the largest overlay in its overlay area. You must define overlays, and overlay areas in main memory are allocated in multiples of 512 bytes for unshared pages, and as multiples of 2048 bytes for shared pages.

There may be up to 63 overlay areas within a root, and thus up to 63 overlay areas within the overlay file. Each overlay area on the file can contain up to 511 separate overlays, and the largest overlay in each area determines the basic size of each overlay area in memory. It is possible to define a *total overlay area* whose size is some multiple of the basic overlay size. This permits several different overlays to reside simultaneously in *basic overlay areas* within the same total overlay area. Since the system may place these overlays into any of the basic areas within a total area, these overlays must be written relocatably (i.e., with code that is position-independent).

Figure 3-3 shows a single area of an overlay file and possible overlay loading, given a total overlay area whose size equals one overlay. The total area is determined by the largest overlay, overlay B.

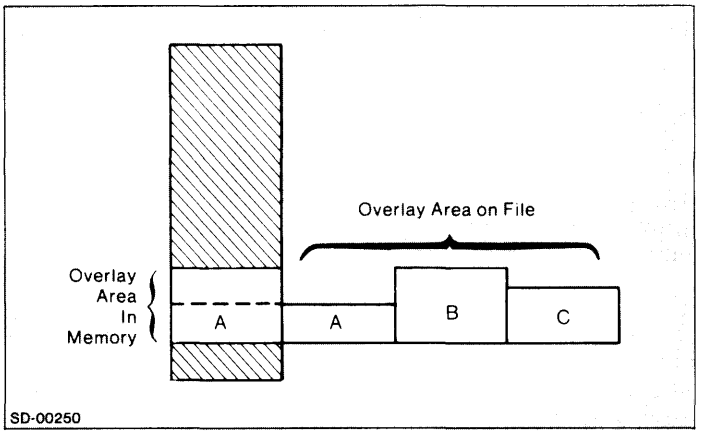


Figure 3-3. Total Overlay Area Size Equals Basic Area Size

Overlay A currently resides in the overlay area. Figure 3-4 shows what the overlay area looks like when the user has requested that the size be doubled. The basic size still equals the size of overlay B, but the area which has been allocated is twice this size. Figure 3-4 shows both overlay A and C residing in the total area.

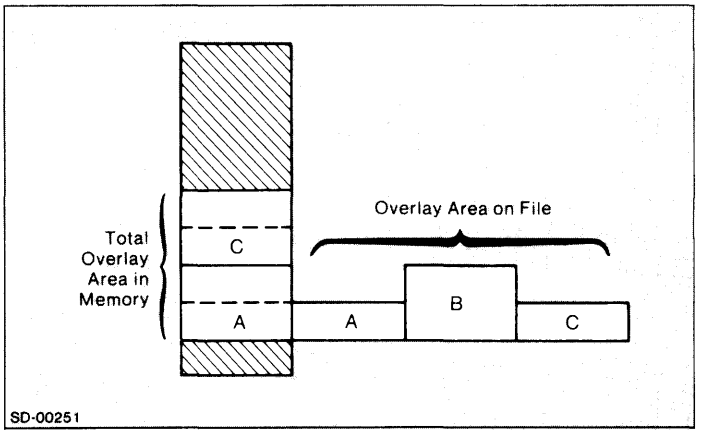


Figure 3-4. Total Overlay Area Equals Twice the Basic Area Size

Object files are formed into overlays by means of a command file which is named in Link command line. Although this technique is described in detail in the *AOS Link User's Manual*, a simple illustration is presented here:

```
XEQ LINK RO/C
```

where the *object module* RO contains the following:

```
RO [A,B,C,D] R1 R2 [E,F,G,H]
```

The command line creates an overlay file, RO.OL, with two overlay areas containing the binaries enclosed within each pair of brackets. If argument switch /H had been applied to any filename, then that overlay would have been loaded as a shared overlay. If the .NREL pseudo-op was used, then overlays would be loaded into the type of pages it specified. By default, overlay code is loaded into unshared pages.

A corresponding program file, RO.PR, is also built with two overlay areas. Each overlay area is large enough to accommodate the largest overlay.

The order in which overlay binaries are given in the command line determines both the overlay number and the overlay area number which will be associated with each overlay. In the example, A,B,C, and D become overlays 0 through 3 of overlay area 0. Likewise, E becomes overlay 0, F, and G become overlay 1, and H becomes overlay 2 -- all of overlay area 1.

The Link utility builds a 15-bit symbol which identifies each overlay within its overlay area. The overlay area number is specified in the left 6 bits of this symbol, and the overlay number is in the right 9 bits of this symbol. The .ENTO and .EXTN pseudo-ops (described in the *AOS Macroassembler Reference Manual* and in the *AOS Link User's Manual)* provide the mechanism for specifying overlay areas and numbers symbolically.

Figure 3-5 depicts the use of the .ENTO pseudo-op in the relocatable objects E, F, G, and H of the command file described above.

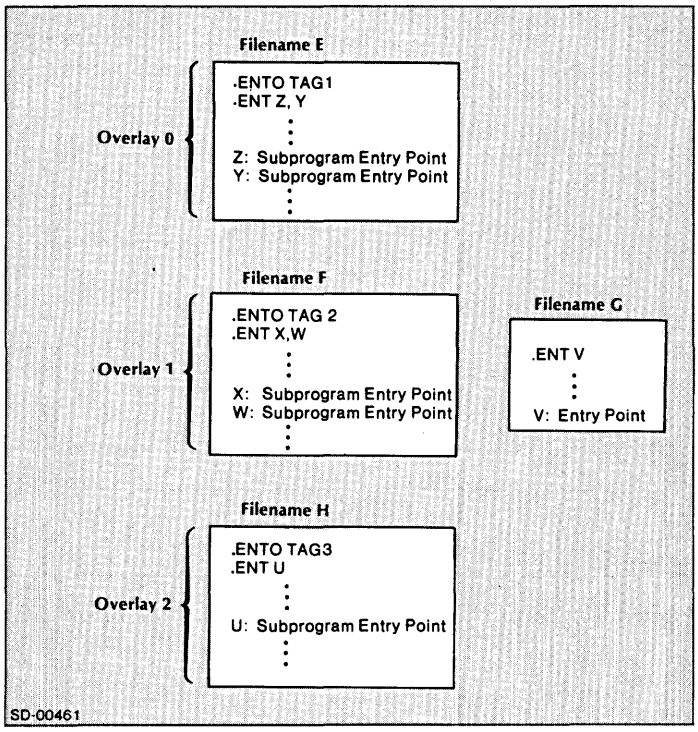


Figure 3-5. Area 1 of Overlay File RO.OL

To call these overlays from a program, locations containing the overlay area number and overlay numbers could be set aside as shown in the extract of RO.PR code that appears in Figure 3-6.

```
.EXTN  TAG1,TAG2,TAG3
.EXTN  Z,Y,X,W,V,U
.OV0:  TAG1              ;TAG1 IS
                         ;1000
.OV1:  TAG2              ;TAG2 IS
                         ;1001
.OV2:  TAG3              ;TAG3 IS
                         ;1002
.
.
.
LDA    0,.OV1            ;GET OVERLAY 1'S
                         ;NUMBER INTO AC0

overlay call             ;CALL OVL1
```

SD-02990

Figure 3-6. Overlay Call Illustration

In Figure 3-6, after an overlay load command is issued, all of overlay 1 in area 1 (including both F and G) is loaded, and routines V, W, and X can then be called from within the program.

Overlays in Different Partitions

The Link utility groups code and data from object modules into separate memory partitions according to information contained within the objects (put there by the operation of pseudo-ops or by special block types described in the *AOS Link User's Manual)* or within the Link utility command line. The Link utility builds six predefined partitions:

- absolute
- ZREL
- NREL unshared code
- NREL shared code
- NREL unshared data
- NREL shared data

These types are described in detail in *AOS Link User's Manual* and in the *AOS Macroassembler Reference Manual.* Two of these types permit the use of overlays: NREL unshared code and NREL shared code. There may be several different partitions of a single type; however, by default, only one partition of each type is created. If one of the objects within an overlay area contains a directive to Link to create a new partition, an overlay area is allocated immediately in that new partition. Such Link directives can be issued either by using the .NREL pseudo-op or by an Address Information Block.

Figure 3-7 shows how an overlay file and an overlay area would be constructed if all of the code in objects a,b,c, and d were destined for a single partition. This is similar to Figures 3-4 and 3-5. Figure 3-8 shows how the overlay file would be built if objects a and d contained code destined only for partition X, but b and c contained code destined only for partition Y. Finally, Figure 3-9 indicates how the overlay file would look if objects a, c, and d were to form overlays for partition X, but overlay b was to go to partition Y.

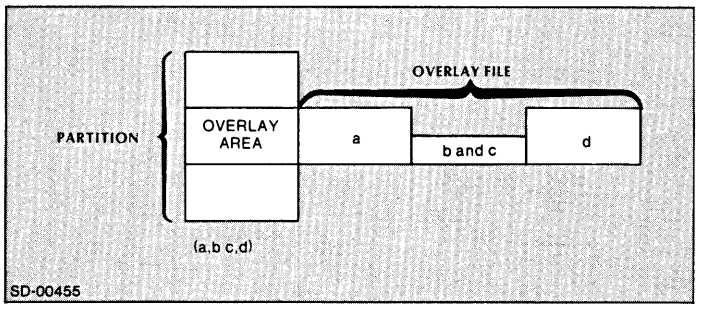


Figure 3-7. Overlay Area Built for a Single Partition

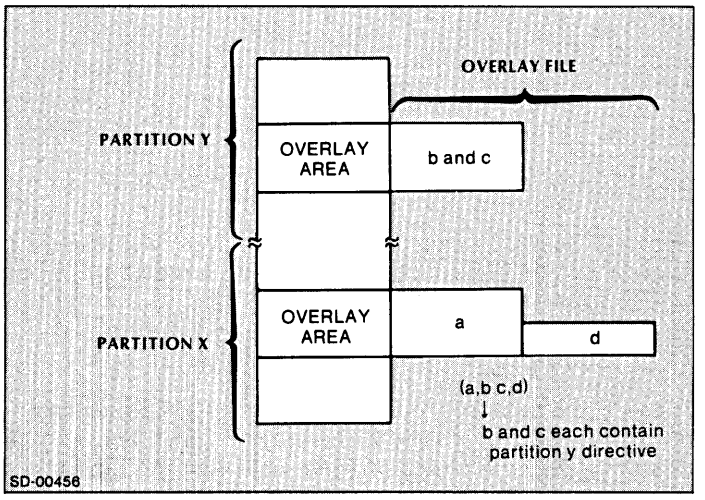


Figure 3-8. Overlay Areas with Two Partition Directives

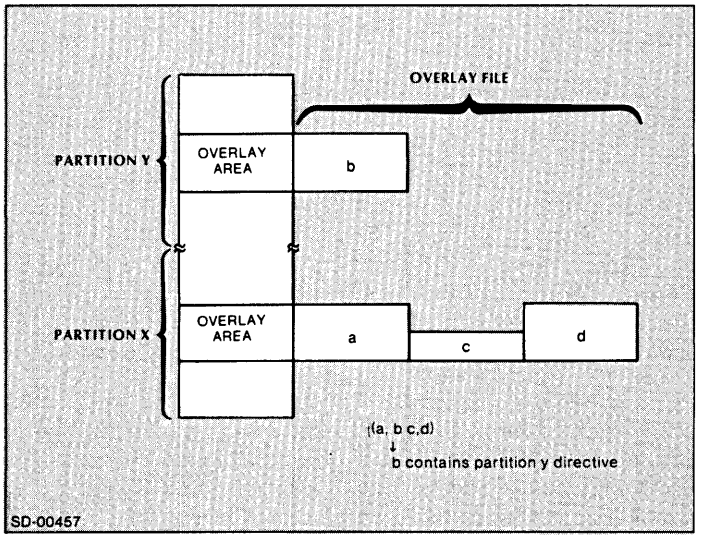


Figure 3-9. Overlay Areas with One Partition Directive

The Macroassembler and Link permit both shared and unshared code to be placed in the same object module. If such a module is placed in an overlay, the code will actually be split into two different overlays; shared code goes to the overlay in the shared partition, and unshared code goes to the overlay in the unshared partition. Whenever such a splitting of overlays occurs, you must use separate .ENTO statements in the two parts of the module, and each .ENTO statement must follow immediately its associated .NREL statement.

Figure 3-10 shows a module, OVLA, which becomes split into two overlays in two partitions when it is bound.

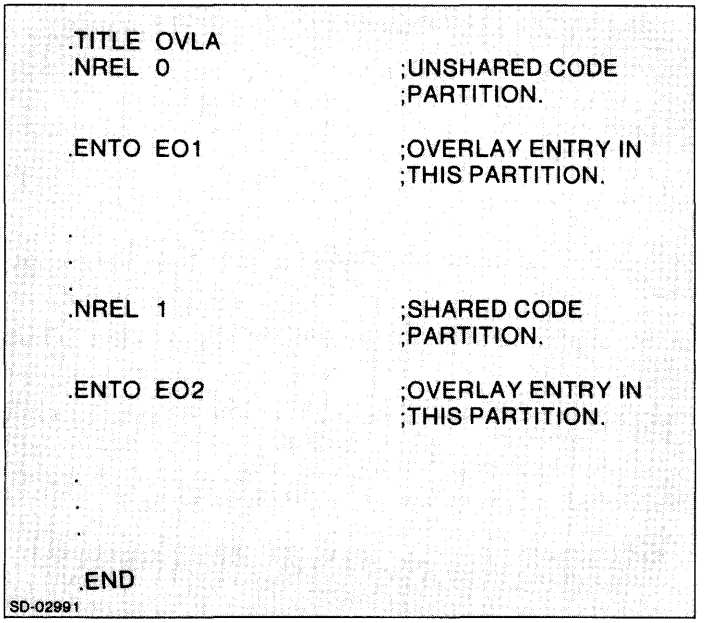


Figure 3-10. Splitting Overlay Code into Separate Partitions

Consult the Link utility and Macroassembler reference manuals for more information concerning the use of Address Information Blocks and the .NREL/.ENTO pseudo-ops.

Shared Routines

Shared routines are dynamically relocatable modules found in a library of shared routines called a *shared library*. The Shared Library Builder utility creates this library. When a shared library is built, it is assigned a numeric identifier from 0 to 63 (library numbers 0 and 1 are reserved for system use). Shared routines, by definition, are always read into shared memory areas. Shared routines are also always built in multiples of 1024 words (2048 bytes).

Shared routines must be both re-entrant and position-independent. They must be re-entrant because they are shareable. They must be position-independent because they are loaded into any shared page. You define the shared routine area when you bind the program. Link function switch /M specifies the number of 2K-byte pages which will be reserved as a single area within the shared area for use by all shared routines.

The system maintains 2K-byte shared routines on a least-recently-used basis, while it maintains larger shared routines on a first-available basis. This means that if the system needs a single 2K-byte page to load a 2K-byte shared routine and all shared routine pages are filled, it will overwrite that free page which was least-recently accessed. However, faced with a similar need for a larger area, it will overwrite as many free pages as necessary. Thus if you write shared routines that are no larger than 2K bytes each, the efficiency of programs using these routines may be improved.

Using Resource Calls to Manage Procedures

The system provides access to overlays and shared routines with a generalized procedure-call mechanism. This mechanism is implemented using three system calls: ?RCALL, ?KCALL, and ?RCHAIN. Using these calls, you may defer until Link time the decision where to place procedures: into an overlay, into a shared routine, or into the permanent resident context (root). Each procedure entry is defined by a .PENT pseudo-op; .ENTO is not used with the resource calls. Also, each procedure must be position-independent. That is, all internal procedure references must be relative to some point in the procedure.

Using the procedure-call mechanism, any procedure can call any other procedure (including itself). The calling procedure must begin with a ?RSAVE macro instruction, and each of the called procedures must end with an RTN instruction. Link resolves calls to each procedure appropriately, and a system resource manager (bound into the root) performs the loading of overlays or shared routines as required at execution time. The operation of the resource manager is invisible to the execution of the procedures themselves.

Three resource calls are defined:

?RCALL — Release the calling resource, and acquire the new resource before calling the desired procedure.

?KCALL — Keep the calling resource, and acquire the new resource before calling the desired procedure.

?RCHAIN — Release the current resource, and acquire the new resource before leaving the calling procedure. ?RCHAIN can be used only within a procedure which has been ?RCALLed or ?KCALLed. Upon exit from the called procedure, control returns to the originating procedure, not to the calling procedure.

?RCALL releases the calling resource before loading the new resource. Upon completion of the procedure in the new resource, the system reloads the calling resource, if necessary (see Figure 3-11). This will not entail an actual disk read if this task, or another task, has not used the resource area.

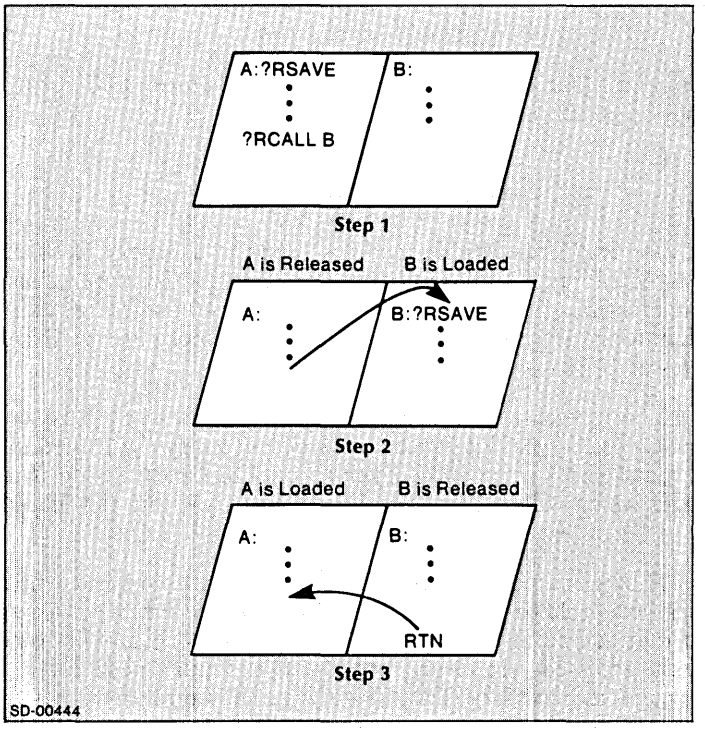


Figure 3-11. ?RCALL Illustration

After A is released (Step 2), a new overlay or shared routine could be asynchronously loaded into the old resource's area by another task. Control cannot return to the caller (A, Step 1) until A is reloaded. Note that the ?RCALL requires the system to preserve the state of A and restore it when A is reloaded.

?KCALL loads a new resource and transfers control to its entry point. In contrast to ?RCALL, ?KCALL does not release the calling resource. Figure 3-12 shows the operation of ?KCALL.

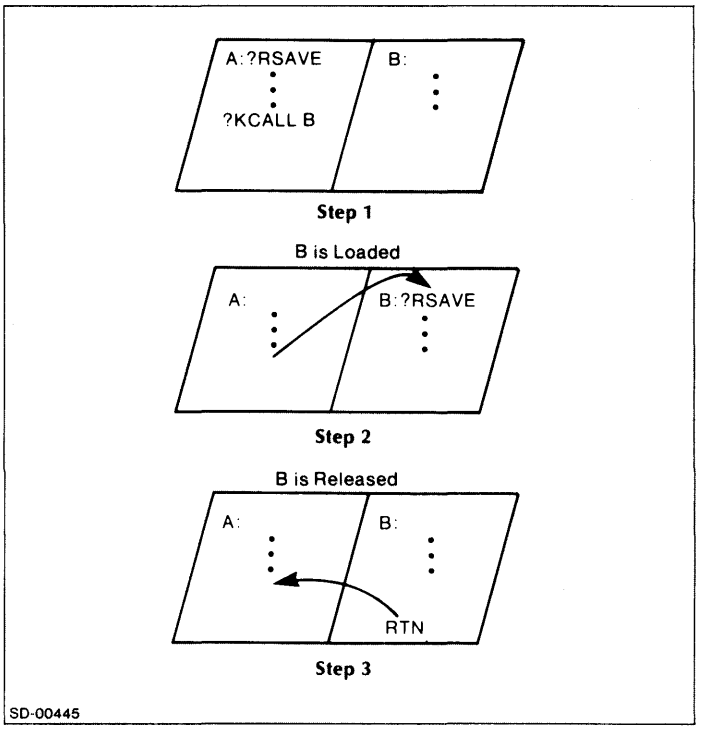


Figure 3-12. ?KCALL Illustration

Indiscriminate use of ?KCALL can lead to a *resource deadlock*. A resource deadlock occurs when all tasks requiring shared routines or overlays are waiting for shared routines or overlays to be released. That is, the system may need to release a given resource before it can load a new one in its place, yet be unable to release that resource. This will happen if one overlay issued a ?KCALL for an overlay procedure that would be loaded into the same overlay area that the caller occupies. For this reason, we recommend that you use ?RCALL.

?RCHAIN is used to split resources which are too large into smaller sequential pieces. It can be used only by procedures which reside in resources that have been ?RCALLed or ?KCALLed. Figure 3-13 illustrates ?RCHAIN.

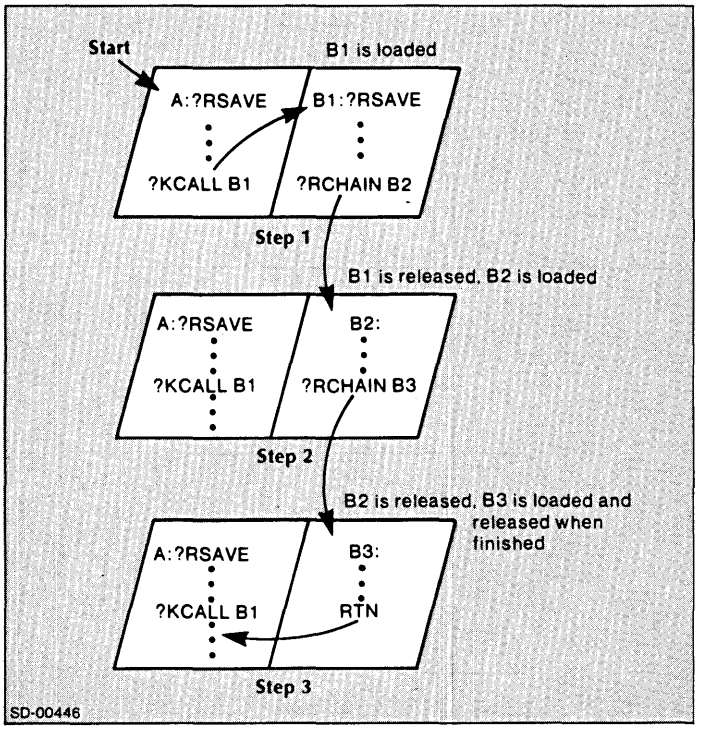


Figure 3-13. ?RCHAIN Illustration

As Figure 3-13 shows, procedure A ?KCALLs B1; B1 is loaded. At the end, B1 calls B2 with an ?RCHAIN. This releases B1 and loads B2. Likewise, B2 ?RCHAINs to B3, releasing B2 and loading B3. After B3's RTN, control returns to procedure A, the original ?KCALL's normal return. Note that B1 and B2 are not terminated by RTN, since there is no return of control back to the module issuing an ?RCHAIN.

?RCALL and ?KCALL behave as JSR instructions and generate a return address that can be saved by an ?RSAVE call at the beginning of the called routine. ?RCHAIN behaves as a JMP instruction and does not generate a return address. Thus, routines called by ?RCHAIN should not do an ?RSAVE.

Resource Call Formats

The preceding illustrations consider the use of resource calls in overlays and in shared routines, but omit any reference to procedures which are bound into the root. When a procedure which is bound into the root calls another procedure, releasing has no effect; likewise, no resource reload is made when return from the called procedure occurs. The system takes advantage of this by transforming, at Link time, certain uses of resource calls into simple EJSR instructions; this optimizes their execution. Table 3-1 shows what happens when ?KCALL and ?RCALL are changed into EJSR instructions; ?RCHAIN is never changed to EJSR, no matter what medium the procedure is called from.

When a resource in the root calls another resource in the root, Link changes the call to an EJSR instruction. Likewise, Link changes to EJSR instruction calls from one overlay to another within the same overlay area.

Table 3-1. Resource Calls Changed to EJSR Instructions

From	To	Root or Bound Shared Routine	Overlay	Shared Library Routine
Root	?KCALL	EJSR ,0	JSR @ 13	JSR @ 13
	?RCALL	EJSR ,0	JSR @ 14	JSR @ 14
	?RCHAIN	JSR @ 15	JSR @ 15	JSR @ 15
Overlay	?KCALL	JSR @ 13	JSR @ 13	JSR @ 13
	?RCALL	JSR @ 14	JSR @ 14	JSR @ 14
	?RCHAIN	JSR @ 15	JSR @ 15	JSR @ 15
Shared Library Routine*	?KCALL		JSR @ 13	
	?RCALL		JSR @ 14	
	?RCHAIN		JSR @ 15	

* Bound into root or not

Those resource calls which are not changed to EJSRs are resolved as calls to one of three resource manager routines, taken from URT.LB and bound into the root:

?KCALL becomes JSR @13;
?RCALL becomes JSR @14;
?RCHAIN becomes JSR @15.

Following each of these JSRs to a resource manager is a one-word procedure entry descriptor (these descriptors are shown in Figure 3-13). Since ?RCALL and ?KCALL may be transformed into EJSR instructions, the general format for these calls and for ?RCHAIN omits an exception return. In fact, since ?RCHAIN never regains control, it also lacks a normal return:

?KCALL *[argument]*
normal return
alternate return

?RCALL *[argument]*
normal return
alternate return

?RCHAIN *[argument]*
(no return)

Each argument to a resource call must have been defined in a .PENT statement. Note that you do not use the .EXTN pseudo-op to reference the argument when you pass an argument in-line. Each system call macro generates the appropriate external reference pseudo-op. If you do not pass an argument to the resource call, then it must be passed on the top of the stack when the call is made (see "Passing Procedure Entry Arguments to Resource Calls").

If the system detects an error condition during its attempt to execute a resource call, or if a resource deadlock occurs, the system transfers control to an error-processing module with entry ?BOMB. If you supplied no routine with this entry, the system supplies a default routine with this entry. The default ?BOMB routine simply terminates the calling process and passes the appropriate error code to the father process.

If you write your own ?BOMB and an error condition arises, the system transfers control to this routine and supplies information to the routine describing the condition: an error code in AC0, a procedure descriptor in AC1, and the fault address on the stack. The procedure descriptor describes the type of procedure causing the error, and the entry's address. Depending upon the medium containing the procedure, one of three types of descriptors will be provided; these are illustrated in Figure 3-14.

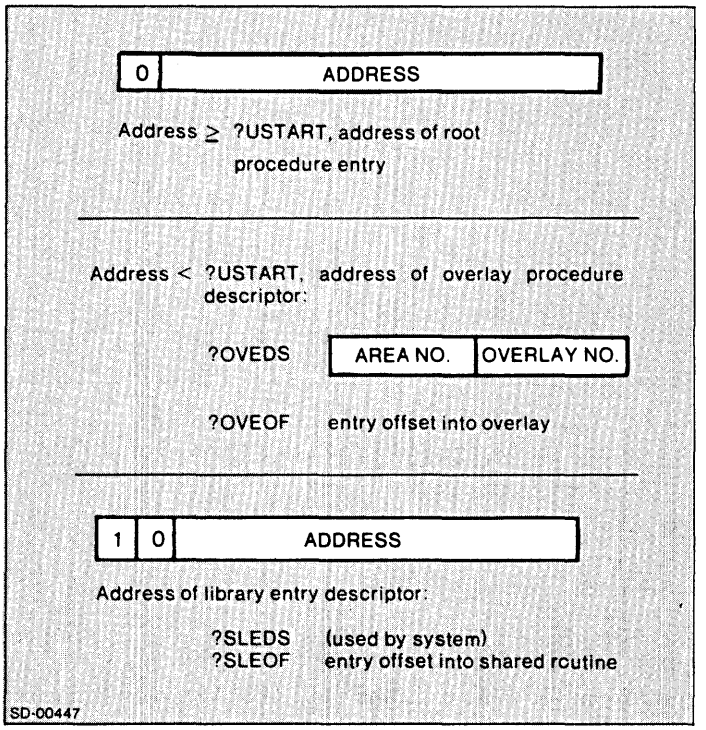


Figure 3-14. Procedure Entry Descriptors

?USTART, a symbol defined by Link, equals the starting address of the first word of user code (default unshared partition); the system places tables, needed to manage the user program, below ?USTART in the unshared partition. If the task has no outstanding resource, the descriptor is zero.

If a procedure call in a shared routine names a procedure entry that the system cannot find (perhaps because the tables below ?USTART have been destroyed), then the system will send to ?BOMB in AC1 the descriptor shown in Figure 3-15.

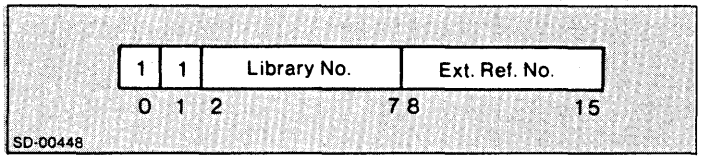


Figure 3-15. Library Descriptor upon Unknown Shared Routine

The External Reference field describes procedure entries called by routines within the library. There is one unique External Reference number for each different procedure entry that is called; these numbers range from 0 (the first procedure entry called in the lowest numbered shared routine) through *n* (the last unique procedure entry called in the highest shared routine issuing such a call).

Passing Procedure Entry Arguments to Resource Calls

Procedure entries are generally passed in-line as arguments to resource calls (e.g., ?RCALL *procedure-name).* Alternatively, procedure entry descriptors (shown previously in Figure 3-14) can be passed on the top of the stack. If so, the system POPs this descriptor off the stack before entering the called procedure. The .PTARG pseudo-op takes a procedure entry name and transforms it to the appropriate descriptor. Thus a calling sequence passing a procedure entry name on the stack to an ?RCALL looks as follows:

```
.NAME: .PTARG NAME
       .
       .
       .
       LDA 0,.NAME      ;GET PROCEDURE
                        ;ENTRY
       PSH 0,0          ;DESCRIPTOR AND
                        ;PUSH IT
                        ;ONTO STACK.

       LDA 0, ARG1      ;PASS ARG1,

       LDA 1, ARG2      ;ARG2, AND

       LDA 2, ARG3      ;ARG 3 TO
                        ;PROCEDURE
                        ;NAME.

       ?RCALL           ;IMPLICIT POP
                        ;BY THE SYSTEM
```

SD-02992

Placing the procedure entry descriptor on the stack allows one resource call in a single procedure to call any procedure, passing specific arguments in the accumulators to each procedure.

Alternate Return from Resources

Ordinarily, when the execution of a resource has completed, control returns to the first word following the resource call. However, you can have the resource return control to the second word following the resource call, by incrementing the caller's return address with the instruction:

```
ISZ ?ORTN,3
RTN
```

(Remember that AC3 contains the frame pointer.) This provides you with an alternate return address for use in whatever way you desire. The sequence works like this: after you've incremented the return address (ISZ ?ORTN,3), the RTN instruction passes control to an alternate entry point in the call processor. The call processor then passes control back to the correct point in the calling program.

System Management of Resource Calls

Any task's current resource (i.e., root, overlay, or shared routine) is always described in that task's TCB, offset ?TCUD. This description is identical to the descriptors shown back in Figure 3-14. The ?RCALL, ?KCALL, and ?RCHAIN procedure calls require two extra words on the user stack, located between the caller's return block and the callee's temporaries. AOS uses these two words to store information from the called procedure. Therefore, all ?RCALLing and ?KCALLing procedures must begin with the ?RSAVE macro instruction, which saves two additional words on the stack and end with the RTN instruction. All procedures that were acquired with the ?RCHAIN call must begin with the ?RSAVE macro instruction; but only the last procedure in any ?RCHAIN sequence can end with RTN, since RTN terminates the chain. Figure 3-16 illustrates the stack after an ?RSAVE instruction has executed, and gives the parametric names defined for stack locations.

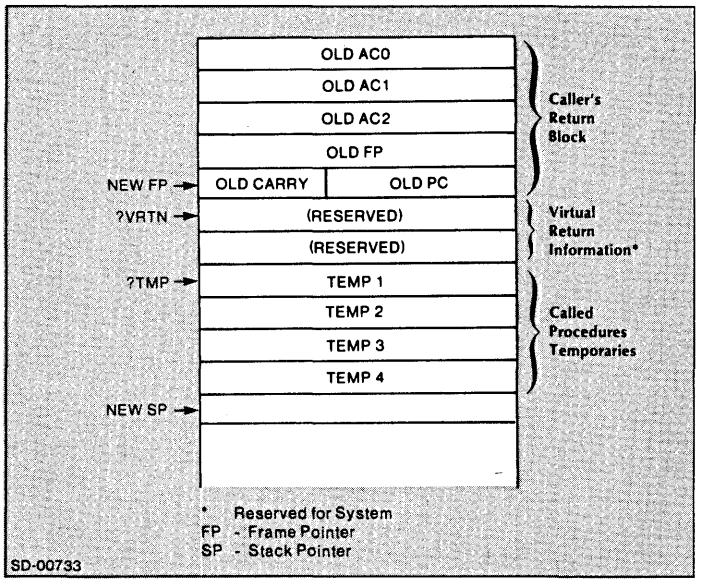


Figure 3-16. Resource Call Stack after ?RSAVE

Each word in the return block is defined parametrically in PARU.SR, and the macroassembler will recognize them. PARU.SR defines the following mnemonics for return block offsets:

Mnemonic	Meaning
?OAC0	Caller's AC0.
?OAC1	Caller's AC1.
?OAC2	Caller's AC2.
?OFP	Caller's frame pointer.
?ORTN	Caller's carry and return address.
?VRTN	Points to the first word of the two system-reserved words containing virtual return information.
?TMP	Points to the first word of the callee's temporary area.

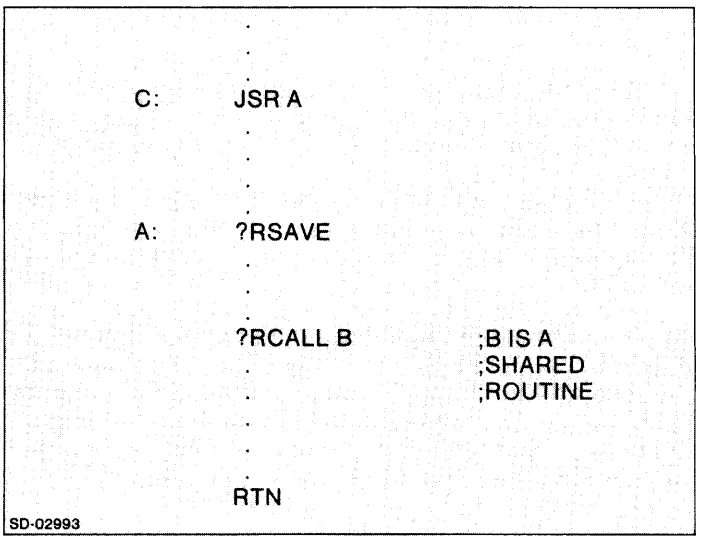
Consult the *Programmer's Reference Manual, ECLIPSE-line Computers,* "Stack Manipulation," for a description of the user stack.

Requirements of Runtime Relocatability

As stated earlier, you must write shared procedures relocatably, since the system may load shared routines into any available shared routine slot, and may load overlays into any basic area of a multiple overlay area.

In addition, since ?RCALL releases and reloads procedures, these procedures must be written with *runtime relocatability*, since a procedure may be released and reloaded into different areas of memory during the execution of a program. In order to ensure runtime relocatability, be certain that all location references are made as offsets relative to some point in the procedure.

However, a more subtle consequence follows from the requirements of runtime relocatability: *no* assembly language instruction, such as JSR, using a fixed address, can be issued before an ?RCALL in a procedure. That is, no such instruction can be issued if the resource will be released and reloaded, possibly invalidating the fixed address reference. The following extract of procedure code shows a JSR whose return value will be wrong if the procedure is reloaded into a different area of memory:



The address C + 1, to which RTN returns control, will not be the proper return address if the procedure has been relocated upon return from procedure B. Alternatively, if the procedure had issued ?KCALL B within subroutine A, the return would be executed without flaw since the procedure could not have been relocated before control returned from subroutine A.

The complete set of assembly language instructions subject to this runtime relocation restriction is: JSR, EJSR, LEF, ELEF, PSHJ, POPJ, XOP, and PSHR. If you avoid releasing a procedure containing one of these instructions, or if you use such a procedure in a single basic area, then you can use these instructions at will.

Primitive Overlay Management

We strongly recommend that you avoid using primitive overlay calls in your applications. The generalized stack-based mechanism ?RCALL provides you with a much better environment since overlay management is automatic. Except in very rare circumstances, use the resource call just described.

Two actions are required to access an overlay using primitive overlay calls: load the overlay, and release it when it is no longer needed. Table 3-2 lists the overlay calls.

Table 3-2. Primitive Overlay Management Calls

Step	Call
1. Load an overlay.	?OVLOD
2. Release the overlay.	?OVREL, ?OVEX, or ?OVKIL

As part of its resource management activities, the system maintains an overlay use count (OUC) of the number of tasks using an overlay which is currently resident. OUC is decremented each time an overlay is released. When OUC goes to zero, some other overlay may then be loaded into this overlay area.

As long as any task is using an overlay and has not released it, then no other overlay can be loaded into the basic overlay area it occupies. Of course, if the total overlay area consists of several basic areas, another task may use any free basic area in the total area. This is true even if some higher priority task issues an overlay load request.

?OVLOD loads an overlay; it can send control either to the beginning or to an offset within the overlay. Loading is either conditional or unconditional. The latter loads the requested overlay when the overlay area is released (regardless of whether or not the overlay's image currently resides in the overlay area). Under conditional loading, the system first checks to see whether or not the overlay currently resides in the overlay area. If it does, then the overlay's OUC is incremented and no I/O is performed. If it does not, then the overlay is loaded into the area and its OUC is set to one.

You must explicitly release each overlay loaded by ?OVLOD.

?OVREL decrements OUC and releases the area if OUC equals zero. This call must not be issued from within the overlay that is to be released. ?OVEX decrements OUC and releases the area if OUC equals zero; additionally, this call directs control to a specified address. ?OVEX is particularly useful if the caller is returning from a subroutine within an overlay. Finally, ?OVKIL decrements OUC, releases the area if OUC equals zero and kills the calling task.

Abnormal program termination results when ?OVKIL kills the last task in the active TCB queue.

System Call Summary

The following list summarizes the calls used for memory management:

?FLUSH	Write the contents of a shared page to disk.
?GCRB	Get base of current resource.
?GSHPT	List the current shared partition size.
?KCALL	Make a general procedure call, and do not release the calling resource.
?MEM	List NMAX and unshared page parameters.
?MEMI	Change the number of unshared pages.
?OVEX	Exit an overlay and return to a specified location.
?OVKIL	Exit an overlay and kill the calling task.
?OVLOD	Perform a primitive overlay load, with optional transfer of control to the overlay.
?OVREL	Release an overlay and return to the caller.
?RCALL	Make a general procedure call, and release the calling resource.
?RCHAIN	Release the current resource, acquire a new resource; the chained procedure will return control to the originating procedure.
?RPAGE	Release a shared page.
?SCLOSE	Close a file which has been ?SOPENed for shared page access.
?SOPEN	Open a file for shared page access.
?SPAGE	Perform a shared page read.
?SSHPT	Establish a new shared partition.
?UNWIND	Unwind the stack and restore the previous environment.
?WALKBACK	Return information on previous frame in the stack.

?FLUSH

Write a shared page to disk.

?FLUSH
exception return
normal return

Input/Output

Input:

AC0 any address in the page to be flushed (the system derives the logical page number by stripping off bits 6-15).

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

ERNSA The address is not in the shared area.

Description

This system call writes the contents of a shared page out to disk. It does not, however, release the shared page. Control will not take the normal return until the flush is completed. If the page to be flushed is empty (perhaps already released), then control takes the normal return and the call is effectively nonoperational.

?GCRB

Get the base of the current resource.

?GCRB
normal return

Input/Output

Input:

none.

Output:

AC0 contains the base of the current resource.

AC1 unchanged.

AC2 unchanged.

Description

?GCRB returns a label indicating the base of the current resource. You can use this information, in conjunction with resource calls, to determine the current base-relative offset of a moveable resource. (A *moveable resource* is a resource which may move to various locations during execution.) To determine the absolute address of an offset, add the value of the relative offset to the value of the resource base.

NOTE: There is no exception return to this call. If the system encounters an invalid resource base, it transfers control to an error-detection routine, ?BOMB.

?GSHPT

List the current shared partition size.

?GSHPT
exception return
normal return

Input/Output

Input:

none.

Output:

AC0 number of the starting shared page.

AC1 number of pages in the shared area.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This system call returns the size of the shared area in the context of the process issuing this call. The logical page number of the first shared page, from 1 through 31 (page zero can never be shared) is returned, as well as the number of pages in the area. If a starting page number of zero is returned, this indicates that there is no shared page; likewise, zero is returned as the number of pages in the shared area.

The starting shared page number is also maintained in USTST, and the number of pages in the shared area is maintained in USTSZ. Both these locations are in the User Status Table (described at length in Appendix C).

?KCALL

Keep the calling resource and acquire a new one.

?KCALL *[procedure entry]*
normal return
alternate return

Input/Output

Input:

Input accumulators and carry are passed to the procedure.

Output:

Unchanged from input unless modified by the procedure (i.e., ?OAC0 through ?OAC2 have been changed by the called procedure).

Exceptional Condition Codes in AC0 Presented to ?BOMB

FILE SYSTEM codes

MEMORY codes

CHANNEL-RELATED codes (due to internally allocated channels)

Description

This call keeps the calling procedure (which may reside in the root, in an overlay, or in a shared routine), loads a new resource, and transfers control to the requested procedure entry. If the target procedure entry is not passed in-line, then it must be passed on the top of the stack (the system will then POP the word from the stack).

Each called procedure must be defined by .PENT, and must end with a RTN instruction. The calling procedure must begin with a ?RSAVE macro instruction. Control will take the normal return upon executing RTN, unless the called procedure issues ISZ ?ORTN,3 in the body of the procedure. (Remember that AC3 contains the frame pointer.) In this case, control goes to the alternate return.

If the task issuing ?KCALL is killed, then the system releases the current resource. The resource issuing the ?KCALL, however, remains unreleased. Thus you should not kill a task until all its outstanding ?KCALLs have gone to completion.

?KCALL requires greater amounts of memory than ?RCALL since it does not release the calling resource, and it may lead to resource deadlocks. For this reason, we recommend that you use ?RCALL instead of ?KCALL. Consult "Using Resource Calls to Manage Procedures," presented earlier in this chapter.

?MEM

List NMAX and unshared page parameters.

?MEM
exception return
normal return

Input/Output

Input:

none.

Output:

AC0	maximum number of unshared pages available for use in a ?MEMI call.
AC1	number of unshared pages currently in use.
AC2	highest currently-used address in unshared space. This value is always page-adjusted, i.e., the least significant ten bits are always set to ones.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This system call indicates the maximum number of unshared pages which are available, the number of pages which are currently in use for unshared purposes, and the highest currently-used unshared address in a context. The value returned will always be page-adjusted, i.e., the right ten bits will be set to ones. Thus if a context has twenty shared pages and seven unshared pages, system call ?MEM would indicate that only five more pages are available.

The number of unshared pages is also maintained by the system in USTBL of the User Status Table.

?MEMI

Change the number of unshared pages.

?MEMI
exception return
normal return

Input/Output

Input:

AC0	number of pages to be allocated or two's complement of number to be deallocated.

Output:

AC0	destroyed.
AC1	new highest address.

Exceptional Condition Codes in AC0

ERMEM	Insufficient amount of memory available. Possibly an attempt has been made to exceed the amount of memory allocated to the process when it was created.
ERMRL	Memory release error.
ERNSW	Swap file space exhausted.

Description

This system call increases or decreases the number of unshared pages in a context. Pass a positive or two's complement negative value to this system call indicating the number of pages by which the context is to be increased or decreased. On return from the call, the new highest unshared address is returned, and this value is also updated in USTBL of the User Status Table.

Exercise care to ensure that pages are not allocated which are not truly required by the process. Allocating unneeded pages will increase system overhead if the process is swapped to disk. Also take care not to increase a resident process's memory requirements beyond the bounds of physical memory. This could cause a system deadlock.

?OVEX

Release an overlay area and return.

?OVEX
exception return

Input/Output

Input:

AC0	bit 0: ignored bits 1-6: area number bits 7-15: overlay number	.ENTO argument
AC2	return address upon successful execution of this call.	

Exceptional Condition Codes in AC0

EROVN Invalid overlay number; this overlay is not in the overlay area.

FILE SYSTEM codes

Description

This command exits an overlay loaded by a previous call to ?OVLOD. This command decrements OUC for the overlay and releases the area if OUC goes to zero. Additionally, control returns to an address specified by the caller. Typically, this might be the return address of the caller if returning from a subroutine within an overlay.

?OVKIL

Exit an overlay and kill the task.

?OVKIL
exception return

Input/Output

Input:

AC0	bit 0 ignored	ENTO argument
	bits 1-6: area number	
	bits 7-15: overlay number	

Output:

None.

Exceptional Condition Codes in AC0

EROVN Invalid overlay number; this overlay is not in the overlay area.

FILE SYSTEM codes

Description

This command kills the calling task and decrements the OUC, releasing the overlay loaded by a previous call to ?OVLOD if OUC goes to zero. This command may be issued from within the overlay which is to be released.

If ?OVKIL kills the last active task, the system will generate the error message "Last task was killed," and will abort the program.

?OVLOD

Load and go to an overlay.

?OVLOD
exception return
normal return

Input/Output

Input:

AC0 bit 0: set to one, unconditional load; set to zero, conditional load.
bits 1-6: area number
bits 7-15: overlay number
} .ENTO argument

AC1 bit 0: set to one if control goes to an offset, reset to zero if control goes to an address.
bits 1-15: address or offset to which control shall go. (If bits 0-15 contain -1, control returns to the ?OVLOD normal return.)

AC2 passed to overlay (unless AC1 contains -1).

Output:

AC0 unchanged.

AC1 base of the overlay being used.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERADR Offset specified in AC1 exceeds the overlay size.

EROVN Value specified in AC0 cannot be found in the overlay structure.

FILE SYSTEM codes

Description

This call loads an overlay and optionally transfers control to a point within the overlay. Unconditional loading will load the overlay whether or not it is currently resident in the given overlay area; conditional loading will load the overlay only if it is not already resident.

If AC1 contains -1, control goes to the ?OVLOD normal return after the overlay has been loaded. Otherwise, two actions are possible. If AC1 bit 0 is set to one, then bits 1-15 are treated as an offset into the overlay where control will go; otherwise bits 1-15 are treated as an absolute address receiving control in the overlay. If the overlay area is some multiple of the basic area size, then bits 1-15 must be an offset since it cannot be known beforehand which slot in the total overlay area will receive the overlay.

The overlay base is always passed in AC1 to the overlay when it receives control. Output values shown above are received only if AC1 contained -1 upon input, or if the loaded overlay returned control by means of RTN. Otherwise, output values at the ?OVLOD normal return will be those values input to ?OVEX.

NOTE: If the loaded overlay received control directly from ?OVLOD, and this overlay issues ?OVKIL, the ?OVLOD normal return will never be taken since the task is then killed.

?OVREL

Release an overlay area.

?OVREL
exception return
normal return

Input/Output

Input:

AC0	bit 0: ignored	.ENTO argument
	bits 1-6: area number.	
	bits 7-15: overlay number	

Output:

AC0	unchanged.

Exceptional Condition Codes in AC0

EROVN	Invalid overlay number. This overlay is not in the overlay area, or no such area exists.

Description

After ?OVLOD loads an overlay, this call decrements the OUC and releases the area if OUC goes to zero. This command must not be issued from the overlay which is to be released (since the normal return would be in the overlay that is being released).

?RCALL

Release one resource and acquire a new one.

?RCALL *[procedure entry]*
normal return
alternate return

Input/Output

Input:

Input accumulators and carry are passed to the procedure.

Output:

Unchanged from input unless modified by the called procedure (i.e., ?OAC0 through ?OAC2 have been changed by the called procedure).

Exceptional Condition Codes in AC0 Presented to ?BOMB

FILE SYSTEM codes

MEMORY codes

CHANNEL-RELATED codes (due to internally allocated channels)

Description

This call releases the calling procedure (which may reside in the root, in an overlay, or in a shared routine), loads a new procedure and transfers control to the requested procedure entry. If the target procedure entry is not passed in-line, then it must be passed on the top of the stack. (The system will then POP the word from the stack.)

Each called procedure must be defined by .PENT, and must end with a RTN instruction. The calling procedure must begin with a ?RSAVE macro instruction. Control will take the normal return upon executing RTN, unless the called procedure issues ISZ ?ORTN,3 in the body of the procedure. (Remember that AC3 contains the frame pointer.) In this case, control goes to the alternate return.

Since ?RCALL always releases the calling resource, if it (and ?RCHAIN, optionally) is used exclusively, no resource deadlock can occur.

?RCHAIN

Chain to a new procedure.

?RCHAIN *[procedure entry]*

Input/Output

Input:

Input accumulators and carry are passed to the procedure. (Output to caller's caller same as input to ?RCHAIN, unless the called procedure modifies ?OAC0, ?OAC1, or ?OAC2.)

Exceptional Condition Codes in AC0 Presented to ?BOMB

FILE SYSTEM codes

MEMORY codes

CHANNEL-RELATED codes (due to internally allocated channels)

Description

Like ?RCALL, this call releases the calling procedure (which may reside in the root, in an overlay, or in a shared routine), loads a new procedure and transfers control to the requested procedure entry. If the target procedure entry is not passed in-line, then it must be passed on the top of the stack (the system will then POP the word from the stack).

Each called procedure must be defined by .PENT and bracketed by ?RSAVE/RTN.

Unlike ?RCALL, ?RCHAIN has no normal return. When the system executes RTN in the called procedure, control goes to the procedure which issued the initial ?RCHAIN *not* to the ?RCHAIN caller.

Figure 3-17 clarifies the operation of ?RCHAIN by showing the stack before and after this call.

?RCHAIN (continued)

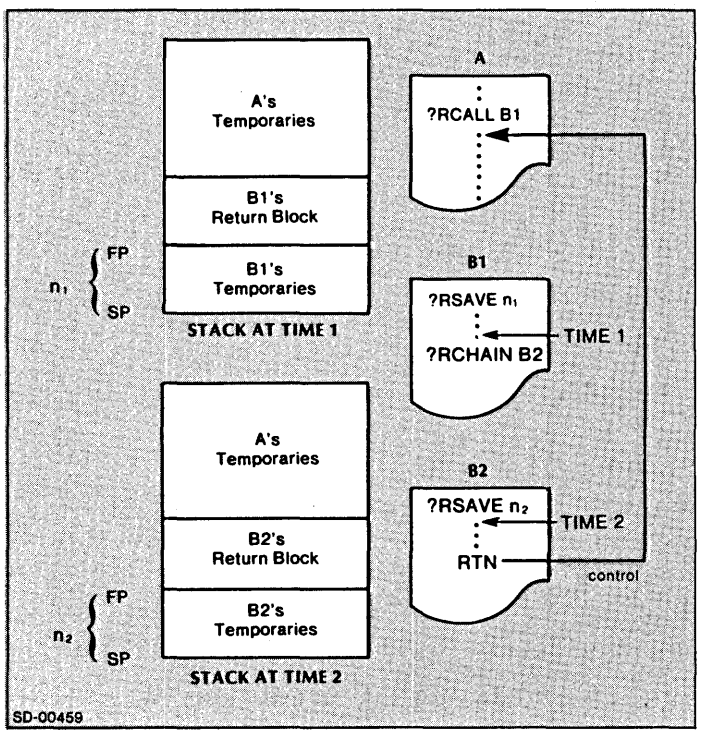


Figure 3-17. Stack Before and After ?RCHAIN

?RPAGE

Release a shared page.

?RPAGE
exception return
normal return

Input/Output

Input:

AC0 bit zero: 1=flush page to disk.

bits 1-15: any address in the page to be released (the system derives the logical page number by stripping off bits 6-15).

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

ERNSA The page is not within the shared area.

Description

This system call releases a page into which a shared file has been read by a previous call to ?SPAGE. Issuing this call will decrement the file (or portion of the file's) use count and the page containing the shared information will be placed in the LRU chain, if the use count is now zero.

If the page targeted by AC0 is empty (perhaps it was released earlier), then control simply takes the normal return.

If you set bit zero of AC0 to one, the ?RPAGE call will flush the specified page to disk in addition to releasing it; it will return after the I/O is complete. Thus an ?RPAGE with AC0's bit zero set to one is equivalent to ?FLUSH followed immediately by a normal ?RPAGE.

NOTE: Issuing this call does not close the file; the file should be closed by all users who ?SOPENed it before it can be opened for other purposes.

?SCLOSE

Close a file opened for shared page access.

?SCLOSE
exception return
normal return

Input/Output

Input:

AC1 bit zero: 1=release any shared pages.

bits 1-15: channel number.

Output:

AC1 unchanged.

Exceptional Condition Codes in AC0

CHANNEL-RELATED codes

Description

This call breaks the association between a channel and a file that was established by a previous call to ?SOPEN. Files opened on channels by calls to ?OPEN or ?GOPEN must be closed by calls to ?CLOSE and ?GCLOSE respectively. We describe ?OPEN and ?CLOSE in Chapter 6; ?GOPEN and ?GCLOSE are described in Chapter 13.

By setting bit 0 of AC1 to 1, you can cause the automatic release of any pages in the file that have not previously been released. If you don't set this bit, and if any of the file's pages are still part of your shared partition, ?SCLOSE will return an error when you try to close the file.

Files must be closed after use to ensure the updating of file status information and to free channels for other use. All files are automatically closed when the process which opened them terminates or CHAINs.

?SOPEN

Open a file for shared page access.

?SOPEN
exception return
normal return

Input/Output

Input:

AC0 byte pointer to filename.

AC1 channel number (-1, system selects a channel).

AC2 if 0, perform a read-only shared page open; else perform a read/write shared page open.

Output:

AC0 unchanged.

AC1 channel number used to open the file.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERILO Attempt to ?SOPEN a file whose file element is not a multiple of four.

FILE SYSTEM codes

Description

Before a file can be read by a call to ?SPAGE, that file must be opened for shared page access by a call to ?SOPEN. Successful shared opening of a file requires that the file's element size be a multiple of four. If the input contents of AC2 are zero, then regardless of the caller's access rights to the file, read-access only is permitted. Conversely, if the input contents of AC2 are nonzero, then read/write access is allowed if the caller has these access rights. If read/write access is obtained, the system assumes that the file has been modified, and will flush the file to disk when it is closed. System call ?SCLOSE is used to close a file which was opened previously by ?SOPEN.

?SPAGE

Perform a shared page read.

?SPAGE *[packet address]*
exception return
normal return

Input/Output

Input:

AC1 channel number.

AC2 base address (?PSTI) of parameter packet.

Output:

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

MEMORY codes

CHANNEL-RELATED codes

Description

The parameter packet passed to system call ?SPAGE is identical to that passed to the physical record I/O calls, ?RDB/?WRB (see Chapter 13). This packet, shown in Figure 3-18, is a table with the following structure.

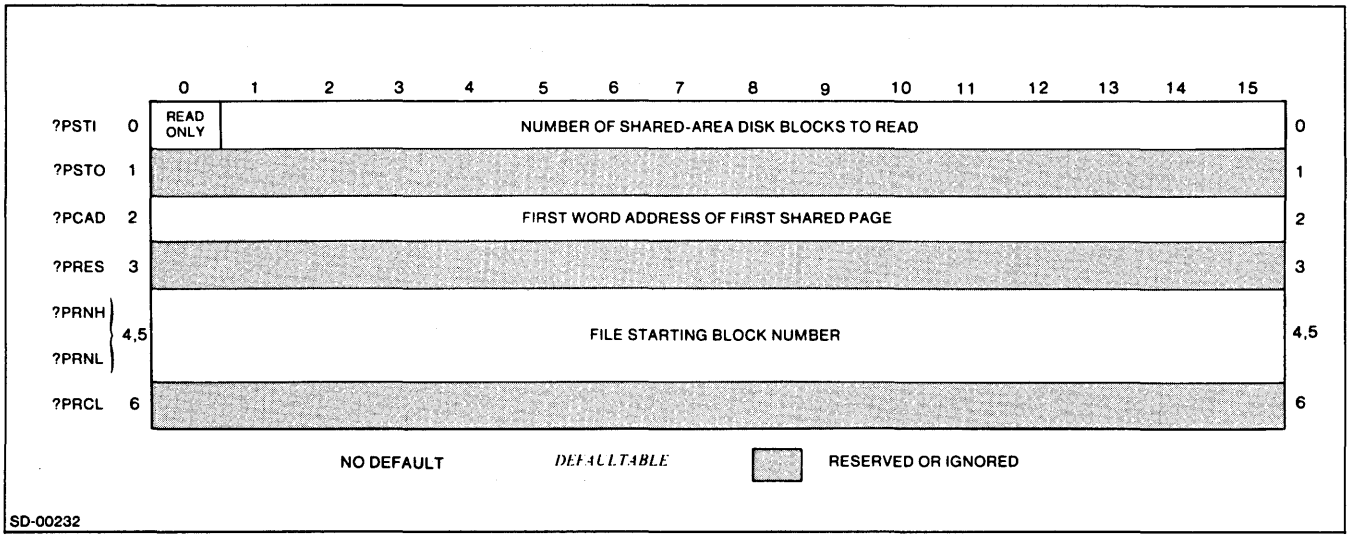


Figure 3-18. ?SPAGE Parameter Packet

Offset	Contents
?PSTI	Number of disk blocks to be read into the shared area (must be a multiple of 4). If you set bit zero of ?PSTI to one, you will get only read access to shared pages, even if the ?SOPEN on the file specified read/write access. Consequently, the system will not have to flush the page(s) to disk when they are ultimately released.
?PSTO	Reserved for system use.
?PCAD	Starting address of the first word in the first shared page to receive file data; start of page boundary.
?PRES	Reserved for system use.
?PRNH	High order portion of starting file block number.
?PRNL	Low order portion of starting file block number (must be a multiple of 4).
?PRCL	Reserved for system use.

This system call reads a multiple of four contiguous disk blocks into one or more shared pages in a context. Before this call can be issued, the shared open call, ?SOPEN, must have been issued to open the file.

Issuing the ?SPAGE call will read the desired portion of the file into a shared page(s) if the page(s) does not already exist in main memory; if it does reside in main memory, then the page(s) will be mapped into the caller's shared partition. Additionally, the use count of the desired file will be incremented, and if a previous ?SPAGE call has been issued to read pages into the named shared partition pages, they will be released.

?SSHPT

Establish a new shared partition.

?SSHPT
exception return
normal return

Input/Output

Input:

AC0 starting logical page number (1-31).

AC1 number of pages in the new shared area.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

ERSHP Pages specify an unshared area, or are illogical.

ERMEM Insufficient amount of memory available.

Description

This call establishes a new shared partition. That is, it permits pages to be allocated or deallocated so that a differently sized shared area exists within the current context. If the new shared partition will either allocate pages presently allocated for unshared use or release shareable pages which are currently being shared, then the exception return is taken (ERSHP). Unless one of these conditions exists, this call can be used to release a shared area in the current context.

?UNWIND

Unwind the stack, restore the previous environment.

?UNWIND
normal return

Input/Output

Input:

none.

Output:

ACO	contains either the base of the restored resource or O.
AC1	contains the logical address (within the previous resource) which *would have* assumed control, given a normal return.
AC2	unchanged.
AC3	contains a pointer to the frame (level) which was restored.

Description

This system call unwinds the stack and returns it to the original environment. Program counter control passes to the instruction following the ?UNWIND. ?UNWIND also loads the accumulators with information about the resource in the previous environment. In general, you use ?UNWIND in conjunction with resource calls when you want to restore a previous resource but not necessarily return there. Code following ?UNWIND can transfer control to the normal return in the return block (AC1), to a fixed address relative to the address in the return block (AC1 + *n*), or to a fixed position within the restored resource (AC0+ *n*).

If you did not use a resource call to arrive at the current environment, the system returns a zero in AC0.

Notes

1. There is no exception return for ?UNWIND.
2. ?UNWIND affects the stack just as RTN does.
3. The ?UNWIND call itself must be either root resident or in a resource which will not be released by unwinding the stack.
4. The code generated by ?UNWIND may not be a single word. Keep this in mind before you issue a skip instruction prior to a ?UNWIND.

?WALKBACK

Return information on previous frames in the stack.

?WALKBACK
normal return

Input/Output

Input:

First ?WALKBACK Call:

AC2 contains 0.

Subsequent ?WALKBACK Calls:

AC1 unchanged from previous call.

AC2 unchanged from previous call.

AC3 contains current frame pointer (carry preserved).

Output:

AC0 if AC1 bit 0=0, AC0 contains the normal return for this environment.
if AC1 bit 0=1, AC0 contains the procedure entry descriptor for this environment.

AC1 bit 0=0, if you did not use a resource call to arrive at this environment.
bit 0=1, if you used a resource call to arrive at this environment.

AC2 contains the address of the frame for this environment level; otherwise, 0, indicating that there are no more frames left.

AC3 contains the current frame pointer (carry is preserved).

Description

This system call traces back through the stack to return information about the calling resource at a particular environment level. The first ?WALKBACK you issue starts with the current level. You can use this call repeatedly until you reach the desired level.

Set AC2 to 0 before your first use of ?WALKBACK. On subsequent calls, AC2 contains the address of the stack frame referenced by the last ?WALKBACK. Do not change the contents of AC1 or AC2 between calls, since ?WALKBACK uses this information to point to previous stack levels.

Examples

In this section we show one way to implement a certain function using resource calls. The function, PROJECTOR1, is a simple one: write two one-line messages to the console.

To invoke PROJECTOR1, type the command line

```
XEQ PROJECTOR1 console-name ↵
```

You need to supply the console name in the command line so that the program can open and write the messages to it. Figure 3-19 illustrates PROJECTOR1.

The first line of the program contains an ?RSAVE macro call, which performs stack-related initialization for the subsequent ?RCALLS. You must begin every program containing resource calls with the ?RSAVE call. ?GTMES, line 15, is a system call described in Chapter 9. It fetches the console name from the invoking command line. PROJECTOR1 then issues a resource call, ?KCALL, to load and execute resource PICT1. When PROJECTOR1 receives control again, it issues a 10-second delay (line 20). This gives you time to read the first text message and distinguish it from the next. (Delay is described in Chapter 9; it suspends the caller for a caller-specified number of seconds.) PROJECTOR1 issues a second ?RCALL, executing resource PICT2. Again, when PROJECTOR1 receives control it pauses before resuming operations. Finally, at GOODBY (line 25), PROJECTOR1 terminates itself. Now lets examine resources PICT1 and PICT2.

PICT1 and PICT2, shown in Figure 3-20, perform identical operations. These routines differ only in the specific text messages they output.

First, they open the console (lines 11-15). Then each writes its text message to the console (lines 16-17). You may ignore the concepts of opening and writing at this time; they are described and illustrated in Chapter 6. Upon completion of the ?WRITE, each issues RTN and returns control to the caller, PROJECTOR1.

```
!0002  PROJE
01
02        000040          SIZE=32.
03 00037'000040 STACK:  .BLK     SIZE     ;ALLOCATE THE STACK
04              ;GET MESSAGE PACKET
05 00077'000004 MSGPK:  .BLK ?GTLN
06        000077'       .LOC MSGPK+?GREQ
07 00077'000003         ?GARG
08        000100'       .LOC MSGPK+?GNUM
09 00100'000001         1
10        000101'       .LOC MSGPK+?GSW
11 00101'000000         0
12        000102'       .LOC MSGPK+?GRES ;BYTE PTR.TO ADRS.OF CON.NAME
13 00102'000052"        CON*2
14        000103'       .LOC MSGPK+?GTLN
15
16                      .END    PROJ

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS

 0001  PROJE    AOS ASSEMBLER REV 01.09          13:46:19 12/04/78
01
02              ;0001   PROJ1
                                .TITL   PROJECTOR1
04                              .EXTN   PICT1,PICT2,ERROR,DELAY
05                              .ENT    CON
06
07              ;THIS PROGRAM FLASHES TWO MESSAGES ONTO THE CONSOLE.
08              ;EACH IS OUTPUT BY A SEPARATE PROCEDURE, PICT1 AND PICT2.
09              ;TO RUN THE PROGRAM, TYPE "PROJECTOR1 'CONSOLE NAME'
10              ;<NEW LINE>".
11
12
13                              .NREL
14              PROJ:   ?RSAVE  0       ;INITIALIZE STACK FOR ?RCALLS
15                                      ;FOR ?DESC AND ?VRTN
16                                      ;FOR ?DESO AND ?VRTN
17                      ?GTMES  MSGPK   ;READ ORIGINAL COMMAND LINE
18 00006'002416         JMP     @.ERROR ;PROCESS THE ERROR
19                      ?RCALL  PICT1   ;CALL PICT1 TO DISPLAY 1ST MSG.
20                                      ;THEN RETURN AND
21 00011'064012         LEF     1,10.   ;GIVE READER 10 SECS. TO READ IT
22 00012'006411         JSR     @.DLA
23                      ?RCALL  PICT2   ;DO SAME FOR 2ND. MSG.
24 00015'064012         LEF     1,10.   ;10 MORE SECS. TO READ IT.
25 00016'006405         JSR     @.DLA
26 00017'152620 GOODBY: SUBZR   2,2     ;PREPARE FOR NORMAL RTN TO CLI
27                      ?RETURN
28 00022'002402         JMP     @.ERROR
29
30
31 00023'000000$.DLA:   DELAY
32
33 00024'000000$.ERROR: ERROR
34 00025'000012 CON:    .BLK    10.     ;AREA TO RECEIVE CONSOLE NAME
35
```

Figure 3-19. PROJECTOR1 Routine (continues)

```
  0003 PROJE

CON    000025' EN     1/05     1/34     2/13
DELAY  000004  XN     1/04     1/31
ERROR  000003  XN     1/04     1/33
GOODB  000017'        1/26
MSGPK  000077'        1/18     2/05     2/06     2/08     2/10     2/12     2/14
PICT1  000001  XN     1/04     1/20
PICT2  000002  XN     1/04     1/24
PROJ   000000'        1/14     2/16
SIZE   000040         2/02     2/03
STACK  000037'        2/03
.DLA   000023'        1/22     1/25     1/31
.ERRO  000024'        1/18     1/28     1/33
?GTME  004545  MC     1/17
?RCAL  006422  MC     1/19     1/23
?RETU  004600  MC     1/27
?RSAV  000124  MC     1/14
?XCAL  000001         1/18     1/20     1/24     1/28
```

Figure 3-19. PROJECTOR1 Routine (concluded)

```
 0001  PICT1     AOS ASSEMBLER REV 01.09          13:47:08 12/04/78
01                ;0001 PICT1 AOS ASSEMBLER
                                .TITL   PICT1
03                              .PENT   PICT1
04                              .EXTN   ERROR,CON
05
06                      ;THIS PROCEDURE OUTPUTS THE FIRST TEXT MESSAGE.
07
08                              .NREL
09
10 00000'163710         PICT1:  SAVE    0
11       000000
12
13 00002'024416                 LDA     1,.CON
14 00003'125120                 MOVZL   1,1    ;FORM A BYTEPOINTER
15 00004'044426                 STA     1,SCREEN+?IFNP ;STORE IT IN PKT.
16                              ?OPEN   SCREEN
17 00011'002442                 JMP     @.ERROR
18                              ?WRITE  SCREEN
19 00016'002435                 JMP     @.ERROR
20 00017'127710                 RTN
21 00020'000000$        .CON:   CON
22
```

Figure 3-20. PICT1 and PICT2 Routines (continues)

```
!0002  PICT1
01
02                          ;SCREEN I/O PACKET
03
04 00021'000014             SCREEN: .BLK ?IBLT
05        000021'                    .LOC SCREEN+?ICH
06 00021'000000                      0
07        000022'                    .LOC SCREEN+?ISTI
08 00022'000011                      ?RTDY+?OFOT                ;WRITE DYNAMIC RECORDS
09        000023'                    .LOC SCREEN+?ISTO
10 00023'000000                      0
11        000024'                    .LOC SCREEN+?IBAD
12 00024'000072"                     MESSAGE*2                  ;BYTE PTR. TO MESSAGE
13        000025'                    .LOC SCREEN+?IRES
14 00025'000000                      0
15        000026'                    .LOC SCREEN+?IRCL
16 00026'000040                      32.                        ;32 CHARS. LONG
17        000027'                    .LOC SCREEN+?IRLR
18 00027'000000                      0
19        000030'                    .LOC SCREEN+?IRNH
20 00030'000000                      0
21        000010                     .LOC ?ICH+?IRNL
22 00010 000000                      0
23        000032'                    .LOC SCREEN+?IFNP
24 00032'000000                      0
25        000033'                    .LOC SCREEN+?IMRS
26 00033'177777                      -1
27        000034'                    .LOC SCREEN+?IDEL
28 00034'177777                      -1
29        000035'                    .LOC SCREEN+?IBLT
30                          ;END OF PACKET
31
32 00035'052110             MESSAGE: .TXT   "THIS IS THE FIRST MESSAGE.<012>"
33        044523
34        020111
35        051440
36        052110
37        042440
38        043111
39        051123
40        052040
41        046505
42        051523
43        040507
44        042456
45        005000
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
```

Figure 3-20. PICT1 and PICT2 Routines (continued)

```
 0003  PICT1
01
02 00053'000000$        .ERROR: ERROR

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS
  0004 PICT1

CON    000002  XN     1/04     1/21
ERROR  000001  XN     1/04     3/02
MESSA  000035'         2/12     2/32
PICT1  000000' PN      1/03     1/10
SCREE  000021'         1/15     1/17     1/19     2/04     2/05     2/07     2/09
                       2/11     2/13     2/15     2/17     2/19     2/23     2/25
                       2/27     2/29
.CON   000020'         1/13     1/21
.ERRO  000053'         1/17     1/19     3/02
?OPEN  004315  MC      1/16
?WRIT  004436  MC      1/18
?XCAL  000001          1/17     1/19
```

```
 0001  PICT2     AOS ASSEMBLER REV 01.09          13:47:41 12/04/78
01                ;0002 PICT2
                                  .TITL   PICT2
03                                .PENT   PICT2
04                                .EXTN   ERROR,CON
05
06                ;THIS PROCEDURE OUTPUTS THE SECOND TEXT MESSAGE
07
08                                        .NREL
09
10 00000'163710 PICT2:            SAVE    0
11        000000
12
13 00002'024416                   LDA     1,.CON
14 00003'125120                   MOVZL   1,1       ;FORM A BYTEPOINTER
15 00004'044426                   STA     1,SCREEN+?IFNP ;AND STORE IT IN PKT
16                                ?OPEN   SCREEN
17 00011'002447                   JMP     @.ERROR
18                                ?WRITE  SCREEN
19 00016'002442                   JMP     @.ERROR
20 00017'127710                   RTN
21 00020'000000$.CON:             CON
22
23                ;SCREEN I/O PACKET
24
```

Figure 3-20. PICT1 and PICT2 Routines (continued)

```
25 00021'000014 SCREEN:          .BLK ?IBLT
26       000021'                 .LOC SCREEN+?ICH
27 00021'000000                  0
28       000022'                 .LOC SCREEN+?ISTI
29 00022'000011                  ?RTDY+?OFOT      ;WRITE DYNAMIC RECORDS
30       000023'                 .LOC SCREEN+?ISTO
31 00023'000000                  0
32       000024'                 .LOC SCREEN+?IBAD
33 00024'000072"                 MESSAGE*2        ;BYTEPOINTER TO MESG.
34       000025'                 .LOC SCREEN+?IRES
35 00025'000000                  0
36       000026'                 .LOC SCREEN+?IRCL
37 00026'000045                  37.              ;37 CHARS. LONG
38       000027'                 .LOC SCREEN+?IRLR
39 00027'000000                  0
40       000030'                 .LOC SCREEN+?IRNH
41 00030'000000                  0
42       000031'                 .LOC SCREEN+?IRNL
43 00031'000000                  0
44       000032'                 .LOC SCREEN+?IFNP
45 00032'000000                  0
46       000033'                 .LOC SCREEN+?IMRS
47 00033'177777                  -1
48       000034'                 .LOC SCREEN+?IDEL
49 00034'177777                  -1
50       000035'                 .LOC SCREEN+?IBLT
51              ;END OF PACKET
52
53 00035'040516 MESSAGE:         .TXT "AND THIS IS THE SECOND TEXT MESSAGE.<012>"
54       042040
55       052110
56       044523
57       020111
58       051440
59       052110
60       042440
```

Figure 3-20. PICT1 and PICT2 Routines (continued)

```
01                ;0002 PICT2
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17 00060'000000$.ERROR:          ERROR
18                               .END

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS



  0003 PICT2

CON    000002  XN     1/04     1/21
ERROR  000001  XN     1/04     3/17
MESSA  000035'        1/33     1/53
PICT2  000000' PN     1/03     1/10
SCREE  000021'        1/15     1/17     1/19     1/25     1/26     1/28     1/30
                      1/32     1/34     1/36     1/38     1/40     1/42     1/44
                      1/46     1/48     1/50
.CON   000020'        1/13     1/21
.ERRO  000060'        1/17     1/19     3/17
?OPEN  004315  MC     1/16
?WRIT  004436  MC     1/18
?XCAL  000001         1/17     1/19
```

Figure 3-20. PICT1 and PICT2 Routines (concluded)

End of Chapter

Chapter 4
Interprocess Communications

The operating system provides a facility for processes to communicate with each other. This facility, called *Interprocess Communications (IPC),* provides a means of sending free format messages of arbitrary length between any two processes.

This chapter describes IPC primitives for those people wishing to perform sophisticated process synchronization and intercommunication. An example of such an application is the system's use of the IPC for argument passing to created processes and/or passing results upon process termination.

Only the primitive calls to send an IPC message, ?ISEND and ?IS.R, require a user to have process privilege ?PVIP. Users not needing IPC primitives may still use the IPC, treating it like a standard peripheral device following device-independent I/O techniques. This use of the IPC is described in Chapter 6.

Interprocess communications are sent between ports. A *port* is a full-duplex communications path to a process. Each process can have up to 128 ports. Ports are identified using a port number. The system also permits each port number to be given a name. Names are assigned to port numbers by the ?CREATE call, described in Chapter 5.

When a message is transmitted by one process to another via the IPC, the sending process issues a send request to the system. In this send request, the process specifies the location of the IPC header. The IPC *header* is a parameter block that specifies information pertinent to the message transfer, such as the origin and destination port numbers and the length of the message.

Potential message receivers issue a complementary receive request which also specifies the location of a message header. If the origin and destination port numbers in the two headers match, then a message is sent between the two processes.

Portions of the sender's header are written over the receiver's header during each message transmission. In fact, some transmissions may consist solely of passing header information from sender to receiver.

Message-sending processes regain control immediately. When there is no matching receive for the message, by default the system spools the message until a suitable receive is issued. (By spooling we mean that the message is written to disk until an appropriate receive is issued.) A process can have a maximum of 35 messages spooled on disk at any one time. The sum of the sizes of all the messages spooled may not exceed 32,768 words. The system orders messages in a first-in, first-out (FIFO) manner. Alternatively, you can use an option in the ?ISEND packet to cause the system to signal an error if no outstanding receive exists.

The IPC facility also permits a process to send a message to itself for testing or other purposes.

IPC Header Structures

The send and receive IPC headers are each 7 words long and are defined as follows:

Send Header (for ?ISEND call)

Offset	Contents
?ISFL	System flags.
?IUFL	User flags.
?IDPH ?IDPL	Destination port number.
?IOPN	Origin port number.
?ILTH	Length of message in words.
?IPTR	Address of message.

Receive Header (for ?IREC call)

Offset	Contents
?ISFL	System flags.
?IUFL	User flags, copied from send header.
?IOPH ?IOPL	Origin port number.
?IDPN	Destination port number, translated from send header.
?ILTH	Length of buffer in words, copied from send header.
?IPTR	Address of buffer. (Entire buffer must be in unshared area of process issuing ?IREC call.)

You direct the action with certain bits in the *system flags* word, ?ISFL. By using the following bit masks, you can select these possible options:

Offset	Contents
?IFSTM	Send the message back to this process.
?IFNSP	Do not spool this message; signal an error if there is no ready receiver.

System flags are defined in an analogous manner for the receiver's IPC header:

Offset	Contents
?IFRFM	Receive an IPC message sent by this process to itself.
?IFSOV	Spool the message if the receive buffer is too small.
?IFNBK	Signal an error if no message is spooled for this receiver.

If ?IFSOV is not selected, then the message overflow will be discarded if the receive buffer is too small. If ?IFNBK is not selected, then the calling task will be suspended until a message is sent to it.

If a looped message transmission is to occur, i.e., one which is sent and received by the same process, then bits ?IFSTM and ?IFRFM in the send and receive headers must each be set to one. No port numbers need be specified in the send or receive headers; i.e., offsets ?IDPH, ?IDPL, ?IOPN, ?IOPH, or ?IOPL for looped transmissions.

To identify where a message comes from, use the bit masks ?IFPR (first bit) and ?IFGH (second bit) to examine the system flag word in the receiver's IPC header. These two bits work together in the following way:

- If ?IFPR = 0, then the system sent the message (and ?IFGH will be 0). Otherwise, another process sent the message.
- If ?IFPR = 1 and ?IFGH = 0, then the primary context sent the message.
- If ?IFPR = 1 and ?IFGH = 1, then the GHOST context sent the message.

The user flag word, ?IUFL, is simply copied from the transmitter's header to the receiver's header. This word may be used for any purpose mutually agreed upon by the sending and receiving processes. For a specific use of ?IUFL, see the section "Receiving Process Termination Messages," later in this chapter.

In the sender's header, ?ILTH and ?IPTR describe the length of the message and its starting address respectively. In the receiver's header, ?ILTH and ?IPTR indicate the size of the receiver's buffer and its starting address. If the sender's header indicates a message length (?ILTH) of zero words, then message transmission will only copy the sender's header (all words except ?ISFL) to the receiver's header. In this case, the message-pointer word, ?IPTR, can be used for transmitting 16 bits of data (using any conventions) just like the user flag word, ?IUFL. Whenever the message-length word is nonzero, word ?IPTR is not copied to the receiver's header.

Words ?IUFL and ?ILTH are always copied from the sender's header to the receiver's header. (This permits the headers themselves to transmit one or two words of information.) Thus the receiver can determine the length of the message which has been sent.

Words ?IDPH and ?IDPL, the global destination port number in the send header, are translated to a local destination port number and this number is put in word ?IDPN of the receive header. Likewise, word ?IOPN, the local origin port number in the send header, is translated to a two-word global origin port number and placed in words ?IOPH and ?IOPL of the receive header. (If you set bit ?IFSTM in word ?ISFL, all three port number words in the receive header will be -1.)

The copying and translating procedures are illustrated in Figure 4-1.

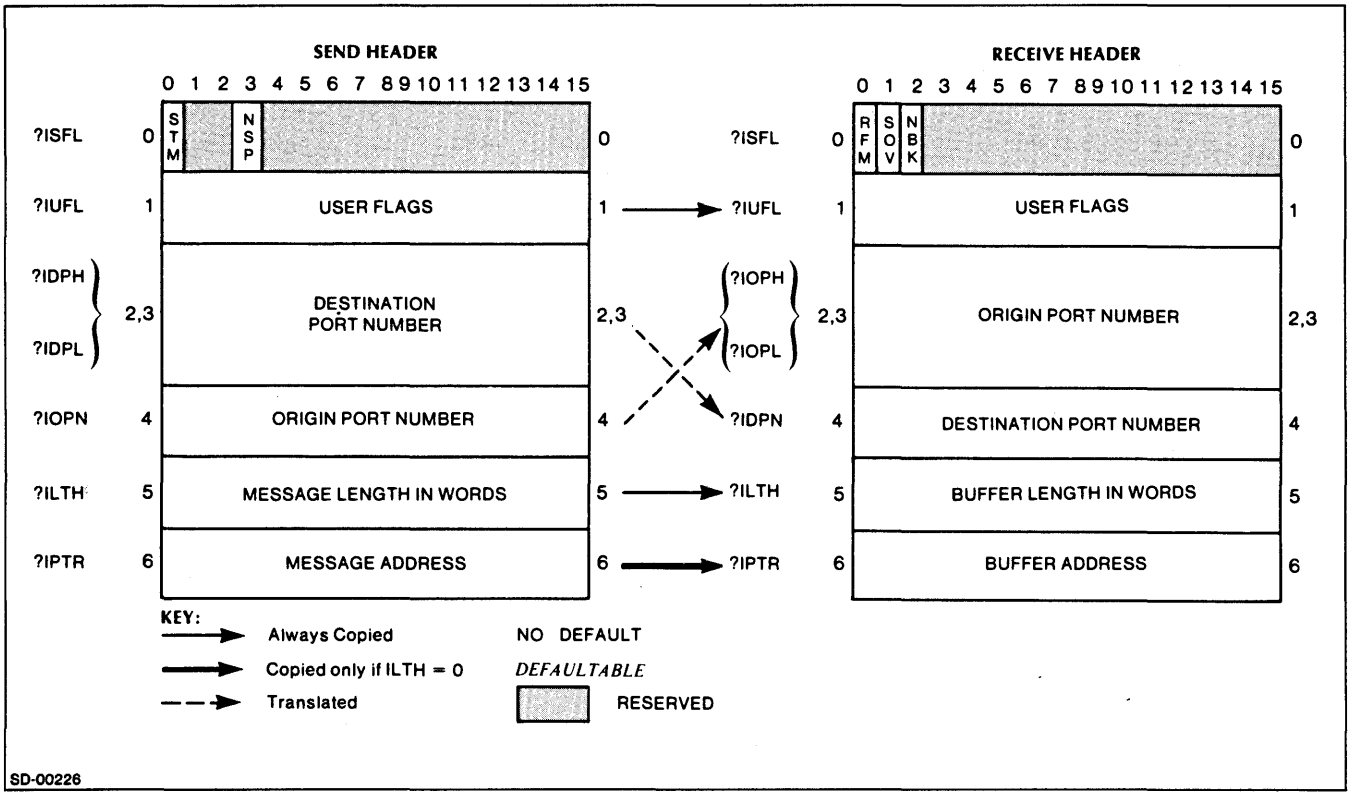


Figure 4-1. Send and Receive Header Structure

Port Numbers

There are two kinds of port numbers: local and global. A *global port number* is 32 bits long and is unique within the entire system. If one process refers to another process's port, then it must use a global port number.

A *local port number* is a number between 1 and 127 used by a process to refer to its own ports. The first time that a local port number is used in system calls ?ISEND, ?IREC, or ?TPORT, the system assigns a global port number to the port. The system maintains a table which enables it to translate local port numbers and PIDs into global port numbers, and vice-versa. System call ?TPORT can be used to perform this translation.

Port Matching Rules

When a process issues an ?IREC or ?ISEND system call, it specifies its local port number and the other process's global port number. The system translates the local port number into its global equivalent.

The message transmission will occur whenever a sender and a receiver have origin and destination ports which match. The matching of the port numbers can be either explicit or implicit.

An explicit match occurs when the origin and destination ports specified by both sender and receiver are the same.

An implicit match of port numbers may occur when one or both of the port numbers specified in the receiver header is a zero. A zero port number in the receiver's origin slot means that a message can be received from any sender. A zero port number in the receiver's destination slot means that a message can be received on any of the receiver's ports. At the completion of each transmission, the ports actually used will be copied into the receiver's header.

IPCs cannot be broadcast. If there are two or more potential implicit receivers of an IPC message, only the first will receive it.

If a process has more than one task waiting on ?IREC system calls, the system generally treats them in a FIFO manner when applying port matching rules. If, however, both port numbers of the receiving task are zero, all other tasks waiting for ?IREC system calls to complete that have specified either port to be nonzero will be examined for possible matches first. If more than one task has specified both port numbers to be zero, the system will process them in a FIFO order.

System ?ISEND and ?IREC Logic

The system calls, which implement IPC message sending and receiving, are ?ISEND and ?IREC respectively. Figure 4-2 charts the flow of operations when the system executes ?ISEND, while Figure 4-3 does the same for ?IREC.

The "move message" sequences in Figures 4-2 and 4-3 consist of the following steps.

First, the system moves as much of the message as will fit into the receiver's buffer. If there is enough room for the entire message, then the receiver's task is readied with control going to its normal return.

If the receiver's buffer is not sufficiently large to store the entire IPC message, control goes to its error return. If ?IFSOV is not set, the message overflow is lost. If ?IFSOV is set, the entire message is spooled. The receiver process may then increase the length of its buffer to the value copied to ?ILTH on the previous transmission and issue another ?IREC call.

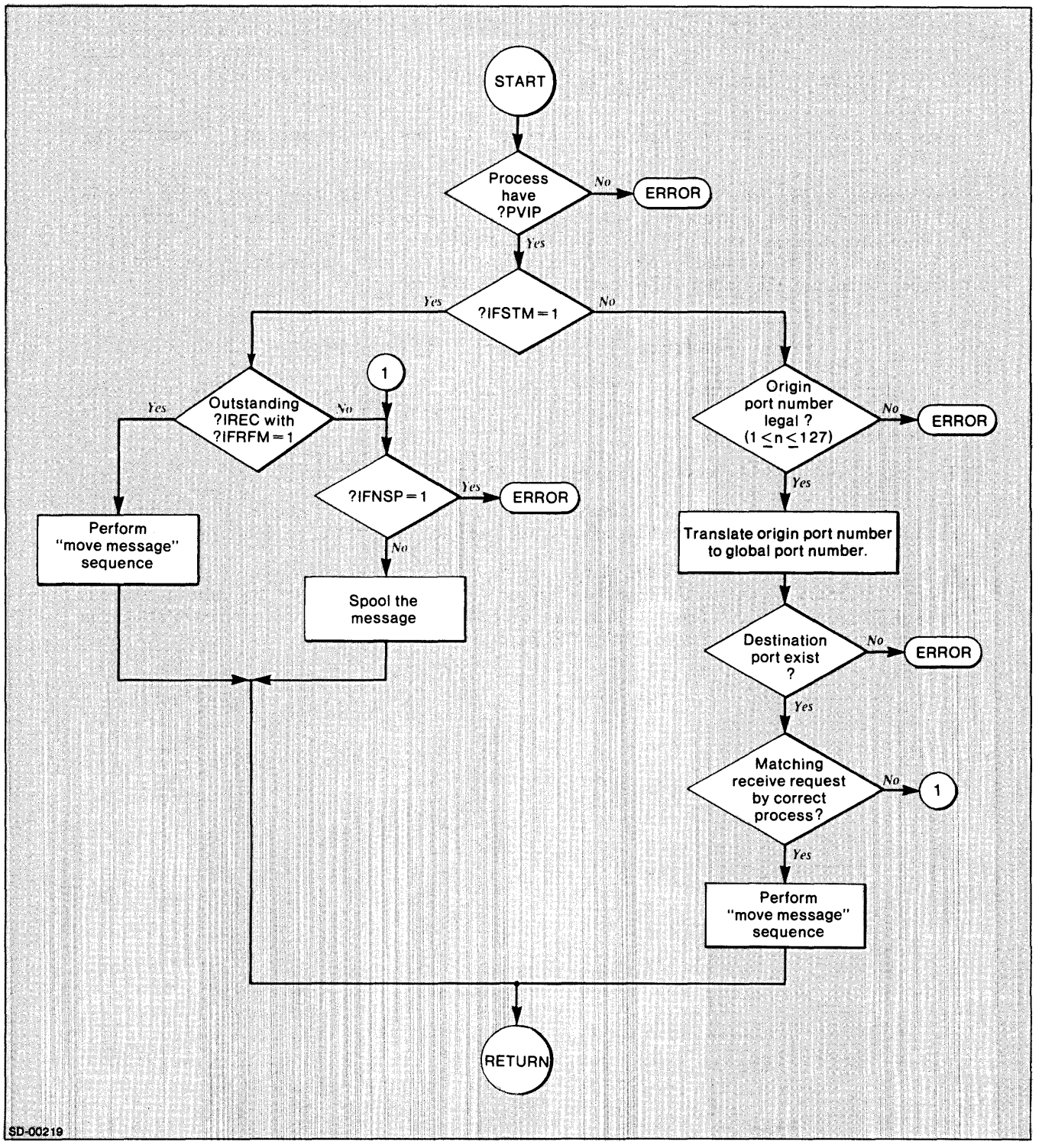


Figure 4-2. ?ISEND Logic

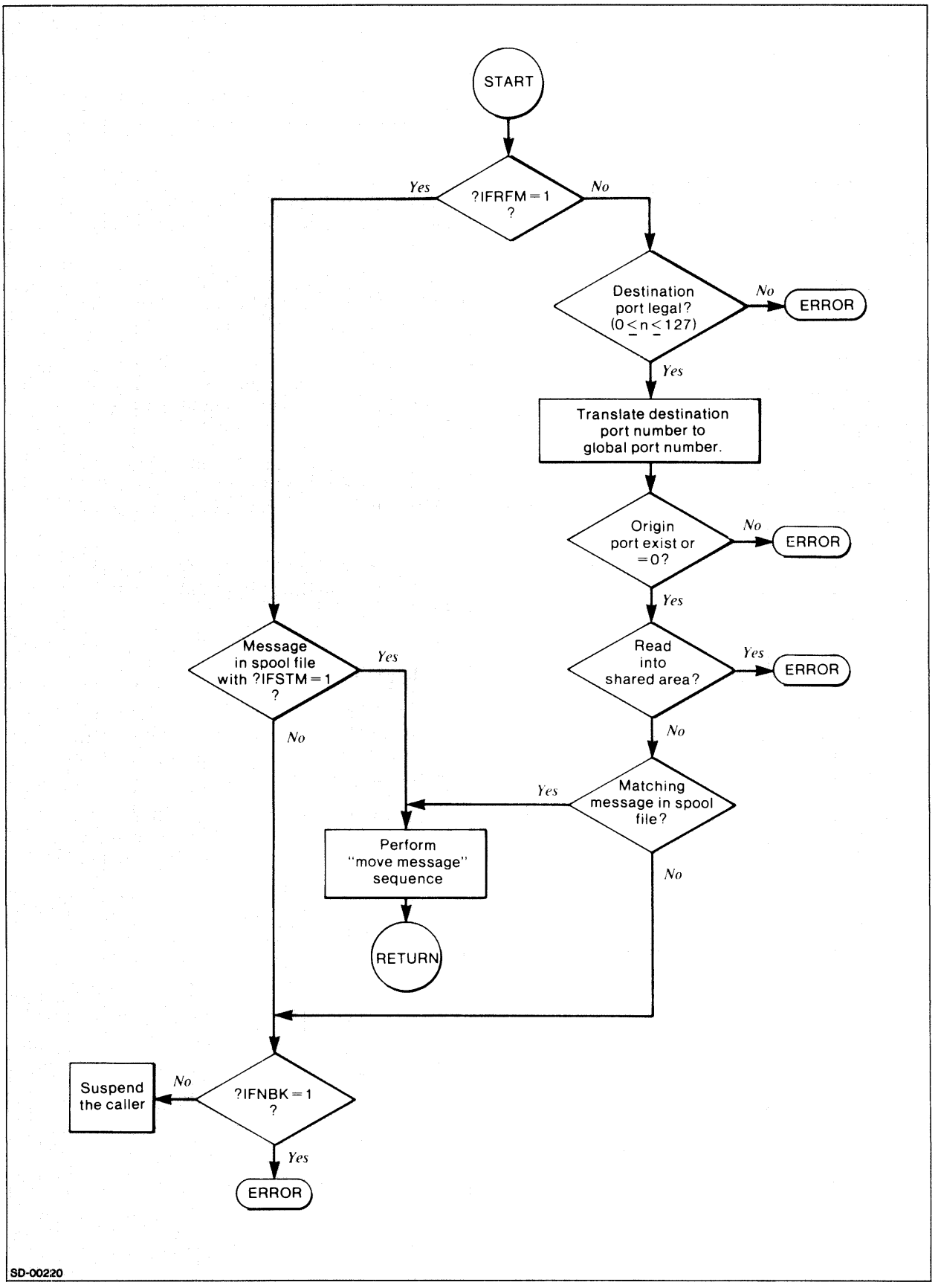


Figure 4-3. ?IREC Logic

Naming an IPC Port

As mentioned previously, in order for two processes to establish an IPC communication, they must each know the other's port number. This poses a problem for the initial communication between two processes, since neither has any way of knowing what port numbers the system assigned to the other process.

System call ?ILKUP provides a technique for solving this problem. If a process creates an IPC entry (using the ?CREATE system call) and assigns a name to this entry which is known by another process, then that other process can issue an ?ILKUP call to determine the actual port number which the name represents.

For example, suppose that process B, wishing to receive, creates an IPC entry PROCB equal to local port number 100. Process B then issues a receive request with 100 in its destination slot and zero in its origin slot. The system then translates this local port number to a global port number.

Process A wishing to send an IPC to process B, issues an ?ILKUP call with the name PROCB and determines the specific global port number to which PROCB is equivalent. A then transmits using this global port number. When B receives the IPC, B also receives process A's send port number, since the transmitter's header always overwrites the receiver's header.

If process B had wanted to ensure that it received from one specific process and only that process, then process B must have performed an ?ILKUP call to some other IPC entry which had been created by the appropriate sender. Both the sender and receiver processes would have specified port numbers (instead of zero).

The system call ?GCPN finds the port number associated with the console of a particular process. To use this call, supply the process name or PID in AC0. The system returns the appropriate port number in AC1 (the port number's high order bits) and AC2 (the low order bits).

Receiving Process Termination Messages

Whenever any process is terminated, an IPC message is sent to that process's father. Depending upon the reason for the termination, the message will be constructed in different ways. In every case, if any father process wants to receive a termination message notifying it of a son's demise, the father must issue an ?IREC call on global origin port ?SPTM and local destination port 0.

Termination message information is always supplied in the IPC header user flag word ?IUFL. Optionally, it may also be supplied in the IPC message text. User information is supplied as shown in Figure 4-4.

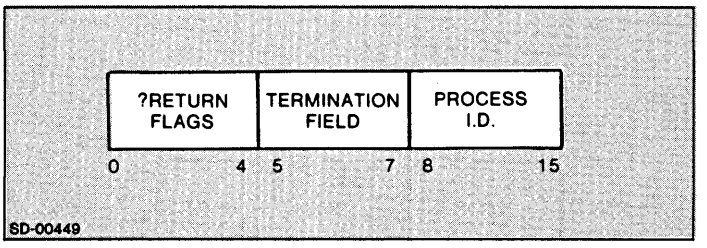


Figure 4-4. ?IUFL Contents

The PID of the terminated process will always be supplied in the right byte of the flag word. Bits 5 through 7 indicate the cause of the process's termination:

0 The process terminated itself, either by a ?TERM or by a ?RETURN call.

1 The process was terminated by a user trap; the type of trap is indicated in word 5 of the IPC message (described below).

2 The process was terminated by a console interrupt.

3 The process was terminated by a superior process which issued a ?TERM call.

4 The process was terminated by the system because of an error. The error code is indicated in ?IPTR.

5-7 Currently undefined.

If the process terminated itself by a ?RETURN call, then the flag bits input in AC2 to ?RETURN will be copied into the ?RETURN flags field. If CLI is the father, then by convention these flag bits are defined as follows:

?RFCF Message is in CLI format.

?RFEC Error code in AC0.

?RFWA Warning condition.

?RFER Error condition.

?RFAB Abort condition.

If CLI is not the father (i.e., ?RFCF is not selected), then bits 0-4 in the flag word are returned and their interpretation is by any convention agreed to by the terminated son and its father.

If ?RETURN is issued with a message that is in CLI format, then the message is prefixed with a two-word header:

word 0 Byte length of the message;

word 1 Contents of AC0 input to ?RETURN (error code);

word 2 Start of the message text.

If a zero-length message is transmitted, then the ?RETURN message consists solely of words 0 and 1.

If the process terminated itself by a ?TERM, then information will be passed to the father in one of two ways. If the son provided no IPC header (and therefore no message), the system will provide the termination information in the father's IPC user flag word, ?IUFL. This information will be as shown above; bits 0 through 4 will be zeros. If the son provided an IPC header and specified a nonzero message (in ?ILTH of the son's IPC header), then the message will be deposited in the father's buffer.

If the process was terminated by a trap, then in addition to the above-described user flag information, a message is returned to the father. This information has the following format:

word 0 Contents of AC0 when the trap occurred.

word 1 Contents of AC1 when the trap occurred.

word 2 Contents of AC2 when the trap occurred.

word 3 Contents of AC3 when the trap occurred.

word 4 Bit 0, carry; bits 1-15, PC.

word 5 Trap description:

bit 0=1

Trap occurred while control was in the operating system. This is an unusual condition which should not occur.

bit 0=0

Trap occurred while control was in the user context.

bit 11=1

The system detected an error in one of the following system databases: User Status Table, Task Control Blocks. If this bit is set, the contents of words 0-4 are not meaningful. You can avoid this trap by ensuring that these databases, which are found in the user context starting at location 400, are never modified by your program.

bit 12=1

Attempt to write into a write-protected area.

bit 13=1

Memory MAP validity error, such as an attempt to refer to an address which is outside the user context.

bit 14=1

Defer error, more than 14 levels of indirection in an address reference.

bit 15=1

Attempt to issue a machine-level I/O instruction without first having issued the system call ?DEBL (see Chapter 8).

Recall that MAP validity errors are possible even when the highest address of a context is octal 77777, because a context of less than 32 pages may have a logically void area between its unshared and shared portions (see Figure 3-1 of Chapter 3). That is, locations following the last unshared location and preceding the first shared location will be unaccessible.

Finally, upon a termination by either a console interrupt or a ?TERM issued by a superior process, information is passed to the father only in the flag word. That is, ?ILTH in the header will contain zero.

System Call Summary

The following system calls let you manage interprocess communications:

?GCPN	Take a process name or process ID, and return the port number of the process's console.
?GPORT	Obtain the ID of a process to which a global port number is assigned.
?ILKUP	Determine the port number associated with a name.
?IREC	Receive an IPC message.
?ISEND	Send an IPC message.
?IS.R	Send and receive an IPC message.
?TPORT	Translate a local port number into its 32-bit global equivalent.

?GCPN

Get a console port number.

?GCPN
exception return
normal return

Input/Output

Input:

AC0	PID or byte pointer to process name, or -1 (for current process).
AC1	0, if AC0 is PID; -1, if AC0 is a byte pointer.
AC2	reserved (must be 0).

Output:

AC0	unchanged.
AC1	console port number (high order bits).
AC2	console port number (low order bits).

Exceptional Condition Codes in AC0

ERFDE	The file (process name or PID) does not exist.
ERPRH	You attempted to access a process not in the hierarchy.

Description

This system call takes a process name or PID and returns the port number of that process's console. The system resolves the pathname or PID specified in AC0. If the system fails to find the name or ID, or if the process is not an IPC type entry, it returns either error code ERFDE or ERPRH.

?GPORT

Find the owner of a port.

?GPORT
exception return
normal return

Input/Output

Input:

AC1, AC2 — global port number

Output:

AC1 — PID of process owning the port.

AC2 — local port number

Exceptional Condition Codes in AC0

ERIPV — The port number is not currently assigned.

Description

This call accepts a global port number and returns the PID of the process to which the port is assigned and its local port number.

?ILKUP

Look up a port number.

?ILKUP
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname.

Output:

AC0, AC1 port number.

AC2 entry type.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This system call determines the port number that was associated with a name. The system resolves the pathname pointed to by AC0. If the name is not found, or if the entry found is not an IPC type entry, then the system returns the error code ERFDE.

?IREC

Receive an interprocess communication.

?IREC *[header address]*
exception return
normal return

Input/Output

Input:

AC2 address of IPC header.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERMPR Address of receive buffer is not in unshared area of calling process.

IPC EXCEPTIONAL CONDITION codes

Description

This call is issued by a process wishing to receive an interprocess communication. If the origin port number equals zero, then a message can be received from any sender. If the destination port number equals zero, then the IPC message can be received on any of the receiver's ports. When the transmission occurs, the actual origin and destination port numbers will be written into the receiver's header, displacements ?IOPH through ?IDPN.

The structure of the receiver's header (see Figure 4-1, shown previously) is as follows:

Offset	Contents
?ISFL	System flags.
?IUFL	User flags.
?IOPH ?IOPL	Origin port number.
?IDPN	Destination port number.
?ILTH	Size of receive buffer, in words.
?IPTR	Address of receive buffer. (Entire buffer must be in unshared area of process issuing ?IREC call.)

?ISEND

Send an interprocess communication.

?ISEND *[header address]*
exception return
normal return

Input/Output

Input:

AC2 address of the IPC header.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERPRV Caller not privileged for this action.

IPC EXCEPTIONAL CONDITION codes

Description

This call sends an interprocess communication. The structure of the sender's IPC header (see Figure 4-1) is as follows:

Offset	Contents
?ISFL	System flags.
?IUFL	User flags.
?IDPH, ?IDPL	Destination port number.
?IOPN	Origin port number.
?ILTH	Message word length.
?IPTR	Message address.

The caller must have privilege ?PVIP to issue this call successfully.

?IS.R

Send and then receive an interprocess communication.

?IS.R
exception return
normal return

Input/Output

Input:

AC2 address of IPC header.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERPRV Caller not privileged to make this call.

IPC EXCEPTIONAL CONDITION codes

Description

This call performs an ?ISEND and, if it is successful, immediately follows it with an ?IREC with the same ports. The calling task is suspended until the ?IREC is complete or an exception return is taken. The header for this call is identical to an ?ISEND header, except that it is followed by two more words: ?IRLT (the length of the receive buffer, in words) and ?IRPT (a pointer to the receive buffer's address).

In many cases you will want a process to issue an ?ISEND on some channel and then wait for a reply on the same channel; this call allows efficient IPC communication in these cases by combining the two calls into one.

?TPORT

Translate a port number.

?TPORT
exception return
normal return

Input/Output

Input:

AC2 local port number.

Output:

AC1, AC2 } global port number.

Exceptional Condition Codes in AC0

ERIVP Invalid port number.

Description

This call will return the 32-bit global port number equivalent to the given local port number. If no global port number has been assigned, this call will assign one.

Examples

Here we show how two processes can intercommunicate using the IPC facility. The first process runs program HEAR.PR, shown in Figure 4-5. HEAR calls three external routines: ERROR and SON (both described in Chapter 2), and DELAY (illustrated in Chapter 9). DELAY simply makes a program pause for a specified number of seconds.

The sequence of operations which HEAR.PR performs are as follows. First, HEAR creates a son process to run SPEAK.PR. HEAR then creates an IPC entry named PORTR for receiving SPEAK's message. By prearrangement, HEAR knows that SPEAK's send port will be named PORTS. Using this information, HEAR looks up the entry number corresponding to PORTS by issuing ?ILKUP, and builds its own receive IPC header. HEAR arbitrarily assigned 1 to be the local port number corresponding to PORTR.

Having set up the header, HEAR waits for SPEAK's message by issuing a ?IREC. HEAR needs to know the size of this message, since it did not set the spool bit in the header. If (without the spool bit set) HEAR does not know how long the message is, HEAR could receive a truncated message (buffer too small).

Note that a common way to receive messages of unknown length is to set the spool bit ?IFSOV and initially specify a zero-length buffer. This will ensure that the message is spooled. Recall too that the sender's header, words ?IUFL through ?ILTH, are always copied to the receiver's header. This enables the receiver to determine the precise length of the incoming message and reissue ?IREC.

After receiving the message, HEAR terminates itself and returns this message to the CLI for display on the console screen:

HELLO, THIS IS YOUR SON SPEAKING

AS YOU READ THESE WORDS, I AM TERMINATED
AND SO ARE YOU

SPEAK goes through a similar chain of operations (see Figure 4-6). First, it creates an IPC entry for sending (PORTS, with local port number 1). SPEAK then sets up its send header and sends the above message. Following this, SPEAK terminates itself by issuing ?TERM.

```
 0001  HEAR      AOS ASSEMBLER REV 03.40          12:40:00 02/13/81
                          .TITL   HEAR
 02                       .EXTN   ERROR,SON,DELAY
 03
 04                ;THIS PROGRAM CALLS "SON" TO CREATE A SON PROCESS
 05                ;(TO RUN "SPEAK"), AND THEN LISTENS FOR AN IPC MESSAGE
 06                ;FROM THAT SON. "HEAR'S" DESTINATION PORT IS
 07                ;"PORTR", ORIGIN PORT IS "PORTS". FATHER SENDS
 08                ;SON'S IPC MSG TO CLI UPON TERMINATION.
 09
 10
 11                       .NREL
 12 00000'020441 HEAR:    LDA     0,.SPEAK         ;CREATE SON PROCESS
 13 00001'006446          JSR     @.SON    ;TO EXECUTE "SPEAK.PR"
 14 00002'020457 GETOP:   LDA     0,.PTS   ;GET ORIGIN PORT NUMBER
 15                       ?ILKUP
 16                ;CHECK TO SEE IF ERROR "ERFDE" (ENTRY NOT CREATED YET)
 17 00005'000426          JMP     TEST
 18 00006'040463          STA     0,RHDR+?IOPH     ;SET UP IPC RECEIVE HEADER
 19 00007'044463          STA     1,RHDR+?IOPL     ;ORIGIN PORT #'S
 20 00010'064001          LEF     1,1              ;SET UP DESTINATION PORT#
 21 00011'044462          STA     1,RHDR+?IDPN
 22 00012'020436          LDA     0,.PTR
 23                       ?CREATE IPCEN            ;CREATE AN IPC ENTRY
 24                ;NAMED "PORTR" FOR RECEIVE. ?CREATE IS DESCRIBED IN
 25                ;"FILE CREATION AND MANAGEMENT"
 26
 27 00017'002435          JMP     @.ERROR
 28                       ?IREC   RHDR             ;NOW RECEIVE SPEAK'S MSG.
 29 00024'002430          JMP     @.ERROR
 30 00025'126470          ELDA    1,.MSBUF         ;TERMINATE SELF, RETURN MSG.
 31       000051
 32 00027'030447          LDA     2,FLAGS
 33                       ?RETURN
 34 00032'002422          JMP     @.ERROR
 35 00033'024432 TEST:    LDA     1,.ERFDE
 36 00034'106404          SUB     0,1,SZR
 37 00035'002417          JMP     @.ERROR          ;?ILKUP FAILED FOR OTHER REASON
 38 00036'064012          LEF     1,10.            ;SEND PORT NOT BUILT, WAIT
 39 00037'006427          JSR     @.DLA            ;10 SECONDS AND
 40 00040'000742          JMP     GETOP            ;AND LOOK UP AGAIN.
 41
 42 00041'000104".SPEAK:  .+1*2
 43 00042'051520          .TXT "SPEAK.PR"
 44       042501
 45       045456
 46       050122
 47       000000
 48 00047'000000$.SON:    SON
 49 00050'000122".PTR:    .+1*2
 50 00051'050117          .TXT "PORTR"
 51       051124
 52       051000
 53 00054'000000$.ERROR:  ERROR
 54
 55                ;PACKET TO CREATE AN IPC ENTRY
 56       000055'IPCEN:   .LOC     IPCEN+?CFTYP
 57 00055'000036          ?FIPC    ;TYPE IS "IPC"
 58       000056'         .LOC     IPCEN+?CTIM
 59 00056'177777          -1       ;TIME BLOCK SET TO CURRENT TIME
 60       000057'         .LOC     IPCEN+?CACP
```

Figure 4-5. HEAR Routine (continues)

```
 0002  HEAR
01 00057'177777          -1       ;ACL CONTAINS USER'S NAME AND ALL PRIVILEGES
02        000060'        .LOC     IPCEN+?CPOR
03 00060'000001          1        ;LOCAL DESTINATION PORT #
04 00061'000144".PTS:    .+1*2
05 00062'050117          .TXT "PORTS"
06        051124
07        051400
08 00065'000025 .ERFDE:  ERFDE    ;IPC ENTRY NOT YET BUILT BY SPEAK
09 00066'000000$.DLA:    DELAY
10        000067'RHDR:   .LOC     RHDR+?ISFL
11 00067'000000          0        ;NO SYSTEM FLAGS
12        000070'        .LOC     RHDR+?IUFL
13 00070'000000          0        ;NO USER FLAGS EITHER
14        000071'        .LOC     RHDR+?IOPH
15 00071'000000          0        ;ORIGIN PORT NUMBER
16        000072'        .LOC     RHDR+?IOPL
17 00072'000000          0        ;  "      "      "
18        000073'        .LOC     RHDR+?IDPN
19 00073'000000          0        ;DESTINATION PORT #
20        000074'        .LOC     RHDR+?ILTH
21 00074'000062          50.      ;MESSAGE BUFFER SIZE.
22        000075'        .LOC     RHDR+?IPTR
23 00075'000100'         MSBUF    ;POINTER TO MSG. BUFFER
24
25 00076'100144 FLAGS:   ?RFCF+100.       ;MESSAGE IS 100 CHARS. LONG
26 00077'000200".MSBUF:  MSBUF*2
27 00100'000063 MSBUF:   .BLK     51.
28                       .END HEAR

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS


  0003 HEAR

DELAY 000003  XN     1/02     2/09
ERROR 000001  XN     1/02     1/53
FLAGS 000076'        1/32     2/25#
GETOP 000002'        1/14#    1/40
HEAR  000000'        1/12#    2/28
IPCEN 000055'        1/24     1/56#    1/58     1/60     2/02
MSBUF 000100'        2/23     2/26     2/27#
RHDR  000067'        1/18     1/19     1/21     1/29     2/10#    2/12     2/14
                     2/16     2/18     2/20     2/22
SON   000002  XN     1/02     1/48
TEST  000033'        1/17     1/35#
.DLA  000066'        1/39     2/09#
.ERFD 000065'        1/35     2/08#
.ERRO 000054'        1/27     1/29     1/34     1/37     1/53#
.MSBU 000077'        1/30     2/26#
.PTR  000050'        1/22     1/49#
.PTS  000061'        1/14     2/04#
.SON  000047'        1/13     1/48#
.SPEA 000041'        1/12     1/42#
?CREA 000413  MC     1/23
?ILKU 001236  MC     1/15
?IREC 001203  MC     1/28
?RETU 006360  MC     1/33
?XCAL 000001         1/16     1/24     1/29     1/34
```

Figure 4-5. HEAR Routine (concluded)

```
 0001  SPEAK     AOS ASSEMBLER REV 01.09          12:28:44 12/05/78
01
                                    .TITL   SPEAK
03                                  .EXTN   ERROR, DELAY
04                  ;THE SOLE PURPOSE OF THIS PROGRAM IS TO SEND AN IPC
05                  ;MESSAGE TO A BROTHER PROCESS, AND THEN TO TERMINATE
06                  ;ITSELF. SPEAK'S ORIGIN PORT IS "PORTS", AND ITS
07                  ;DESTINATION PORT, "PORTR".
08
09                                  .NREL
10 00000'020437             SPEAK:  LDA     0,.PTO
11                                  ?CREATE  IPCNT              ;CREATE AN IPC ORIGIN
12                                                      ;PORT NAMED "PORTS" FOR SEND.
13 00005'002436                     JMP     @.ERROR
14 00006'020436             GETNM:  LDA     0,.PTD              ;NOW LOOK UP DEST.
15                                  ?ILKUP                      ;PORT.
16 00011'000417                     JMP     TEST    ;SEE IF ENTRY NOT CREATED YET.
17 00012'040441                     STA     0,SHDR+?IDPH    ;SET UP DEST.PORT.#
18 00013'044441                     STA     1,SHDR+?IDPL
19 00014'064001                     LEF     1,1                 ;SET UP ORIGIN PORT #.
20 00015'044440                     STA     1,SHDR+?IOPN
21                                  ?ISEND  SHDR
22 00022'002421                     JMP     @.ERROR
23 00023'102000                     ADC     0,0                 ;TERMINATE SELF
24 00024'152400                     SUB     2,2                 ;NO IPC MSG.TO FATHER
25                                  ?TERM                       ;THE END.
26 00027'002414                     JMP     @.ERROR
27 00030'024420             TEST:   LDA     1,.ERFDE
28 00031'106414                     SUB#    0,1,SZR
29 00032'002411                     JMP     @.ERROR   ;?ILKUP FAILED,OTHER REASON
30 00033'064012                     LEF     1,10.     ;RECEIVE PORT NOT BUILT
31 00034'006402                     JSR     @.DLA               ;WAIT 10 SECS.,
32 00035'000751                     JMP     GETNM               ;AND TRY AGAIN.
33 00036'000000$            .DLA:   DELAY
34
35 00037'000100"            .PTO:   .+1*2
36 00040'050117                     .TXT    "PORTS"
37       051124
38       051400
39
40 00043'000000$            .ERROR: ERROR
41 00044'000112"            .PTD:   .+1*2
42 00045'050117                     .TXT    "PORTR"
43       051124
44       051000
45
46 00050'000025             .ERFDE: ERFDE
47                  ;IPC HEADER
48       000051'            SHDR:   .LOC    SHDR+?ISFL          ;NO SYSTEM FLAGS.
49 00051'000000                     0
50       000052'                    .LOC    SHDR+?IUFL          ;NO USER FLAGS
51 00052'000000                     0
52       000053'                    .LOC    SHDR+?IDPH          ;DEST. PORT #
53 00053'000000                     0
54       000054'                    .LOC    SHDR+?IDPL          ; "      "  "
55 00054'000000                     0
56       000055'                    .LOC    SHDR+?IOPN          ;ORIGIN PORT #
57 00055'000000                     0
58       000056'                    .LOC    SHDR+?ILTH          ;MESSAGE WORD LENGTH.
59 00056'000062                     50.
60       000057'                    .LOC    SHDR+?IPTR          ;PNTR.TO BUFFER
```

Figure 4-6. SPEAK Routine (continues)

```
  0002  SPEAK
 01 00057'000064'                MSBUF

!0003  SPEAK
01
02                ;PKT TO CREATE IPC ENTRY
03        000060'         IPCNT:  .LOC    IPCNT+?CFTYP    ;TYPE IS
04 00060'000036                   ?FIPC                   ;"IPC"
05        000061'                 .LOC    IPCNT+?CTIM     ;TIME BLOCK AT
06 00061'177777                   -1                      ;CURRENT TIME
07        000062'                 .LOC    IPCNT+?CACP     ;ACL =USERNAME
08 00062'177777                   -1                      ;+ ALL PRIVILEGES.
09        000063'                 .LOC    IPCNT+?CPOR     ;LOCAL ORIGIN
10 00063'000001                   1                       ;PORT #.
11
12 00064'044105         MSBUF:  .TXT "HELLO, THIS IS YOUR SON SPEAKING.<012>
13        046114
14        047454
15        020124
16        044111
17        051440
18        044523
19        020131
20        047525
21        051040
22        051517
23        047040
24        051520
25        042501
26        045511
27        047107
28        027012
29        000105'
30        000105'
31        000105'
32        000105'
33        000105'
34        000105'
35        000105'
36        000105'
37        000105'
38        000105'
39        000105'
40        000105'
41        000105'
42        000105'
43        000105'
44        000105'
45        000105'
46        000105'
47        000105'
48 00105'040523 AS YOU READ THESE WORDS, I AM TERMINATING <012>
49        020131
50        047525
51        020122
52        042501
53        042040
54        052110
55        042523
56        042440
57        053517
58        051104
59        051454
60        020111
```

Figure 4-6. SPEAK Routine (continued)

```
 0004  SPEAK
01        020101
02        046440
03        052105
04        051115
05        044516
06        040524
07        044516
08        043440
09        000132'
10        000132'
11        000132'
12        000132'
13        000132'
14        000132'
15        000132'
16        000132'
17        000132'
18        000132'
19        000132'
20        000132'
21        000132'
22        000132'
23        000132'
24        000132'
25        000132'
26        000132'
27        000132'
28 00132'005101 AND SO ARE YOU. <012>"
29        047104
30        020123
31        047440
32        040522
33        042440
34        054517
35        052456
36        020012
37        000000
38
39
40
41
42
43                                .END    SPEAK

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS
```

Figure 4-6. SPEAK Routine (continued)

```
  0005 SPEAK

DELAY 000002  XN     1/03    1/33
ERROR 000001  XN     1/03    1/40
GETNM 000006'        1/14    1/32
IPCNT 000060'        1/12    3/03    3/05    3/07    3/09
MSBUF 000064'        2/01    3/12
SHDR  000051'        1/17    1/18    1/20    1/22    1/48    1/50    1/52
                     1/54    1/56    1/58    1/60
SPEAK 000000'        1/10    4/43
TEST  000030'        1/16    1/27
.DLA  000036'        1/31    1/33
.ERFD 000050'        1/27    1/46
.ERRO 000043'        1/13    1/22    1/26    1/29    1/40
.PTD  000044'        1/14    1/41
.PTO  000037'        1/10    1/35
?CREA 000406  MC     1/11
?ILKU 001170  MC     1/15
?ISEN 001102  MC     1/21
?TERM 005202  MC     1/25
?XCAL 000001         1/12    1/16    1/22    1/26
```

Figure 4-6. SPEAK Routine (concluded)

End of Chapter

Chapter 5
File Creation and Management

A *file* is a collection of related data treated as a unit. Each file has a *filename* by which the system and the user can address it. This chapter deals with a mechanism for organizing files into higher-level structures and with the manipulation of these structures. The next chapter discusses the actual storage and retrieval of file data.

Devices are the physical means for storing and retrieving information. There are two distinct kinds of devices: those used strictly for input and/or output (such as consoles and card readers), and those used both for I/O and data storage and retrieval. The second kind is commonly called a *multifile device* and includes disks and magnetic tape units. The term *magnetic tape* used here includes 7- and 9-track tape units; the term *disk* refers to all types of disk units, including diskettes.

Disk File Structures

The following information is useful to you when you create files. The system and system utilities also create files and directories, and structure them to meet requirements of the operating system.

Every disk file in the operating system is built from one or more 256-word (512-byte) disk blocks. The system ties together disk blocks within files by employing a hierarchical index.

The basic unit of storage in this file organization is a *file element* consisting of one or more contiguous blocks (i.e., blocks with sequential physical addresses). The file element size is specified when a file is created. The system allocates file space in multiples of file elements. Thus if a file with a file element size of 8 grows, it will grow in units of 8 blocks.

Consider a file created with an element size of two. Initially, this file consists of two contiguous blocks (see Figure 5-1). If file storage requirements exceed the two blocks, an index block is allocated which contains the addresses of each of the file elements; the index element provides no data storage of itself, and each index element is one block (*not* one file element size) in length.

As data storage requirements continue to grow, more file elements will be allocated and added to the list maintained in the index. If all space in the index becomes exhausted, another level of indexing will be added. Pointers in the topmost level index element will then point to further index elements. Adding index levels as the need for file data storage increases can continue to a maximum of three levels.

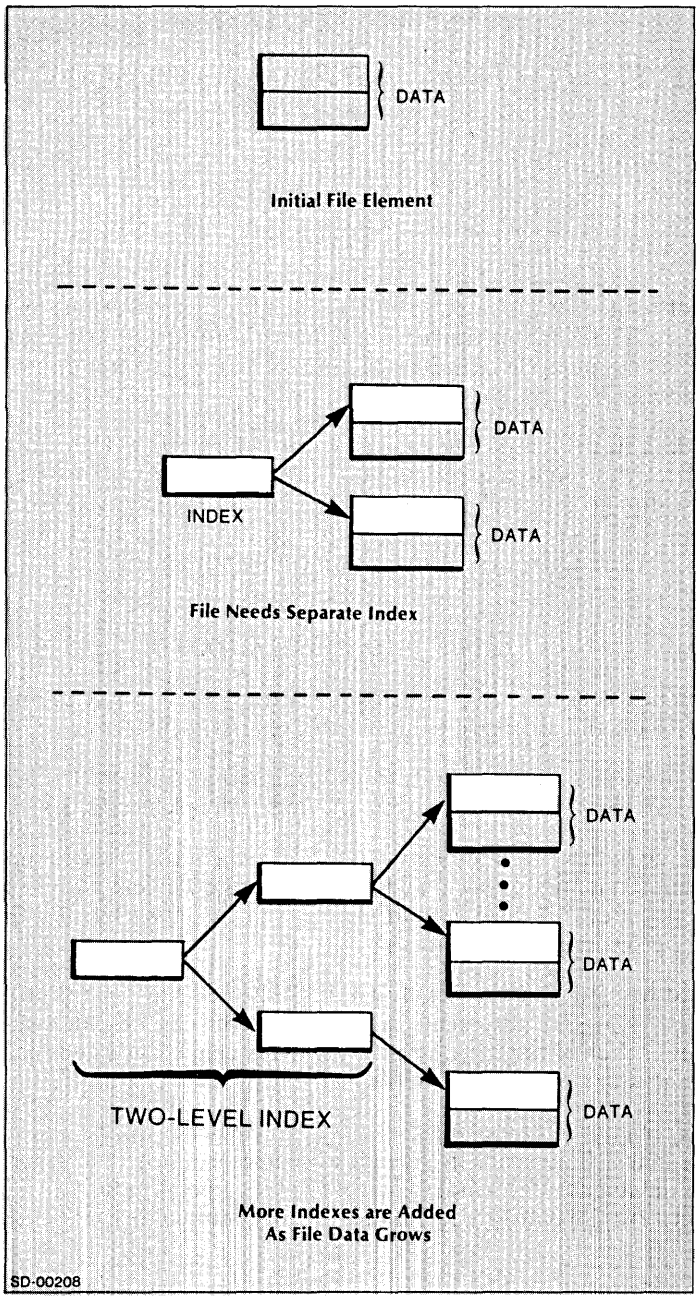


Figure 5-1. Stages in File Growth

In general, you will never be aware of index elements; the system stores and retrieves information so that you need not be concerned with the actual linkage between disk blocks. However, you should be aware of file organization, since it determines the tradeoffs in designing file structures. For example, large file element sizes could be used for creating data storage in contiguous disk areas that do not need indexes; these could be accessed quickly. Smaller file element sizes require more disk accesses, but permit more efficient utilization of disk storage. (It is usually easier to find four free blocks that are at sequential addresses than it is to find 500 free blocks at sequential addresses.)

The largest possible disk file would contain 2**24 disk blocks. Also, as might be expected, the total disk storage in a system is never available for user data storage, since some storage is always required by the system itself for purposes such as the disk bootstrap and block allocation structures.

Naming Files

Each disk file is given one name, a *filename*. Each filename is a byte string of from 1 to 31 ASCII characters. Legal characters include: upper- or lowercase alphabetics, numerics, period (.), dollar sign ($), question mark (?), and underscore (_). In filenames, upper- and lowercase alphabetic characters are considered to be the same. Thus the following three filenames are equivalent: Adam, ADAM, aDaM.

Groups of users are free to select any desired conventions for naming files or file families. System utilities adopt the following convention for naming certain types of files:

Type of file	File Extension
assembly language source file	.SR
CLI macro file	.CLI
object file	.OB
program file	.PR
temporary files	.TMP (filename must begin with a ?)

You produce *source files* with a text editor utility program or the CLI CREATE command, then you assemble or compile the program to produce an *object file*. The object file is made into an executable *program file* by linking it with optional libraries using the Link utility. Processes that require short-term disk storage should use *temporary files*.

Directory Hierarchy Structure

There are two types of disk files: directories and all other files. A *directory* is a file which contains and catalogs information used to access other files and other devices.

All directories are connected in a structure resembling an inverted tree. The top or root of this structure is a directory called the *system root directory*. All other directories are inferior to the system root, and each directory except the root is pointed to by an entry in some higher directory. Higher directories are called superior, and each directory that is pointed to by an entry in a superior directory is termed an inferior directory. (The terms *superior* and *inferior* denote relative positioning within a tree.) Each superior directory is the parent of its immediately inferior directories.

Directory Entries

Each directory contains entries for all of its son files. These entries include the file's name, information needed to access the data contained in the file, and other miscellaneous information such as which users may access the file, and the types of access that they are permitted. You can find all the names in a directory by using the ?GNFN call, described in Chapter 6.

Directories also contain other types of entries, such as peripheral entries, IPC entries, link entries, and unit entries. These entries contain a name and other information unique to that type of entry. For example, an IPC entry contains the port number associated with that name, and a link entry contains a pathname segment to be substituted for the link name. Up to 256 different types of directory entries can be defined for use within the operating system. Of this number, types 0 through 127 are reserved for definition by Data General Corporation; these are defined in the user parameter file, PARU.SR. All currently defined file types are listed in Table 5-1. Users may define the remaining types, which are numbered 128 through 255 inclusive.

Table 5-1. File Types Defined by AOS

Mnemonic (Octal Value)	Type	Comments
?FCON (61)	Console (hard-copy or CRT)	Cannot be created with ?CREATE.
?FCPD (14)	Control Point Directory	Explained in this chapter.
?FCRA (45)	Card Reader	Cannot be created with ?CREATE.
?FDIR (12)	Disk Directory	Explained in this chapter.
?FDKU (24)	Disk Unit	
?FFCC (110)	FORTRAN Carriage Control Files	
?FGFN (3)	Generic Filename	Explained in Chapter 6.
?FGLT (42)	Generic Labeled Tape	Explained in Chapter 6.
?FIPC (36)	IPC Port Entry	Explained in Chapter 4.
?FLCC (107)	FORTRAN Carriage Control Files	
?FLDU (13)	Logical Disk	Cannot be created with ?CREATE.
?FLNK (0)	Link	Explained in this chapter.
?FLOG	System Log File	Used by the system.
?FLPA (54)	Programmed I/O Line Printer	Cannot be created with ?CREATE.
?FLPC (55)	LP2 Line Printer	Cannot be created with ?CREATE.
?FLPD (30)	Data Channel LP2 Unit	Cannot be created with ?CREATE.
?FLPU (27)	Data Channel Line Printer	Cannot be created with ?CREATE.
?FMCU (25)	Multiprocessor Communications Unit	Cannot be created with ?CREATE.
?FMTF (2)	Magnetic Tape File.	Explained in Chapter 6.
?FMTU (26)	Magnetic Tape Unit	Cannot be created with ?CREATE.
?FNCC (106)	FORTRAN Carriage Control Files	
?FOCC (111)	FORTRAN Carriage Control Files	
?FPLA (53)	Plotter	Cannot be created with ?CREATE.
?FPRG (101)	Program File	These should contain executable code.
?FQUE (41)	Queue Entry	

(continues)

Table 5-1. File Types Defined by AOS

Mnemonic (Octal Value)	Type	Comments
?FSDF (1)	System Data File	Used by the system.
?FSPR (40)	Spoolable Peripheral Directory	See Appendix E (PARU.LS).
?FSTF (103)	Symbol Table File	Used mostly by the system.
?FSYN (74)	Synchronous Communications Line	Cannot be created with ?CREATE.
?FTPA (52)	Paper Tape Punch	Cannot be created with ?CREATE.
?FTRA (44)	Paper Tape Reader	Cannot be created with ?CREATE.
?FTXT (104)	Text File	These should contain ASCII text.
?FUDF (100)	User Data File	Most user files are of this type.
?FUPF (102)	User Profile File	Used by PREDITOR.
?FWRD (113)	AZ-TEXT File	Created by AZ-TEXT.

(concluded)

Note that ?FUDF is the file type usually employed for source or object file storage; e.g., files with filenames ending in .SR and .OB.

The total size of a directory may not exceed 1 megabyte of information. If a directory is enlarged beyond this size, the system returns error code ERNRD. This limit affects only the number of entries that can be placed within a directory, and has no effect on the actual size of the files defined in the directory. Because the entries used to describe files are variable in length, it is not possible to predict exactly how many files can fit within a directory.

Accessing the Directory Structure

At all times, a process has a working directory as a reference point in the directory hierarchy. Either the system root directory or the working directory may be used as a starting point when a file is accessed. The process may also specify a list of other directories which will be searched when the system is given the name of an entry, without a prefix, that does not reside in the working directory. This list is called a *search list*.

If a desired entry is contained within the working directory, then no directory need be specified when the entry is to be accessed. If the desired entry is contained outside the working directory, then the location of the entry must be specified by either an explicit pathname or the presence of the desired entry's directory in the search list.

The search list can contain up to eight directories. The search function seeks an entry by first examining the working directory, and then going on to examine each subsequent directory in the order of appearance in the search list. If the desired entry is not found after looking through the directories in the search list, the message ERFDE is returned. The search terminates and an entry is returned when the first appearance of that entry is located (even if the entry appears in subsequent directories in the search list).

The names of the directories placed in the search list are resolved using the working directory at the time the ?SLIST is issued. Changing the working directory after setting the search list has no effect on the directories in the search list. The old search list is not used in setting up the new search list. If two pathnames resolve to the same directory, that directory will appear in the search list only once. The search list may contain directories to which you do not have execute access; the system will simply ignore these directories.

A *pathname* is a designation of the location of a desired directory entry. A pathname consists of an optional prefix, either by itself or with a list of one or more optional directory names followed by the filename. (A filename alone is a valid pathname.) To construct a pathname which points to another directory, you may use either a prefix alone or a prefix followed by one or more directory names. To designate a non-directory type entry, the prefix must be followed by at least one filename.

If a pathname has no prefix and the file cannot be found in the working directory, then the system will search the directories listed in the search list to find the requested entry. Consider the tree structure in Figure 5-2.

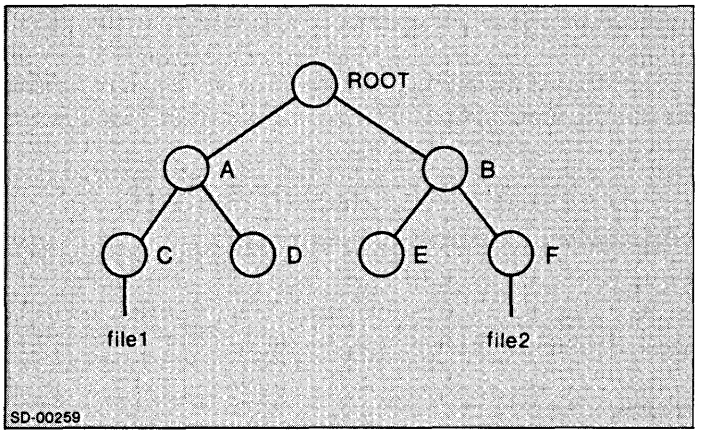


Figure 5-2. Tree Structure Example

If the process's working directory is C and the process needs to access file1, then it need supply only that file's name. If the working directory is other than C, then a pathname must be supplied or C must be named in the search list. Suppose that the working directory is F and the desired file is file1. A complete pathname to reach the desired file consists of a series of names starting at the root and containing A, C, and file1.

Pathname Syntax

Each pathname can consist of a prefix alone or has the following structure:

[prefix]...filename[:filename [:filename]...] terminator

That is, a pathname consists of either a prefix alone or an optional prefix followed by a filename, followed by optional additional filenames separated by colons indicating downward movement in the tree. Each pathname must be terminated by either a null or a new line character (octal 12). The system imposes a limit of 255 bytes on the length of pathnames; system utilities establish a maximum pathname length of 128 characters.

A prefix is one of four single-character symbols:

Prefix	Meaning
:	Start at the system root directory.
=	Start at the working directory.
↑	Move up to the parent directory. More than one ↑ can be used in a pathname.
@	Start at the peripheral directory.

The peripheral directory (inferior to the ROOT) contains the names of system devices and of a special class of files called *generic files*. Chapters 6 and 9 describe the peripheral directory at greater length.

If a pathname has no prefix and the file cannot be found in the working directory, then the system scans directories in the search list to locate the file. Note that the = prefix simply prevents the search list scan.

More than one up-arrow can be used in a prefix to construct a pathname which ascends through more than one father directory. Figure 5-3 depicts a possible directory structure and search list, and then gives a series of example pathnames. User data files bear lowercase names in this illustration.

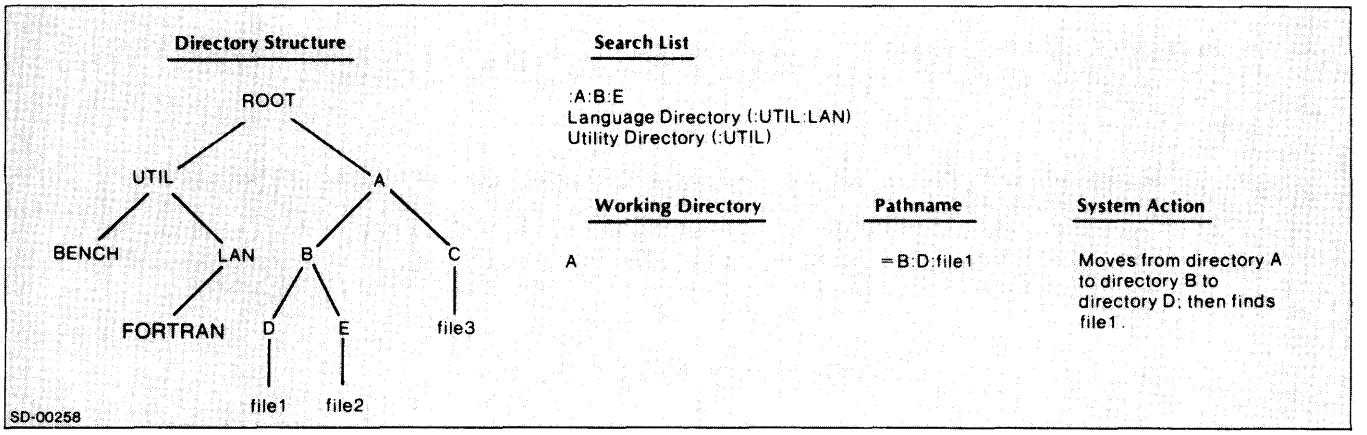


Figure 5-3. Pathname Examples

A complete pathname from the root can be obtained for any pathname by a call to ?GNAME. Assuming that D is the working directory in the tree structure shown in Figure 5-3, a call to ?GNAME with the incomplete pathnames shown would yield these complete pathnames:

Input	Output
file1	:A:B:D:file1
↑E	:A:B:E
=	:A:B:D
↑	:A:B

Link Entries

An entry used solely for directory access is the *link* (type ?FLNK). Whenever a filename corresponding to a link entry appears in a pathname, that filename is replaced in the pathname by the contents of the link entry (except in two special cases, described below). If the entry's contents start with a prefix, pathname resolution is restarted at the directory indicated by the prefix. If the entry's contents do not start with a prefix, then pathname resolution continues at the point in the directory hierarchy where the link entry was encountered. The maximum length of a link is 256 characters.

One possible use of the link mechanism is as an abbreviation for a longer pathname. Thus G might be created as a link equivalent to the pathname from ROOT to A to B in Figure 5-3. A link can also be employed by several processes as a convenient way to use a single file without each having to use up space for a copy of the file in its own directory space. That is, each process could use a pathname, such as :A:B:C:filename, by equating it to LINK1 and using LINK1 in each reference. An alternate way of referring to filename without the link mechanism would be to place that file's parent directory in the search list (described previously). However, this could be done only if no other directory named in the search list contained a file with the same name as filename.

A link is created by system call ?CREATE and is deleted by system call ?DELETE. Creation or deletion of a link in no way affects the file to which it refers. The contents of any link entry (i.e., the pathname for which it is an abbreviation) can be found by issuing system call ?GLINK. If a link is the last (or only) filename in a pathname input to ?CREATE or ?DELETE, the link will not be resolved; this permits link creation and deletion. Thus if the pathname

A:B:C

were input to ?DELETE and B was a link equivalent to D:E, then the effective pathname A:D:E:C would be input to ?DELETE and C would be deleted. If, however, the pathname

A:B

were input to ?DELETE, the link B itself would be deleted (directories D and E would be unaffected).

NOTE: The system does not validate the contents of a link entry until the entry is used in a pathname resolution. Thus, it is possible to create a link entry with illegal filename characters. The system will return an error, however, when you try to use such a link in a pathname.

Controlling Access to Files

The operating system defines five types of access which you, as a system user, may have for each file:

- execute access
- read access
- append access
- write access
- owner access

For nondirectory files, these access types have the following meanings. Execute access means that you may execute the file. Read access permits you to examine the data in a file, and write access permits you to modify the data in the file. Append access has no meaning for nondirectory files. Owner access permits you to change the file's Access Control List (see below).

For directory files, execute access means that you may use the directory in a pathname. Without this type of access, none of the other types are useful because you must use a directory in a pathname in order to enter or use it in any way. Read access allows you to list a directory's entries and the status of the files these entries point to. Append access permits you to insert entries in a directory. Write access permits you to create and delete entries, and modify the corresponding Access Control List of each. Finally, owner access permits you to initialize a logical disk (LD), described in the next section. (Access rights for logical disks are the same as for directories.) Owner access also allows you to modify a directory's Access Control List (described below) and to delete and rename its file entries. Again, none of the above access types are useful without execute access.

Access Control List

The system maintains an Access Control List (ACL) for each directory entry except links. The ACL is an ordered list of names of users who may access the entry, and the type of access to which they are entitled. By *ordered list,* we mean that the system checks the first name in the list, then the second, etc., stopping as soon as it finds a match. Usernames may be specified literally, e.g., ADAM for processes with username ADAM. Alternatively, usernames may be specified generically by means of a template. Three template characters are provided for this purpose: dash (-), plus sign (+), and asterisk (*).

The dash matches any character string not containing a period. The plus sign matches any character string. No username may contain two successive - or + characters. The asterisk matches any single character except a period. Each username can consist of any combination of up to 15 valid filename characters or asterisks; + and - template characters are not included in this 15-character maximum.

Since each ACL is an ordered list, you must avoid placing general templates where they would prevent more specific templates or names from being scanned. For example, the element + R at the beginning of an ACL would give all users read access only, regardless what the rest of the list specified. For this reason the system will truncate any ACL after an entry with a name consisting solely of a + template. Moreover, whenever it is possible, the system will resolve ACLs to reduce their size and render them more efficient. Such resolutions, however, will never change the substance of ACL specifications.

Given the ACL in Figure 5-4, user ED.FORTRAN would be given all access privileges. BOB.FORTRAN would have read and execute access, as would C.FORTRAN and C.BASIC. C.A.BASIC, however, would be allowed only execute access. All users not mentioned in the ACL, such as CA.BASIC, are permitted no access and cannot even determine if the entry exists. If an entry does not have an ACL, then only superusers can access that entry.

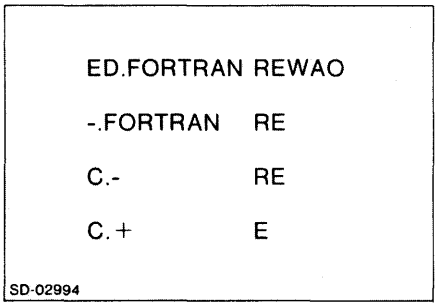


Figure 5-4. Sample Access Control List

Exercise caution when changing the ACLs of your files because it is possible to shut yourself out of your entries by removing yourself from the ACLs of your files and directories.

There are several ways to set an ACL for a file or a directory. One way is to use the CLI command ACL. You can also define a file's ACL from your source code via the ?CREATE, ?SACL, or ?DACL system calls. ?CREATE allows you to define the ACL and other specifications for the new file or directory. ?SACL allows you to set an ACL for a file or directory.

Depending on your input parameters, ?DACL sets, clears, or examines default ACL mode for one or more processes with specific usernames. Default ACL mode is process specific rather than file specific. For example, a process can issue ?DACL to turn on default ACL mode and define a specific ACL for all files it will later establish with ?CREATE. A default ACL defined with ?DACL exists until the ?DACL caller terminates, or until it redefines that default with another ?DACL call.

Protecting Disk Files

Users may want to protect certain files from being deleted, independently of the protection afforded by ACLs. This added protection is particularly useful for files whose ACLs permit all access to a single user. A user would receive all access privileges, for example, to a text file he created using a text editor utility. However, any user with owner access to a directory can delete file entries in that directory *regardless of his access* to the files themselves. Thus, no user can protect himself from inadvertently deleting his own files simply by restricting his access privileges to them.

The system does provide protection for directories currently in use, however. You cannot delete a directory that is either in use as a working directory or referenced in a search list. If you attempt this, the system returns the error code ERDIU.

The system provides an added means of protecting all disk files: the *permanent file attribute.* This attribute prevents a file from being deleted no matter what its ACL entries or that of its directory specify. By default, no file has the permanent attribute. Any user can make a file permanent by naming that file in system call ?SATR. Likewise, ?SATR can remove this attribute from any file. System call ?FSTAT returns information indicating whether or not a target file has the permanent attribute.

Note that the permanent file attribute protects the file only from specific deletion. That is, the parent directory can be deleted even though it contains a permanent file. To be completely safe, you also should make the parent directory permanent.

Forming File Space from Physical Units

The operating system supports a variety of types of disk units, and it permits these units to be intermixed within any system. Before any physical disk can be used in the system for file I/O, it must be processed by the disk formatter utility, DFMTR. This utility, described at length in the *AOS Operator's Guide,* performs a variety of functions. It may optionally analyze each surface of a disk to determine if each word in each block can be written into and read back without error. A block allocation/deallocation table is also built; bad blocks are flagged as being in use, so that they will never be used by the system.

DFMTR also associates one or more individual physical disk units into logical groupings called *logical disks (LDs).* Each LD is a set of one or more physical units which you want to consider as a single, logical unit. Each LD has a single block allocation/deallocation table built to control the use of its storage. Each file will be completely contained within a single LD.

For example, you might define three disks to comprise LD1, one disk to comprise LD2 and some other combination of disks for LD3.

Each LD is also a complete collection of disk space containing a directory tree structure. In fact, each LD has a single directory called a *local root.* It is the root which acts as the foundation for constructing a directory structure such as the one illustrated previously in Figure 5-2. Thus, each LD is a free-standing or local directory tree, because it consists either of a single local root or of a larger structure that has been built from the root by means of one or more ?CREATE commands.

When you bootstrap a system, you specify one LD as the *master LD.* The root of this LD becomes the system root and bears the name ":". Except for the master LD, each LD must be initialized (?INIT) before its files can be accessed. ?INIT introduces each free-standing tree to the system and grafts that tree's local root to some directory in the current system directory tree. The name of each other LD's local root is the name the system manager gave to the LD when the manager created it with the DFMTR utility.

Figure 5-5 shows how an LD named B, containing a tree structure with inferior directories B1 through B5 might be grafted to a master LD to produce a tree.

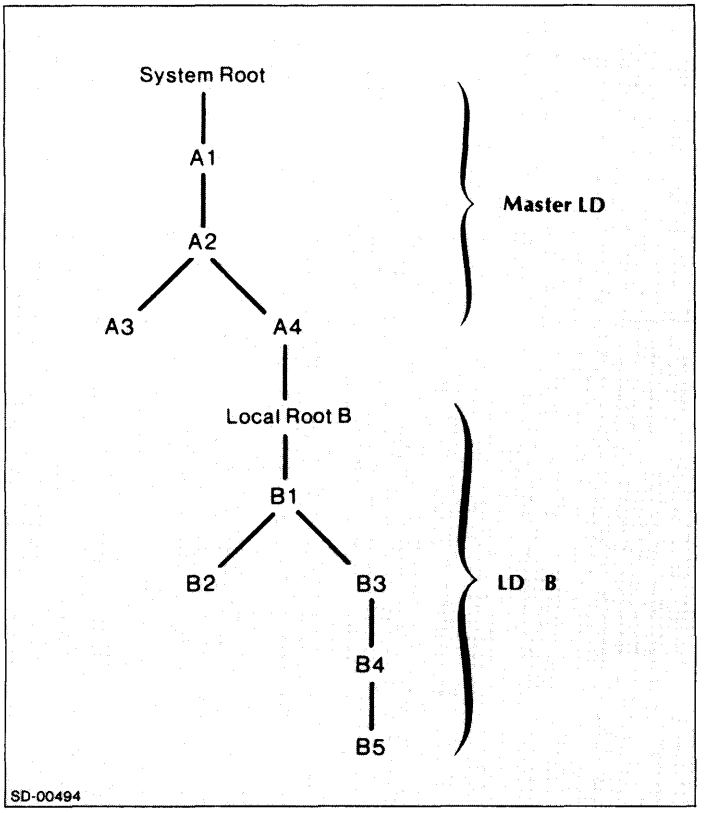


Figure 5-5. LDs in the System File Tree

Each local root is said to be at level zero. Excluding the local root in any LD, no local tree may have more than seven levels of directories. Note that the full tree can have more than seven levels. In Figure 5-5, the master LD has levels 0 through 3, and the LD B has levels 0 through 4; the whole tree has nine levels, 0 through 8.

Each LD also contains an Access Control List for its root. You specify this list when you create the LD. The system will permit only those users who have owner access to the LD root to initialize the LD.

Disk Space Control

Within a directory tree, you can designate certain directories as *control points* when you create them. The control point mechanism restricts file space to a defined limit. Control points encompass disk space within single LDs. At each control point, two variables are maintained: current space allocated (CS), and maximum space allocation (MS). CS is the current number of disk blocks used by all inferior files, including the control point directory (with the exception of those files in an inferior LD). MS is the maximum value for CS.

You should not set the maximum size of a control point to a value less than its current size. If you attempt to, the system responds with the error code ERCPD (you've exceeded Control Point Directory maximum).

There are two kinds of control points: LD roots, and other control point directories. For an LD root, CS is the total space used in the LD; MS is the total space available in a freshly-formatted LD. A control point directory is functionally equivalent to a normal directory, except that a local CS and MS are maintained. When a Control Point Directory is created, CS is set to zero.

Whenever disk space is required for a file to grow, all the control points between the file and the local LD root are checked and updated. In Figure 5-6 both the LD root and CP1 are designated control points. If file 1 needs an additional n blocks, the system first adds n to the CS value of CP1 -- the control point closest to file 1. If CS+n is greater than the MS value for CP1, any attempt to allocate additional space for file 1 would fail (error code ERCPD).

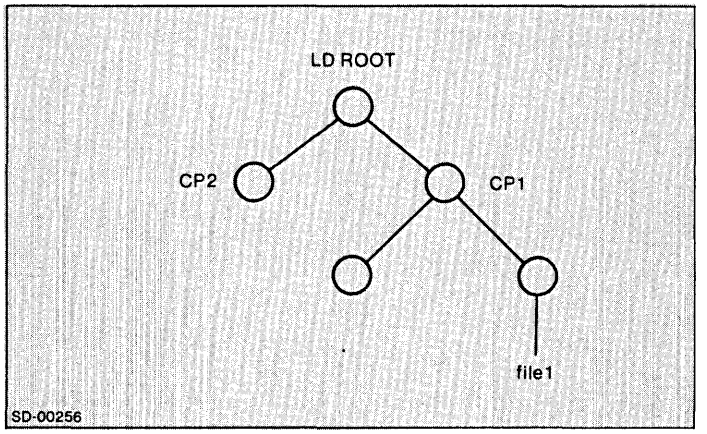


Figure 5-6. Control Points in a Tree

If CS+N is less than or equal to the MS value for CP1, the system checks the next control point -- in this case, the LD root. Then, the system adds n to the CS value for the root. If CS+n is less than or equal to the MS value at this level, the system allocates the additional disk blocks; otherwise, the allocation attempt fails.

No control checking of MS is done when a Control Point Directory is created. Oversubscription is permitted, provided that the total allocation never exceeds the amount available in any superior control point, up to and including the local LD root. Thus CP1 and CP2 might each have an MS of 70 blocks, even though the MS of the LD root equals only 100. However, the cumulative CS of CP1 and CP2 could never exceed 100.

System Call Summary

Before any LD not in the master LD can become available to the user, it must be initialized by system call ?INIT. Then the following general procedure may be performed for file creation and maintenance.

First, create an entry; this is done either with the ?CREATE call or -- as described in Chapter 6 -- with the ?OPEN call.

Having created an entry, you can then change its name with system call ?RENAME. System call ?DIR changes the working directory of a process. A complete pathname from the root can be obtained for any pathname by a call to ?GNAME.

System call ?DELETE deletes a directory entry -- for a disk file -- and releases the storage blocks that it was using. The status information contained in a directory entry can be determined with the ?FSTAT call. Two calls are defined to examine (?GACL) and modify (?SACL) a file's or unit's Access Control List. ?GTACP returns the access types that a user has to a specific file.

Two system calls are provided for managing the search list. System call ?GLIST describes the contents of the current search list. The contents of the search list are set or changed by system call ?SLIST.

An initialized LD remains initialized until it is released by system call ?RELEASE. An LD might be released so that its file space could become free-standing disk units. Alternatively, an LD might be released so that its component volumes could be dismounted and its drives used for other volumes.

The following list summarizes the names and functions of the disk file creation and maintenance calls:

?CGNAM	Get a complete pathname (from a channel number).
?CPMAX	Set maximum size for a Control Point Directory.
?CREATE	Create a directory entry.
?DACL	Set, clear, or examine a default ACL.
?DELETE	Delete a directory entry
?DIR	Change the working directory.
?FSTAT	Obtain an entry's status information.
?GACL	Read an Access Control List.
?GLINK	Read the contents of a link entry.
?GLIST	Read the search list.
?GNAME	Get a complete pathname (from a pathname).
?GRNAME	Get a resolved complete pathname.
?GTACP	Get access control privileges.
?INIT	Initialize an LD.
?RECREATE	Recreate a file.
?RELEASE	Release an LD.
?RENAME	Change a filename.
?SACL	Change the contents of an Access Control List.
?SATR	Set file attributes.
?SLIST	Set the contents of a search list.

?CGNAM

Get a complete pathname (from a channel number).

?CGNAM
exception return
normal return

Input/Output

Input:

AC0 byte pointer to buffer to receive the pathname.

AC1 channel number.

AC2 length of buffer in bytes.

Output:

AC0 unchanged.

AC1 unchanged.

AC2 actual length of pathname, excluding the null terminator.

Exceptional Condition Codes in AC0

ERMPR Designated buffer is too small.

FILE SYSTEM codes

CHANNEL RELATED codes

Description

This system call takes a channel number and returns a complete pathname which starts from the root directory (:). You are required to have read access to the parent directory of the target file, or any access to the file itself, before you can use the ?CGNAM call.

?CPMAX

Set maximum size for a Control Point Directory.

?CPMAX
exception return
normal return

Input/Output

Input:

AC0 contains a byte pointer to the pathname of the target Control Point Directory (CPD).

AC1 contains new sizes for CPD, high order bits.

AC2 contains new size for CPD, low order bits.

Output:

AC0 undetermined.

Exceptional Condition Codes In AC0

FILE SYSTEM Codes

Description

The ?CPMAX call changes the maximum block size for a CPD to the double precision integer value you specify in AC1 and AC2. This call does not change the size of the user directory, only that of the Control Point Directory. Note that you must have access to the CPD as a file in order to issue this call successfully. ?CPMAX is equivalent to specifying the second parameter in the CLI command SPACE.

?CREATE

Create a directory entry.

?CREATE *[packet address]*
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname.

AC2 address of parameter packet.

Output:

AC0 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This system call creates any of the entry types supported by the system. The ?OPEN call, discussed in Chapter 6, can also create (as well as open) files. However, since the ?CREATE call offers a greater measure of control in specifying parameters such as the file element size, you will probably want to issue this call to create files.

If the last filename in a pathname is a link, then the link will not be resolved. In this case, the link destination need not exist, and the link may even contain illegal filename characters.

If you do not have write access to the peripheral directory and attempt to create a file there, the system will give you a normal return if the file already exists and an exception return if the file does not exist. In neither case will the system create the file.

Depending upon the type of directory entry to be created, one of three kinds of packets will be passed with this call: IPC type, directory type, and all other types. Their structures are shown in Figures 5-7 through 5-9 and described in Tables 5-1 through 5-4. Default values are obtained either by selecting the parenthesized values in the "Default" column, or by selecting a value outside the specified permissible range.

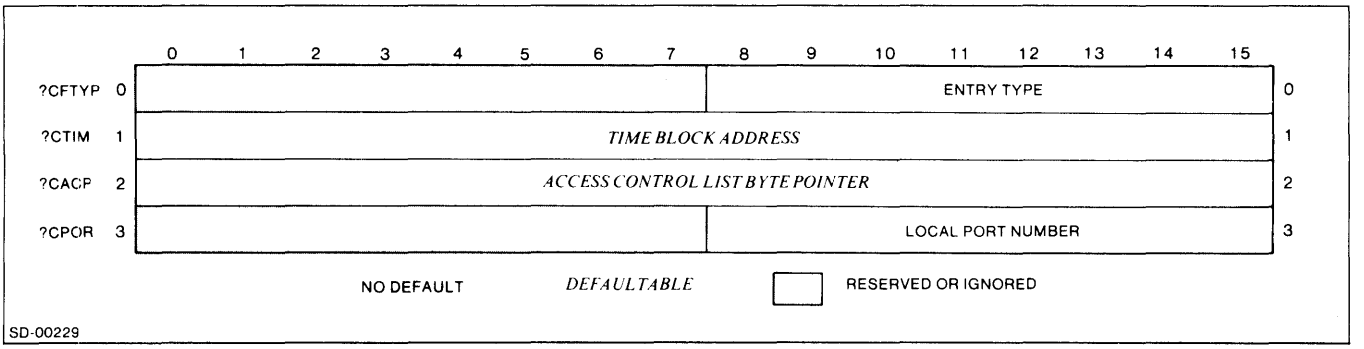


Figure 5-7. IPC Parameter Packet for ?CREATE

Table 5-2. IPC Parameter Packet for ?CREATE

Offset	Contents	Default
?CFTYP	Entry type. Available types are: ?FIPC and ?FSPR (see PARU.SR).	n/a
?CTIM	Address of Time Block.	(-1) All times will be set to the current time.
?CACP	Byte pointer to Access Control List. (See ?SACL for Access Control List format.)	(-1) ACL containing caller's username with all access privileges is built, or default ACL (if enabled).
?CPOR	Local port number, from 1-127.	n/a

NOTE: You may create an IPC entry only in the process's initial working directory (see offset ?PDIR in the ?PROC parameter packet). When any process is terminated, the system deletes all of that process's IPC entries.

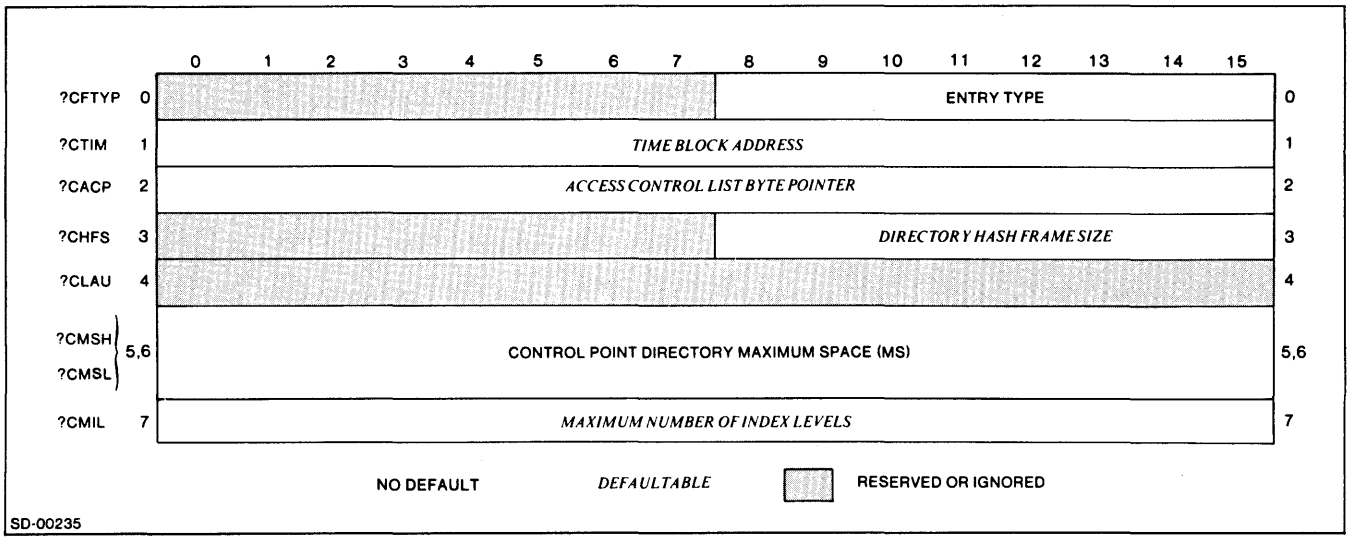


Figure 5-8. Directory Parameter Packet for ?CREATE

Table 5-3. Directory Parameter Packet for ?CREATE

Offset	Contents	Default	Range
?CFTYP	Entry type. Available types are ?FDIR and ?FCPD.	n/a	n/a
?CTIM	Address of Time Block.	(-1) All times will be set to the current time.	n/a
?CACP	Byte pointer to Access Control List.	(-1) ACL containing caller's user name with all access privileges is built, or default ACL (if enabled).	n/a
?CHFS	Directory hash frame size. (Typical applications will set this value to 0 and allow the system to default it.)	(0) System default has frame size.	0-255
?CLAU	Reserved.	n/a	n/a
?CMSH ?CMSL	MS for a control point directory (?FCPD only).	n/a	n/a
?CMIL	Maximum number of index levels, from zero to three inclusive. Specifying zero builds a contiguous file; i.e., one allocated as a series of disk blocks (ignored for links).	(-1) System default maximum number of index levels.	-1 to 3

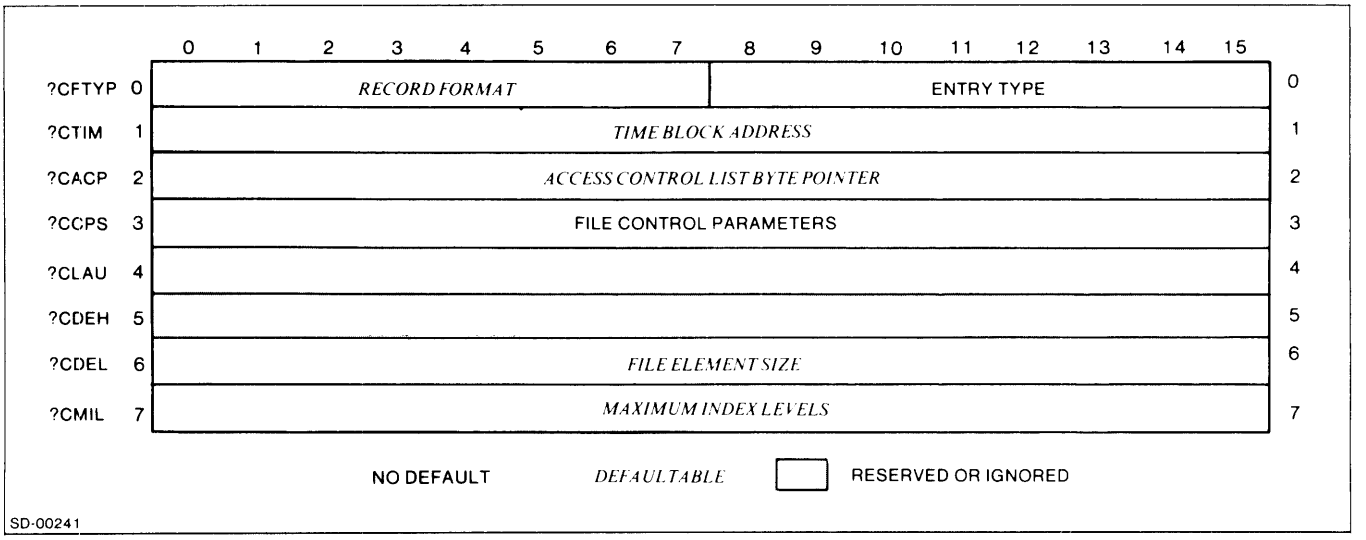


Figure 5-9. Remaining Types Parameter Packet for ?CREATE

Table 5-4. Remaining Types Parameter Packet for ?CREATE

Offset	Contents	Default	Range
?CFTYP	Entry Type and Record Format:		
	record format (left byte)	(0), The record format must be specified at ?OPEN or ?READ/ ?WRITE time.	n/a
	?ORDY - dynamic ?ORDS - data sensitive ?ORFX - fixed length ?ORVR - variable length		
	entry type (right byte) See Table 5-1.	n/a	n/a
	Any type other than IPC types or directory types; e.g., ?FLNK for a link.		
?CTIM	Address of Time Block.	(-1) All times will be set to the current time.	n/a
?CACP	Byte pointer to Access Control List or resolution pathname for a link.	(-1) ACL containing caller's username with all access privileges is built, or default ACL (if enabled).	n/a
?CCPS	File Control Parameters; equals record size for fixed length records; otherwise disregarded.	n/a	n/a
?CLAU	Reserved word.	n/a	n/a
?CDEH	Reserved word.	n/a	n/a
?CDEL	File Element Size (ignored for links).	(-1)1.	-1 to 2**16-2
?CMIL	Maximum number of index levels, from zero to three, inclusive. Specifying zero builds a contiguous file, i.e., one allocated as a series of disk blocks (ignored for links).	(-1) System defualt maximum number of index levels.	-1 to 3

?CACP is a byte pointer to an ACL in the same format as required by ?SACL. If ?CACP is 0, no ACL will be built unless default ACLs are enabled. If ?CACP is -1, an ACL consisting of the caller's username with all access privileges will be built for the entry. For link entries, ?CACP specifies a link resolution pathname; no link resolution pathname (including the null terminator) can exceed 256 bytes.

If an address of a Time Block is given, i.e., if the contents of ?CTIM is other than -1, then a Time Block with the structure shown in Figure 5-10 must be provided.

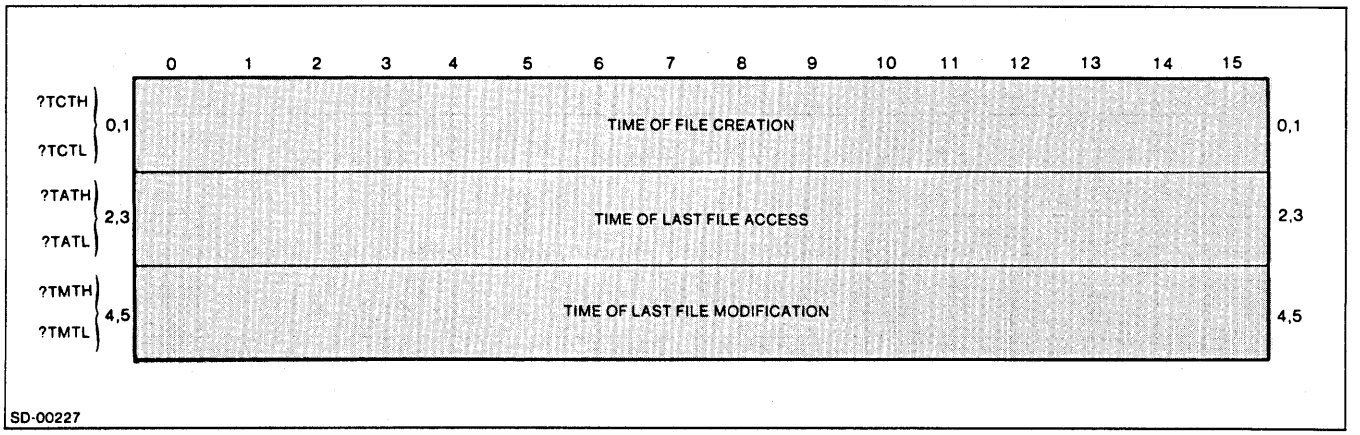


Figure 5-10. Time Block for ?CREATE

Times given in the Time Block are double precision values defined as follows: the high order portion lists the number of days since December 31, 1967. The low order portion lists the time on that day, where the time is expressed as half the number of seconds since midnight.

Offset ?CDEL specifies the number of blocks in one file element.

Record format types (dynamic, data sensitive, etc.) are described at length in Chapter 6.

The ?CREATE call requires the caller to have execute and write or append access to the directory.

?DACL

Set, clear, or examine a default Access Control List.

?DACL
exception return
normal return

Input/Output

Input:

AC0 Flag Word

- -1 turn on user default ACLs.
- 0 return state of user default ACL in AC0 (-1=on, 1=off).
- 1 turn off user default ACLs.

AC1 byte pointer to 128. Word buffer in user space.

If AC0 contains -1, buffer contains new ACL.

If AC0 contains 0, the system returns the current ACL.

If AC0 contains 1, this AC is ignored.

Output:

AC0 unchanged, or current state, depending on AC0 contents on input.

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERACL Illegal ACL.

ERMPR System call parameter address error.

Description

This system call permits the caller to clear default ACLs, set new ones for processes, or examine current default ACLs. The format of the default ACL is:

username<0><access type> *[username1<0><access type> ...]* <0>

For a complete description of the access types, refer to the description of the system call ?SACL in Chapter 5.

?DELETE

Delete a file entry.

?DELETE
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname of file to be deleted.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

ERDID Attempt to delete a directory which contains entries of one or more inferior directories.

ERDIU Attempt to delete a working directory, or any directory named in a search list.

ERPRM Attempt to delete a permanent file.

FILE SYSTEM codes

Description

This system call deletes a directory entry; if it is a disk file, its file space is made available for other use. This call may delete links. If the last filename in a pathname is a link, the link will not be resolved but will be deleted.

If the input pathname has an @ prefix *no deletion occurs* and control takes the normal return if the file exists, or it takes the exception return if the file does not exist.

If a file is open at the time its entry is deleted, the file itself is not deleted. Instead, the system deletes the filename and waits until all users of the file close the file; at this time the file is deleted and its disk space is made available for other use. In either case, control returns immediately to the normal return in the task making the ?DELETE call.

This call requires the caller to have write access to the entry's parent directory or owner access to the file being deleted. This call never scans the search list. In effect, ?DELETE works as though each input pathname has an = prefix.

?DIR

Change the working directory.

?DIR
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname or zero.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

The ?DIR call changes the working directory of the process issuing the call. If the input content of AC0 is zero, then the working directory is changed to the initial working directory specified when the process was created.

The caller must have read and execute access to a directory in order to make it his working directory.

?FSTAT

Get the status of a file.

?FSTAT *[file status address]*
exception return
normal return

Input/Output

Input:

AC0	byte pointer to a pathname or a channel number.
AC1	flags word:
	bit 0=0, AC0 contains a byte pointer.
	bit 0=1, AC0 contains a channel number.
	bit 1=0, resolve links.
	bit 1=1, do not resolve links.
AC2	address to receive file status information.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This system call returns a collection of file status parameters to an area (?SLTH words long) designated by the caller to receive the information. If you provide a pathname ending in a link and you set bit one of AC1, the link will not be resolved, and you will get the status of the link file. If you do not set bit one of AC1, the link will be resolved and you will receive the status of the file to which it points. Links found within a pathname (not ending entry) are always resolved. File status information will be returned in one of four ways, depending upon the type of the file entry: unit, IPC, directory, or other. Figure 5-11 shows the structure of this packet for the various types.

Offsets ?STYP through ?SMIL contain the same information as offsets ?CTYP through ?CMIL in the ?CREATE parameter packet. Only the entry type and time of creation are meaningful for links. For all types except Link and IPC, this permits an entry to be created which has the same characteristics as the file entry whose status was just examined.

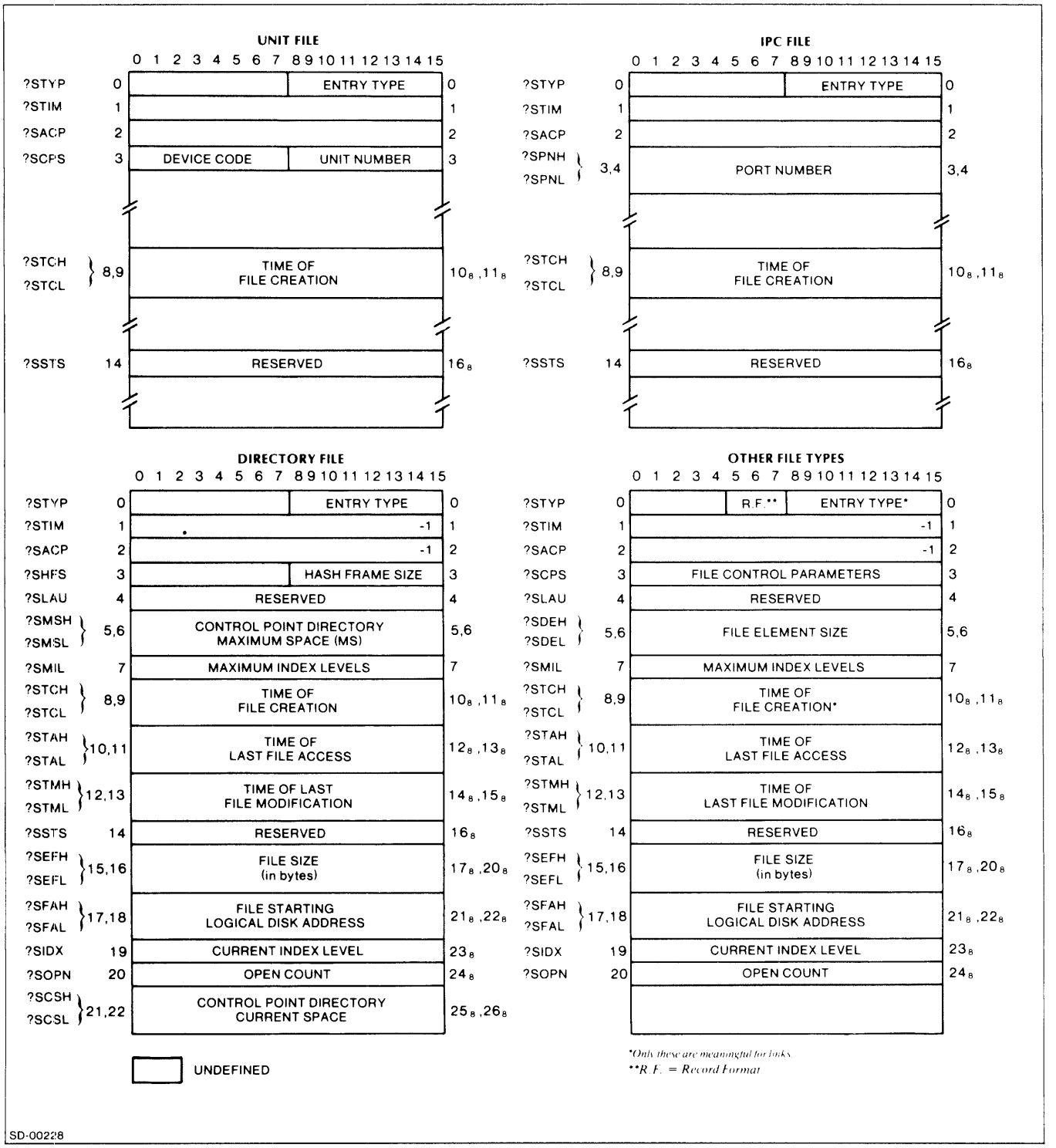


Figure 5-11. File Status Packet Structure

Offset ?SCPS will contain information appropriate to the entry type whose status is being read. For unit types, this word contains the device code (left byte) and unit number (right byte). For IPC types, this word is labeled ?SPNH; it and the following word, ?SPNL, contain the port number. For directories, this offset is labeled ?SHFS and contains the hash frame size. If the entry type is user data and fixed length records are used, then this word contains the record length. (Contents of this word are undefined for other record types.)

Offset ?SDEL contains a number describing the data element size, ranging from 1 through 2**16. For directories, these offsets contain the control point variable MS.

Word ?SMIL contains the maximum number of physical index levels permitted in the file's structure. This value was set when the file was created.

The next three word pairs, ?STCH through ?STML, describe the times of creation and last access of the entry, and the last modification of the file. These times are in the same format used by the ?CREATE parameter packet. The first word of each pair tells the number of days after December 31, 1967, that the event occurred. The second word of each pair indicates the time of day that the event occurred, expressed as half the number of elapsed seconds in the day. You may issue ?CDAY to convert the number-of-days word to the day, month, and year. You may issue ?CTOD to convert the time word to hours, minutes, and seconds. ?CDAY and ?CTOD are described in Chapter 9.

The file status word, ?SSTS, is shown in Figure 5-12. Its bits have the following meaning when set:

Bit	Meaning when set
?FSHB	File is shared.
?FMDB	File was modified while currently open.
?FPRM	File is permanent -- it can't be deleted.
?FDLE	File will be deleted on last close.
?FUDA	File has an associated UDA.
?FOEX	File is exclusively opened.
?FAOA	All users have owner access.
?FAWA	All users have write access.
?FAAA	All users have append access.
?FARA	All users have owner access.
?FAEA	All users have execute access.

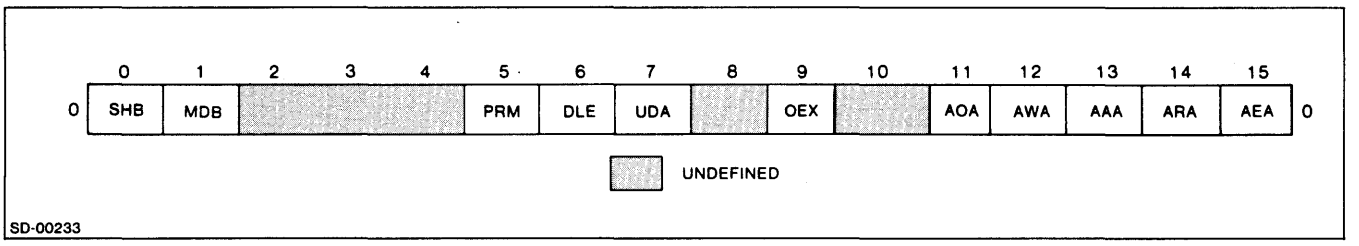


Figure 5-12. File Status Word

The system creates a unit entry for each magnetic tape, disk, or MCA unit in the system. The system uses these entries for managing the intialization of these units.

Offsets ?SEFH/?SEFL contain the size of the file in bytes. The more significant half is in ?SEFH, and the less significant half in ?SEFL. ?SFAH and ?SFAL contain the double-precision starting logical disk address of the file. These values range from 0 to the highest address available to the user.

The current index level count, ?SIDX, indicates the current level of indexing which was constructed for the file. This value will never exceed the value defined in ?SMIL.

Offset ?SOPN contains the open count. This value indicates how many different users currently have the file open.

Offsets ?SCSH and ?SCSL contain CS, the current size control point variable. This number is the amount of disk space currently allocated in this Control Point Directory, and is meaningful only if the file is of type ?FCPD or ?FLDU.

Read access to the parent directory of the target file or any access to the target file, is required with ?FSTAT.

?GACL

Get an entry's Access Control List.

?GACL
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname of directory entry.

AC1 byte pointer to user buffer to receive the ACL.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This system call reads into the caller's buffer the target entry's Access Control List, in the same format as ?SACL. The buffer must be 256 bytes in length. The caller must have read access to the entry's parent directory or owner access to the file itself.

?GLINK

Get the contents of a link entry.

?GLINK
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname of link.

AC1 byte pointer to buffer to receive the contents of the link.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This call copies the contents of a link entry into a user-supplied buffer. If the pathname does not point to a link entry, an error is returned. The buffer must be 256 bytes in length, and the caller must have read access to the parent directory containing the link entry.

?GLIST

Get the search list.

?GLIST
exception return
normal return

Input/Output

Input:

AC1 byte pointer to buffer to receive the search list.

AC2 length of buffer in bytes.

Output:

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This call writes a copy of the search list into a user-specified buffer. The length of the buffer (in bytes) must be placed in AC2 prior to issuing the call. Only the first 511 bytes of the search list are returned. The search list image is in the format:

pathname<0> ... pathname <0> <0>

?GNAME

Get a complete pathname.

?GNAME
exception return
normal return

Input/Output

Input:

AC0 byte pointer to input pathname.

AC1 byte pointer to buffer to receive the pathname.

AC2 length of buffer in bytes.

Output:

AC0 unchanged.

AC1 unchanged.

AC2 actual length of returned pathname, excluding the null terminator.

Exceptional Condition Codes in AC0

ERMPR Designated buffer is too small.

FILE SYSTEM codes

Description

This system call takes an input pathname and returns a complete pathname which starts from the root directory. Thus, for example, the pathname = as input yields the name of the working directory. Read access to the parent directory of the target file, or any access to the file itself is required. However, if pathname is only a prefix (e.g., =), then the system makes no access checks.

?GRNAME

Get a resolved complete pathname.

?GRNAME
exception return
normal return

Input/Output

Input:

AC0 byte pointer to input pathname.

AC1 byte pointer to buffer to receive the pathname.

AC2 length of buffer in bytes; bit 0 is ignored.

Output:

AC0 unchanged.

AC1 unchanged.

AC2 bits 1-15, actual length of returned pathname, excluding the null terminator.

bit 0=0, if file was @INPUT, @OUTPUT, @LIST, @DATA.

bit 0=1, if otherwise.

Exceptional Condition Codes in AC0

ERMPR Designated buffer is too small.

FILE SYSTEM codes

Description

This system call takes an input pathname and returns a complete pathname starting from the root directory. For example, the pathname = as input yields the name of the working directory. The caller must have read access to the parent directory of the target file, or any access to the file itself. However, if the pathname is only a prefix (e.g. =), then the system does not make access checks. If the input pathname is one of the generic filenames @INPUT, @OUTPUT, @LIST, or @DATA, these filenames are resolved, where possible, according to generic files specified to the process. Note that the generic file @NULL returns ":PER:NULL" with bit 0 set to 1 in AC2.

?GTACP

Get access control privileges.

?GTACP
exception return
normal return

Input/Output

Input:

AC0 byte pointer to file pathname.

AC1 byte pointer to username or -1 if the caller wishes to get its own access privileges.

Output:

AC0 access control privileges word. The following bits are set to indicate these privileges:

?FAOB owner access

?FAWB write access

?FAAB append access

?FARB read access

?FAEB execute access

AC1 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This call returns the privileges that a given user has for a specific file as bit settings in AC0. You must have superuser privilege to use this call if AC1 contains a byte pointer to a username (even if it's your own username). Superuser privilege is not required, however, if AC1 contains -1. This is an alternate way of obtaining the access privileges associated with your own username. Similarly, if you lack superuser privilege and attempt to use ?GTACP to determine the access privileges associated with a username, the system returns error code ERPRV -- caller not privileged for this action. The user parameter file, PARU, defines the following masks to assist you in examining the privileges word:

Mask	Mask Definition	Meaning
?FACO	1B(?FAOB)	owner access
?FACW	1B(?FAWB)	write access
?FACA	1B(?FAAB)	append access
?FACR	1B(?FARB)	read access
?FACE	1B(?FAEB)	execute access

The system does not examine the search list to resolve the pathname input to this call. If the call you specify in the pathname cannot be found, the system returns error code ERNNF.

?INIT

Initialize an LD.

?INIT *[parameter packet address]*
exception return
normal return

Input/Output

Input:

AC0	byte pointer to an area, 32 bytes long, which will receive the name of the LD; if AC0 contains zero, no area is specified.
AC1	-1: graft LD onto system's root directory; 0: graft LD onto the working directory; otherwise, byte pointer to pathname of parent directory where the LD's local directory will be grafted.
AC2	address of parameter packet.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

INITIALIZATION and RELEASE codes

Description

This call initializes a logical disk (LD). For this, the caller must have owner access to the LD's root directory, and write or append access to the directory into which it is being incorporated. The caller must also have execute access to each of the disk units comprising the LD. An initialized LD remains initialized until it is released via the ?RELEASE system call.

The ?INIT parameter packet consists of $n+1$ words, where n is the number of disk units comprising the LD which is to be initialized. The strings pointed to by offsets 1 through n in the ?INIT parameter packet must be terminated by a null.

Offset	Contents
0	Number of physical units which will be initialized.
1	Byte pointer to a string naming the first disk unit comprising the LD which will be initialized.
n	Byte pointer to string naming the nth disk unit comprising the LD which will be initialized.

The *AOS System Manager's Guide* contains a list of valid disk names.

?RECREATE

Recreate a file.

?RECREATE
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname of file to be recreated.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

Use this call to optimize the deletion and subsequent recreation of a file, using the same parameters as existed in the original file. The ?RECREATE call destroys all the data in the file, except for the data in the UDA which may be associated with the file. The filestatus parameters for the file are left intact (i.e., filename, Access Control List, creation date, file type, etc.).

If the ?RECREATE call is made while the file is open, it will behave in much the same manner as a ?DELETE call. The file data will still be visible to the user that issued the ?OPEN call until that user issues a ?CLOSE call. The data will not be visible to any users that ?OPEN the file after the ?RECREATE call is issued.

?RELEASE

Release an initialized LD.

?RELEASE
exception return
normal return

Input/Output

Input:

AC0 byte pointer to LD's pathname.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

INITIALIZATION and RELEASE codes

Description

This call releases a previously initialized logical directory. An LD must be released before the drive(s) that it is mounted on can be reused. This call requires the caller to have write access to the parent directory of the LD.

?RENAME

Change a filename.

?RENAME
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname of old filename.

AC1 byte pointer to new filename.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

ERWAD Caller does not have proper access.

FILE SYSTEM codes

Description

This system call adds a new filename to a file, and then deletes the old filename. The caller must have write access to the directory containing the entry pointed to by the old filename, or owner access to the file itself.

If you use this call to rename a logical disk, the LD will retain the new name only until it is released and re-initialized. After that, it will have the original name with which it was formatted.

?SACL

Supply a new Access Control List.

?SACL
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname of directory entry.

AC1 byte pointer to new ACL, or -1 to delete the old ACL.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This call replaces a target directory entry's Access Control List with a list supplied by the caller. This call requires the caller to have write access to the entry's parent directory, or owner access to the file itself. The format of the ACL pointed to by AC1 is:

username <null><access type>*[username1 < null > < access type >...]*<null>

Type the ACL specifications on one line; if you use the carriage return or NEW LINE characters, the system will interpret those as part of the ACL specifications. As the sample format shows, you can supply more than one ACL list with this call. To give certain access types to all users, specify +<access type> where + is the username.

The access type is a byte specifying the type(s) of access permitted for that username. Access types are the following:

?FACA append access

?FACE execute access

?FACR read access

?FACW write access

?FACO owner access

For example, to permit both read and write access, supply ?FACR+?FACW as an access type.

The total length of an ACL must be less than 256 bytes. If the input byte pointer (AC1) is -1, the old ACL will be deleted and no new one will be supplied.

?SATR

Set the attributes.

?SATR
exception return
normal return

Input/Output

Input:

AC0	byte pointer to pathname.
AC1	bit 15 = 1, set the permanent attribute.
	bit 15 = 0, take away the permanent attribute.

Output:

AC0	unchanged.
AC1	unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This system call sets or resets the attributes. At present, only the permanent file attribute is defined. A file with this attribute cannot be deleted.

This call requires the caller to have write access to the entry's parent directory or owner access to the file.

Note that the permanent file attribute protects the file only from specific deletion. That is, the parent directory can be deleted even though it contains a permanent file. To be completely safe, you also should make the parent directory permanent.

?SLIST

Set the search list.

?SLIST
exception return
normal return

Input/Output

Input:

AC2 byte pointer to search list.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERSRE Search list contains too many pathnames.

FILE SYSTEM codes

Description

This call sets the search list. The search list specification can contain a minimum of zero and a * maximum of eight distinct pathnames. (Input pathnames that resolve to the same directory count as one distinct pathname.) Its format is:

[pathname *[terminator pathname...]* terminator]

where *terminator* is null or NEW LINE.

NOTE: Each pathname that you supply with this call is resolved immediately; i.e., it is replaced by a complete pathname from the root to the specified directory.

This has the following implications:

- Pathname prefixes are relative to the current working directory, not to the working directory at the time of future use.
- The ultimate search list after the specification pathnames have been resolved may contain more than 511 bytes. Therefore, you may not be able to get the entire search list with a ?GLIST call.

Examples

Following are two programs which illustrate how to ?CREATE a user data file and then how to ?RENAME it. (Chapter 4 shows how to ?CREATE an IPC entry.) FILCREATE, shown in Figure 5-13, creates a user data file named TEST.

```
 0001  FILCR    AOS ASSEMBLER REV 01.08          16:02:36 12/05/78
01
                                    .TITL   FILCREATE
03                                  .EXTN   ERROR
04
05                  ;THIS PROGRAM CREATES A DATA FILE NAMED "TEST" IN THE
06                  ;CURRENT DIRECTORY, AND INFORMS THE CLI WHEN IT'S DONE.
07                                  .NREL
08
09 00000'020451             START:  LDA     0,NAME          ;GET FILE NAME
10 00001'070432                     LEF     2,PKT           ;GET PKT.ADRS.
11                                  ?CREATE                 ;CREATE THE FILE
12 00004'002406                     JMP     @.ERROR
13 00005'024407                     LDA     1,MSG   ;GET BYTE PNTR.TO MSG.
14 00006'030405                     LDA     2,FLAG          ;GET GOOD RTN FLG.
15                                  ?RETURN
16 00011'002401                     JMP     @.ERROR
17
18 00012'000000$            .ERROR: ERROR
19 00013'100032             FLAG:   ?RFCF+26.  ;MSG. IN IS CLI FORMAT,26 CHARS.
20 00014'000032"            MSG:    .+1*2
21 00015'043111                     .TXT    "FILE NAMED TEST IS CREATED"
22        046105
23        020116
24        040515
25        042504
26        020124
27        042523
28        052040
29        044523
30        020103
31        051105
32        040524
33        042504
34        000000
35
36
37
38
39
40
41
42
43
44
```

Figure 5-13. FILCREATE (continues)

```
45
46                ;CREATE PACKET
47        000033'         PKT:    .LOC    PKT+?CFTYP      ;RECORD TYPE WILL BE
48 00033'000100                   0+?FUDF                 ;GIVEN AT I/O TIME.
49        000034'                 .LOC    PKT+?CTIM       ;CURRENT TIME FOR
50 00034'177777                   -1                      ;TIME BLOCK.
51        000035'                 .LOC    PKT+?CACP       ;BYTE POINTER TO
52 00035'000106"                  ACL*2                   ;ACL.
53        000036'                 .LOC    PKT+?CCPS       ;DISREGARD (TYPE
54 00036'000000                   0                       ;NOT DECIDED YET).
55        000037'                 .LOC    PKT+?CLAU       ;SAME LAU AS
56 00037'177777                   -1                      ;PARENT DIRECTORY
57        000040'                 .LOC    PKT+?CDEH       ;RESERVED WORD.
58 00040'000000                   0
59        000041'                 .LOC    PKT+?CDEL       ;FILE ELEMENT
60 00041'177777                   -1                      ;SIZE= -1.


 0002  FILCR
01        000042'                 .LOC    PKT+?CMIL       ;SYSTEM DEFAULT
02 00042'177777                   -1                      ;MAX.# INDEX LEVELS.
03
04 00043'050101           ACL:    .TXT "PAT<0><?FACA+?FACE+?FACR+?FACW+?FACO>+<0><
05        052000
06        017453
07        000003
08        000000
09
10
11                ;GIVE SELF ALL ACCESS RIGHTS, READ AND XEQ TO OTHERS.
12 00050'000122"          NAME:   .+1*2
13 00051'052105                   .TXT    "TEST"
14        051524
15        000000
16
17                                .END    START

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS

  0003 FILCR

ACL    000043'        1/52    2/04
ERROR  000001  XN     1/03    1/18
FLAG   000013'        1/14    1/19
MSG    000014'        1/13    1/20
NAME   000050'        1/09    2/12
PKT    000033'        1/10    1/47    1/49    1/51    1/53    1/55    1/57
                      1/59    2/01
START  000000'        1/09    2/17
.ERRO  000012'        1/12    1/16    1/18
?CREA  000406  MC     1/11
?RETU  004600  MC     1/15
?XCAL  000001         1/12    1/16
```

Figure 5-13. FILCREATE (concluded)

After it has done this successfully, FILCREATE simply returns to the CLI and issues the message FILE NAMED TEST IS CREATED to indicate that it is done. The Access Control List specified for FILCREATE gives PAT all access privileges, and gives all other users (+) read and execute access.

Figure 5-14 shows the companion program, RENM.

RENM renames the file TEST that we created to TEST.01; then RENM terminates itself and announces what it has done on the console.

```
 0001 RENM  AOS ASSEMBLER
                          .TITL   RENM
02                        .EXTN   ERRCR
03
04                ;THIS PROGRAM RENAMES A FILE NAMED "TEST" TO "TEST.01"
05                        .NREL
06 00000'020436 START:  LDA     0,OLDNM ;GET THE OLD NAME
07 00001'024430         LDA     1,NEWNM ;GET THE NEW NAME
08                      ?RENAME
09 00004'002406         JMP     @.ERROR
10 00005'030406         LDA     2,FLAG          ;GET THE GOOD RETURN FLAG
11 00006'024406         LDA     1,MSG
12                      ?RETURN                 ;RETURN TO THE CLI
13 00011'002401         JMP     @.ERROR
14 00012'000000$.ERROR: ERROR                   ;EXTERNAL ERROR ROUTINE.
15 00013'100030 FLAG:   ?RFCF+24.       ;MSG IS IN CLI FORMAT AND IS 15 CHARACTERS
16 00014'000032"MSG:    .+1*2
17 00015'043111         .TXT "FILE RENAMED TO TEST.01"
18        046105
19        020122
20        042516
21        040515
22        042504
23        020124
24        047440
25        052105
26        051524
27        027060
28        030400
29 00031'000064"NEWNM:  .+1*2
30 00032'052105         .TXT "TEST.01"
31        051524
32        027060
33        030400
34 00036'000076"OLDNM:  .+1*2
35 00037'052105         .TXT "TEST"
36        051524
37        000000
38
39                      .END START

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS

  0002 RENM

ERROR 000001  XN     1/02    1/14
FLAG  000013'        1/10    1/15
MSG   000014'        1/11    1/16
NEWNM 000031'        1/07    1/29
OLDNM 000036'        1/06    1/34
START 000000'        1/06    1/39
.ERRO 000012'        1/09    1/13    1/14
?RENA 000314  MC     1/08
?RETU 002415  MC     1/12
?XCAL 000001         1/09    1/13
```

Figure 5-14. RENM

End of Chapter

Chapter 6
File Input/Output

The previous chapter described disk file organization and use of the file directory for organizing user data. This chapter describes the storage and retrieval of file data.

During system initialization, the names of those devices which will be available to users are written into a system directory called the peripheral directory, :PER. The peripheral directory also contains generic filenames (described later in this chapter). Standard device names and generic filenames are not reserved exclusively for devices and generic files. In fact, whenever you want to make a program reference to a file or device listed in :PER, you must precede that name with the @ prefix. Thus, you may create a file named LPT in your own directory even though @LPT is the system's name for line printer.

Table 6-1 gives the names of all devices which you can find in AOS. For those devices controlled by the EXEC spooler, queuenames are also given.

The system provides *device independence;* that is, I/O devices are accessed as files. Thus, you may request to "read the next record," and, in the proper context, the system will understand this to mean "read the next card from the second card reader."

Record Formats

Data is read from and written to each of the standard devices and magnetic tape or disk files either in blocks (at the device dependent level) or in logical groupings called *records.* The basic I/O system supports four types of records: dynamic, fixed, data sensitive, and variable shown in Figure 6-1.

A *dynamic record* is one whose length (in 8-bit bytes) is specified when it is read or written. A *fixed length record* has a constant length. A *data sensitive record* is a series of characters terminated by a character defined as a delimiter. Several different characters can be defined as delimiters, and delimiter definitions can be changed. A *variable length record* has a fixed length header describing the number of bytes of data that it contains.

Record I/O, performed by the ?READ and ?WRITE system calls, can specify any of the four record types with any device or file.

There is a special-purpose fifth record type: *undefined record length.* You may use this type only to perform a ?READ on an unlabeled mag tape when you don't know the physical block size. The system will read the physical blocks off the tape if you choose this record type.

I/O Operation Sequences

In general, the following sequence of system calls is issued for any I/O transfer; open the file (?OPEN), read (?READ) or write (?WRITE) file data, and then close the file (?CLOSE). A common parameter packet is used with each of these calls, even though certain portions of it are not relevant to all calls.

Opening the file associates that file with a *channel number* from 0 to 100 octal; the system makes this assignment and maintains a set of channel numbers for each process. (The system assigns 100 octal when you specify @NULL as the file.) The channel number is then referred to when I/O is requested; the channel number acts as an abbreviation for the filename that was assigned when the file was opened. A process can use the system call ?CGNAM to obtain a file's complete pathname from its channel number. *

Table 6-1. Device Name/Queue Name List

Name	Device
@CRA,@CRA1	First and second card readers.
@CON0,@CON1,...@CONn	First and succeeding console display/keyboards or asynchronous communications lines.
@DPn0 through @DPn17")	Moving head disk units 0 through 7 on the first controller and 10_8 through 17_8 on the second controller. n is a single alphabetic character indicating the disk unit type. These types are described in the *AOS System Manager's Guide*.
@DKB0 through @DKB11	Fixed-head disk units 0 and 1 on first controller, and 10 and 11 on second controller.
@LPA,@LPA1	First and second nondata channel line printers.
@LPB,@LPB1	First and second data channel line printers.
@LPC,@LPC1	First and second nondata channel line printers of type LP2.
@LPD,@LPD1	First and second data channel line printers of type LP2.
@LPT,@LPT1	First and second line printer queue names.
@MTn0 through @MTn17	Magnetic tape units 0 through 7 on the first controller, and 10_8 through 17_8 on the second controller (unitnames).
@LMT	Labeled magnetic tape.
@MCA and @MCA1	First and second multiprocessor communications adapter controllers (unitnames).
@PLA, @PLA1	First and second digital plotters.
@PLT,@PLT1	First and second digital plotter queue names.
@TPA, @TPA1	First and second high-speed paper tape punches.
@PTP,@PTP1	First and second high-speed paper tape queue names.
@TRA,@TRA1	First and second high-speed paper tape readers.

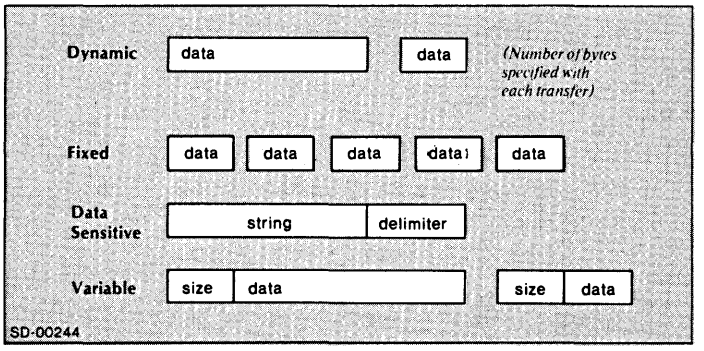


Figure 6-1. Record Formats

Closing the file disassociates the file from the channel number, and I/O cannot be performed on that file until it is specified in another open call. (There are other open calls in the operating system; e.g.; ?SOPEN; each of these has a corresponding close calls; e.g. ?SCLOSE. These calls are not used for record I/O and are discussed elsewhere.)

It is possible to access entire disk units as free-standing files, or to apply the system file hierarchy of a tree structure to these units. Accessing entire units as free-standing files is performed by opening the units without first initializing them. If a unit is first initialized (?INIT), then the system sees the unit as a logical structure (or as a part of such a structure) whose files can be opened separately. These logical structures, called directories, are described in Chapter 5. After all files on an initialized unit have been closed, the unit must be released (?RELEASE) before the unit can be opened as a free-standing file.

Each system has a logical disk (LD) which is the master LD, and its root is the system root (see Chapter 5). Every other LD is a free-standing tree in its own right. That is, it has a local root which can be grafted to any user directory in the master LD or in any directory which has been grafted to the master. System call ?INIT initializes an LD and grafts it onto a directory.

System call ?RELEASE is the inverse of system call ?INIT, and releases a previously initialized LD.

The system maintains a file pointer for each user that has opened a file. There is a separate file pointer for each open channel; the file pointer keeps track of the character position within the file where the next read or write will take place. The pointer can be set either to the start of the file or to the end of the file when the file is first ?OPENed. Normally, the file pointer moves forward through the file as each record or byte string is read or written. However, the ?READ and ?WRITE commands permit the file pointer to be moved backward from its current position. Alternatively, ?SPOS also moves the file pointer. System call ?GPOS returns the file pointer's position for any given parameter packet.

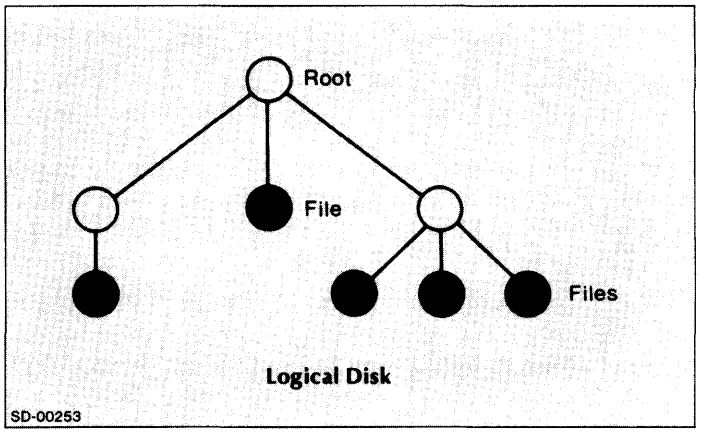


Figure 6-2. Disk Tree Structure

The system does not prevent two or more users from simultaneously updating a record unless the file is *exclusively opened* (this is explained in the ?OPEN call). If simultaneous updates pose a problem, you must use the interprocess (IPC) or intertask communications facilities to synchronize the update requests.

Creating a File at Open Time

You can use the ?OPEN command to create a file before opening it. This use of the ?OPEN command is a common way of creating user data files.

Listing Directory Entries

System call ?GNFN lists all the entries in a given directory to which you have read and execute access. Before issuing ?GNFN you must ?OPEN the directory. Note that this is the only time you issue ?OPEN to open a directory entry.

Deferring or Changing File Specifications

In the cycle of creating a file, opening it, and then performing I/O, you can specify or change the record type, file pointer, and I/O buffer at various points.

If you use the ?CREATE call (discussed in Chapter 5) to create a file, then you can specify the type of record within the data file at creation time. However, when you open the file using the ?OPEN call, you can change the record type from that which you specified when the file was created. You do this by setting the record format bit ?ICRF (offset ?ISTI) in the ?OPEN packet. This sets the record format for that file in all subsequent ?CREATE packets. The default, if ?ICRF was not set, is zero. (See the description of the ?OPEN packet in this chapter.)

When the file is opened, an I/O buffer can be specified. Alternatively, the open parameter packet can indicate that the specification of buffer address will be deferred to the time when I/O is requested. Finally, when reading or writing is requested, the process may specify (or respecify) both the record type and the I/O buffer address.

Extended Parameter Packets

For extended processing of special features during an I/O sequence, AOS allows you to add extended parameter packets to the ?OPEN, ?READ/?WRITE, and ?CLOSE calls. You provide these packets by the following procedure:

1. Set the extended parameter packet bit (?IPKL) in offset ?ISTI of the parameter packet.

2. You must follow the normal parameter packet with a contiguous 16-word address block. In each word of the address block, bit zero serves as a flag bit. The first time you use the extended packet, you must set bit zero of the extended packet address to 1 (see Table 6-2). This tells the system that this word in the address block points to a new extended packet. The system sets the flag bit to zero after executing the call. If you want to change any of the specifications in the extension and then re-use it, be sure to reset the flag bit to 1. If the flag is zero, the system assumes that the packet extension has not changed, and will not re-examine it.

   If you don't want to use the extended function on a particular ?READ or ?WRITE call, set the whole word to zero. You can put the proper address (and the flag bit if the packet has changed) into the word again if you wish to perform the function on a ?READ or ?WRITE.

Bit 0	Bits 1-15	Meaning
0	0	The extended function is not desired in this call.
1	ADDR	The extended function is desired; the packet must be examined. (The bit will be off when the call is finished.)
0	ADDR	The extended function is desired; the packet has not changed and will not be examined.

   NOTE: Undefined results occur if you use this combination before a packet has been examined for the first time.

3. Elsewhere in your code, provide the extended packets pointed to by words in the address block. The structures of these packets are given under the ?OPEN and ?READ/?WRITE calls.

Table 6-2. Extended I/O Address Block

Word Offset	Contents	
	Bit 0	Bits 1-15
?ETSP	flag bit	Word pointer to Screen Management Primitives extended packet. (?READ/?WRITE only.)
?ETFT	flag bit	Word pointer to Selected Field Translation extended packet. (?READ/?WRITE only.)
?ETLT	flag bit	Word pointer to Labeled Magnetic Tapes extended packet. (?OPEN/?CLOSE only.)
remaining words	reserved, must be zeroed.	

Generic Filenames

Generic filenames, listed in directory PER, simplify the use of certain common files. By referring to generic filenames instead of to their actual counterparts, processes avoid the need to know the names of certain actual files. For example, a process may simply open the @LIST file and the system will open the current list file associated with the process. (Be careful not to confuse each process's generic @LIST file with the CLI environment LISTFILE; the latter is explained in the *Command Line Interpreter (CLI) (AOS and AOS/VS) User's Manual.)*

There are six different generic filenames. These files have the following names and conventional uses:

Filename	Use
@CONSOLE	The interactive device associated with a process.
@INPUT	Command input file.
@LIST	The mass output file.
@NULL	A file that remains empty.
@OUTPUT	File receiving short messages.
@DATA	A mass input file.

Any time you want to refer to a generic filename, you must use the @ pathname prefix. This identifies the name as a generic filename, not simply as one of your own filenames. As with standard device names, generic filenames are not reserved. You can name one of your own files INPUT, for example, but it will not be a generic file.

For interactive processes, the @INPUT and @OUTPUT filenames are usually the console device. The @NULL generic filename differs from other generic filenames by not corresponding to any actual pathname. Instead, any output to the @NULL file is thrown away and a read from @NULL generates an end-of-file condition.

The Macroassembler illustrates a typical use of generic filenames. It outputs assembly listings to the @LIST and error messages to @OUTPUT.

Each newly created process may be given actual pathnames corresponding to each of the generic filenames, except @NULL. The actual pathnames are specified in the ?PROC parameter packet:

?PCON	@CONSOLE
?PIFP	@INPUT
?POFP	@OUTPUT
?PLFP	@LIST
?PDFP	@DATA

Each father process can set its son's generic filenames (except @NULL) to reference the same files that are referenced by the father's generic filenames. For example, if the father associates @LPA with @LIST, the father can instruct its son to maintain the same association. To pass its own generic files to its son, the father must block itself until its son is terminated.

Anything the system reads from @INPUT it will copy to the process's @OUTPUT file. This produces character echoing on console devices. If @INPUT, @OUTPUT, and @LIST are set to be devices, then the generic files acquire all the characteristics of the devices. This fact is especially significant for spoolable devices, since each time a spoolable device is opened the system creates a temporary file for it. Upon closing that file, it is enqueued for output to that device. For example, if a line printer is set to be generic file @LIST, then each time @LIST is closed a separate listing is produced on the printer.

@DATA differs from @INPUT by not echoing data to @OUTPUT.

Magnetic Tape Units

Tape units can be opened as a filename and file number, starting with file number 0, e.g., @MTA3:7, the eighth file on unit number 3. The system permits file zero in a tape volume to be specified implicitly. Opening a tape unit without specifying a file number (e.g., @MTA3), opens the first file in that volume. More sophisticated I/O, permitting the specification of the file number when I/O is performed, is described in Chapter 13.

Alternatively, if you have issued the CLI MOUNT command to have the operator mount your tape volume, then you will use the linkname that you supplied in the MOUNT command line in all subsequent references to that tape volume. For example, you might first issue the CLI command

```
MOUNT TAPE1 MOUNT VOLUME AU9058
```

then later open, close, read from, and write to the tape using the linkname TAPE1.

If a user program executes ?READ/?WRITE (or ?RDB/?WRB) system calls to a 7-track tape unit, each 16-bit word will be transferred as two sequential bytes. Only the low-order 6 bits of each byte are written onto the tape or are read from it; the two high-order bits of each byte are set to zeros.

Labeled Magnetic Tapes

A labeled magnetic tape contains your data and information about that data. The information about your data is contained in labels comprising the volume name, the file name, and the file format. The three main advantages of labeled tape are:

- Standard formats allow communication with other operating systems.
- File access can be by name rather than physical file number.
- A logical file can span several physical reels of tape.

Because labeled tapes are designed for interchange between computers, the character set is limited. Only legal characters should appear in the data from which labels are created. The legal characters are:

- Uppercase letters A-Z (lowercase is changed to uppercase).
- Digits 0-9.
- Characters: space ! " % & ' ( ) * + , - . / : ; < > ?

In order to use labeled tapes under AOS, your system must be running a Rev 2.00 or later EXEC, and you must be logged on under the EXEC, either in batch or at a terminal. Since the operator process is not a son of EXEC, you cannot use labeled tapes from the operator console.

You can prepare a tape for labeled tape I/O with the Label utility, documented in the *Command Line Interpreter (CLI) (AOS and AOS/VS) User's Manual*. You can also use the Label utility to delete (SCRATCH) a labeled magnetic tape.

Labeled Tape Format

AOS supports three different formats of labeled tapes:

- AOS format (labels written in ASCII).
- ANSI format levels 1, 2, or 3 (labels written in ASCII).
- IBM format levels 1 or 2 (labels written in EBCDIC).

The formats and levels differ in the number of files allowed in a file set, the allowable record types, the types of labels, and the contents of labels. Table 6-3 defines the number of files and the data formats allowed for each label format and level.

Table 6-3. Label Formats and Levels: Files Per Volume Set, Data Formats

Number of Files:

File	AOS	ANSI1	ANSI2	ANSI3	IBM1	IBM2
Single file, single vol.	X	X	X	X	X	X
Single file, multiple vol.	X	X	X	X	X	X
Multiple file, single vol.	X		X	X	X	X
Multiple file, multiple vol.	X		X	X	X	X

Data Formats:

Format	AOS	ANSI1	ANSI2	ANSI3	IBM1	IBM2
Fixed length (?RTFX)	X	X	X	X	X	X
Variable length (?RTVR)	X			X	X	X
Variable length spanning blocks (?RTVB)						X
Data Sensitive (?RTDS)	X					
Dynamic (?RTDY)	X					
Undefined records (?RTUN)	X				X	X

AOS used only its own format prior to revision 3.04, and now uses its format for default labeling. The CLI uses AOS format, which is not designed for interchange with non-AOS Operating Systems. The ANSI and IBM label formats are designed for interchange between computers running different operating systems. The ANSI format is defined by *ANSI X3.27- 1978* (published by The American National Standards Institute, 1430 Broadway, New York, New York 10018). The IBM format is defined in *IBM Manuals GC26-3746* and *GC26-3795* (published by International Business Machines Corporation, White Plains, New York 10604).

Types of Labels

The following four types of labels each comprise several individual labels. Each individual label is written to a separate block of the tape. All labels are 80 bytes long; the system ignores additional bytes.

Volume Labels	Identify the volume (reel) of magnetic tape. These labels occur only at the start of the volume.
File Header Labels	Identify the file and its characteristics. These labels occur before every file on a labeled tape. If a file spans volumes, each file section starts with file header labels.
End of File Labels	Identify the file and its characteristics. These labels occur after every file on a labeled tape.
End of Volume Labels	Identify the file and its characteristics. These labels occur at the end of a volume of tape. A file may be divided into several sections, each section on a different volume of tape. At every division, end of volume labels occur to indicate the file has not ended, but has spanned volumes.

Figure 6-3 shows how labels and data are written on a labeled tape.

Each type of label contains one or more individual labels. Some labels are necessary, and if not present, the system will return an error. Other labels are used, if present, but are not required; and some are permitted, but are not used. (The permitted labels do not cause errors; the system ignores the information on them.) Table 6-4 lists the types of labels allowed for the various formats and levels.

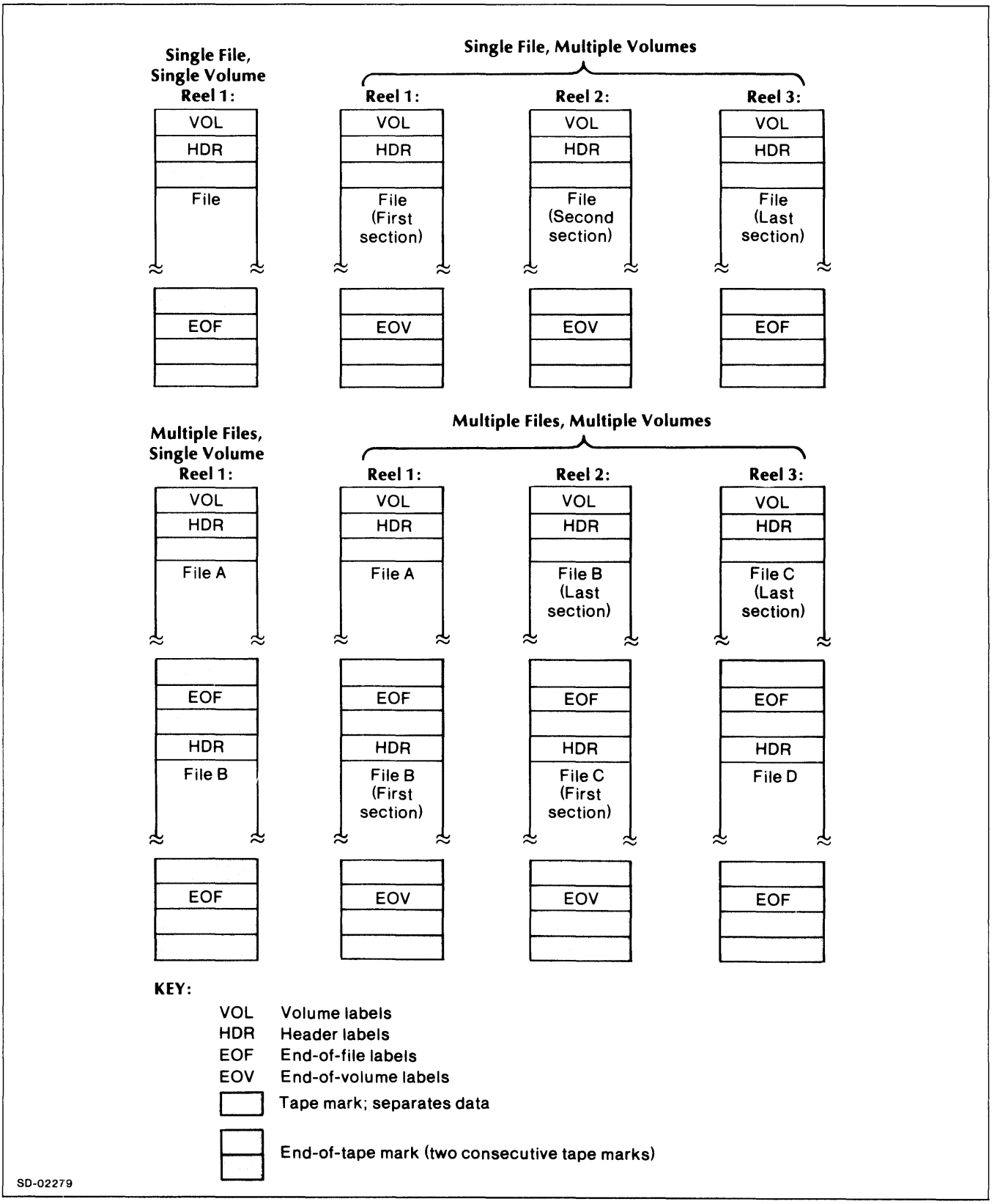


Figure 6-3. Labels and Data on a Labeled Magnetic Tape

Table 6-4. Types of Labels Allowed

Label	AOS	ANSI1	ANSI2	ANSI3	IBM1	IBM2
Volume Labels:						
VOL1	N	N	N	N	N	N
UVL-9	P	P	P	P	N	N
File Header Labels:						
HDR1	N	N	N	N	N	N
HDR2	U	P	P	U	P	U
HDR3-9	P	P	P	P	P	P
UHL1-9	U	P	U	U	U	U
End of File Labels:						
EOF1	N	N	N	N	N	N
EOF2	U	P	P	U	P	U
EOF3-9	P	P	P	P	P	P
UTL1-9	U	P	U	U	U	U
***End of Volume Labels:**						
EOV1	N	N	N	N	N	N
EOV2	U	P	P	U	P	U
EOV3-9	P	P	P	P	P	P
UTL1-9	U	P	U	U	U	U

* The EOV Labels are necessary only if the file spans reels.

where:

N = Necessary

U = Used

P = Permitted

Contents of Individual Labels

Tables 6-5 through 6-9 show the information stored in each individual label for each label type. The first three characters (positions 1-3) identify the label type, and the next character (position 4) identifies the label within the type.

Volume Labels

The volume 1 and user volume labels (see Tables 6-5 and 6-6) are both written by the Label utility or the ?LABEL system call. You can specify the volid, access, and owner name.

You can specify from one to nine user volume labels, each containing up to 76. bytes of data. AOS numbers each label consecutively from one to nine. The data should relate to all the files on this volume.

Note that you cannot use user volume labels for IBM labeled tapes.

Table 6-5. Volume 1 Labels

POSITION	AOS	ANSI1	ANSI2	ANSI3	IBM1	IBM2
01-04	"VOL1"	"VOL1"	"VOL1"	"VOL1"	"VOL1"	"VOL1"
05-10	VOLID	VOLID	VOLID	VOLID	VOLID	VOLID
11	ACCESS	ACCESS	ACCESS	ACCESS	"0"	"0"
12-37	BLANK	BLANK	BLANK	BLANK	BLANK	BLANK
38-41	OWNER NAME	OWNER NAME	OWNER NAME	OWNER NAME	BLANK	BLANK
42-51	OWNER NAME	OWNER NAME	OWNER NAME	OWNER NAME	OWNER NAME	OWNER NAME
52-79	BLANK	BLANK	BLANK	BLANK	BLANK	BLANK
80	VER NO.	VER NO.	VER NO.	VER NO.	BLANK	BLANK

Volume Identifier (VOLID)	This field is one to six characters long and is not duplicated within an installation. This value cannot be defaulted. When you are accessing a file on the volume, include the volid in the pathname of the file.
Access	This field is used in AOS and ANSI tapes. The default is blank, which allows anyone access to the volume. A nonblank character may be written in this field, in which case SUPERUSER must be on in order to access that tape.
Owner Name	This field identifies the owner of the volume; the system ignores it when accessing a file on a volume. The default is a blank owner name.
Version Number	This field refers to the version of the ANSI labeled tape standard used to implement the labeled tape processing. This field must be a 1, 2, or 3 if you intend to read the tape. The system uses version number 3 when you write to tape.

Table 6-6. User Volume Labels

Position	AOS	ANSI1	ANSI2	ANSI3
01-03	"UVL"	"UVL"	"UVL"	"UVL"
04	LABEL NO.	LABEL NO.	LABEL NO.	LABEL NO.
05-80	USER DATA	USER DATA	USER DATA	USER DATA

Header 1, End-Of-Volume 1, End-Of-File 1 Labels

The HDR1 label (see Table 6-7) is identical to the EOF1 and EOV1 labels except for positions 1 through 4, which contain "EOF1" and "EOV1" respectively. Therefore, we will discuss only the HDR1 label.

The fields in the HDR1 label perform three functions:

Error Detection — AOS writes and checks the fields in this area.

Data Protection — For a write operation, you can change the fields in this area from the indicated default by using the extended labeled tape packet in the ?OPEN call. For a read, you can determine the information on the label by using the labeled tape extended packet on the ?OPEN call.

Data Identification — You can use these fields to assure that the file on the tape is indeed the file you requested. The file name cannot be defaulted, and is provided in the ?OPEN packet. When writing, you can change the default by using the extended packet. When reading, you can determine the information in these fields by using the extended packet.

Table 6-7. HDR 1 Labels

POSITION	AOS	ANSI1	ANSI2	ANSI3	IBM1	IBM2
01-04	"HDR1"	"HDR1"	"HDR1"	"HDR1"	"HDR1"	"HDR1"
05-21	FILE NAME	FILE NAME	FILE NAME	FILE NAME	FILE NAME	FILE NAME
22-27	BLANK	FILE SET ID	FILE SET ID	FILE SET ID	FILE SET ID	FILE SET ID
28-31	FILE SEC NO.	FILE SEC NO.	FILE SEC NO.	FILE SEC NO.	FILE SEC NO.	FILE SEC NO.
32-35	FILE SEQ NO.	FILE SEQ NO.	FILE SEQ NO.	FILE SEQ NO.	FILE SEQ NO.	FILE SEQ NO.
36-39	GEN NO.	"0001"	"0001"	"0001"	BLANK*	BLANK*
40-41	VER NO.	"00"	"00"	"00"	BLANK*	BLANK*
42-47	CREATE DATE	"00000"	"00000"	CREATE DATE	CREATE DATE	CREATE DATE
48-53	EXPIRE DATE	EXPIRE DATE	EXPIRE DATE	EXPIRE DATE	EXPIRE DATE	EXPIRE DATE
54	ACCESS	BLANK	ACCESS	ACCESS	ACCESS	ACCESS
55-60	BLOCK COUNT	BLOCK COUNT	BLOCK COUNT	BLOCK COUNT	BLOCK COUNT	BLOCK COUNT
61-73	SYSTEM ID	SYSTEM ID	SYSTEM ID	SYSTEM ID	SYSTEM ID	SYSTEM ID
74-80	BLANK	BLANK	BLANK	BLANK	BLANK	BLANK

* IBM documents indicate these fields contain information, however this information is not used during processing. AOS does not read or write information from or to these fields.

(continues)

Error Detection Fields	
File Section Number	This field indicates which section of a file is currently being processed. The system divides a file into sections when it spans volumes. This assures that the volume mounted contains the proper section of the file and that the file is not processed out of order.
File Sequence Number	This field indicates the sequential number that AOS gives each file on a labeled tape volume set. The wrong sequence number on a file indicates the file was improperly written.
Block Count	This field indicates the number of blocks that were written to the file. If this number in the EOF or EOV label is not the number actually read, a block may have been skipped.
Data Detection Fields	
Expiration Date	This field protects data from being overwritten before a certain date. AOS defaults the expiration date to 90 days after the creation date.
Access Byte	This field indicates who has access to the data. The default value is a blank for ANSI and AOS formats, and is a digit 0 for IBM formats. The default value means anyone has access to the data. A character other than the default value appearing in this field indicates additional privileges are needed before AOS will allow access to the file. You will be allowed access to the file if you have SUPERUSER ON.
Data Identification Fields	
File Set Identifier	This field is the group of files contained on one or more volumes (REELS). The file set identifier is the name of the file set, and is used to assure that a newly mounted volume does indeed belong to that group of files. AOS defaults this name to the volume ID of the first volume in the file set.
File Name	This field identifies this file from the other files in this file set. It cannot be defaulted, and cannot be all blanks.
Generation Number	This field indicates the generation of the file. A file may appear several times on a tape, if each appearance is a different generation of the file. This is useful for archiving changes to a file. The default generation number is 0001.
Version Number	This field indicates which version of the generation is being accessed. The system allows only one (presumably the latest) version of a generation of a file to appear in a file set. The default value for this field is 00.

(concluded)

Header 2, End-Of-Volume 2, End-Of-File 2 Labels

The HDR2 label (see Table 6-8) is identical to the EOF2 and EOV2 labels except for positions 1 through 4, which contain "EOF2" and "EOV2", respectively. Therefore, Table 6-8 illustrates the HDR2 label only.

Table 6-8. HDR 2 Labels

Position	AOS	ANSI3	IBM2
01-04	"HDR2"	"HDR2"	"HDR2"
05	RECORD FORMAT	RECORD FORMAT	RECORD FORMAT
06-10	BLOCK LENGTH	BLOCK LENGTH	BLOCK LENGTH
11-15	RECORD LENGTH	RECORD LENGTH	RECORD LENGTH
16-38	BLANK	BLANK	BLANK*
39	BLANK	BLANK	BLOCK ATTR.
40-50	BLANK	BLANK	BLANK*
51-52	BLANK	BUFFER OFFSET	BLANK
53-80	BLANK	BLANK	BLANK

Record Format	This field indicates the data format. You specify record format in word ?ISTI of the ?OPEN packet. When reading a file that does not have a HDR2 label, and that does not specify a record format, AOS will return a fixed type record format. When writing, AOS will not default this field.
Block Attribute	This field is used along with the record format. The block attribute tells if records are "BLOCKED" (meaning several records in one physical block), "UNBLOCKED" (meaning one record per physical block), or "SPANNED" (meaning a record may be contained in two or more consecutive blocks). AOS writes all records in blocked format. You may request spanning for variable length records with record format ?RTVB.
Block Length	This field indicates the maximum block length of the tape. The ?IMRS field in the ?OPEN packet governs the value written to the label. If you default this field (-1 in ?IMRS), AOS will use the value 2048 when writing, and the value in the label when reading.
Record Length	This field tells the maximum record length in the file which, in the case of fixed length data, is the length of every record. You communicate this via offset ?IRCL in the ?OPEN packet. If you default this field (-1 in ?IRCL), AOS will use the value 210 (136.) when writing, and the value in the label when reading.
Buffer Offset	AOS ignores the buffer offset. However, under ANSI standard, this field indicates, at the start of each physical block, the number of bytes that are not data.

* IBM documents indicate these fields contain information, however this information is not used during processing. AOS does not read or write information from/to these fields.

User Header/Trailer Labels

The user trailer labels (see Table 6-9) have the same format as the user header labels, except positions 1 through 3 contain "UTL" rather than "UHL", and the trailer labels follow the EOF or EOV labels.

Table 6-9. User Header/Trailer Labels

Position	AOS	ANSI2	ANSI3	IBM1	IBM2
01-03	"UHL"	"UHL"	"UHL"	"UHL"	"UHL"
04	LABEL NO.	LABEL NO.	LABEL NO.	LABEL NO.	LABEL NO.
05-80	USER DATA	USER DATA	USER DATA	USER DATA	USER DATA

The user data area in the user header/trailer labels contains information about the file that you may wish to know, but that is not recorded in the system labels. AOS will read or write user header and trailer labels into an area you have specified in an extended label tape packet in ?OPEN.

File I/O on Labeled Magnetic Tapes

Again, to use labeled tapes for file I/O, you must be logged on under the EXEC utility, either in batch or at a terminal. You cannot issue I/O calls against a labeled tape from the operator's console, since the operator process is not a son of EXEC. The OP username must mount all labeled tapes, and the CLI command CONTROL @EXEC OPERATOR ON must be in effect. This command signals EXEC that the operator is available to mount the tapes.

There are two ways to mount a labeled tape: explicitly, by issuing the CLI MOUNT command; or implicitly, by issuing the ?OPEN system call. The syntax of the MOUNT command is:

```
MOUNT/VOLID=volid linkname operator-message
```

where:

linkname	is the name of the link entry associated with the tape's filename.
operator-message	is a text string (which usually instructs the operator to mount the tape).
volid	is the six-character volume identifier.

The MOUNT command creates links for both labeled and unlabeled tapes. When you issue MOUNT against a labeled tape, EXEC passes the message string to the operator and creates a link entry for the filename in your initial working directory. The link resolves to @LMT:volid when you issue ?OPEN, ?READ, ?WRITE, or ?CLOSE to that tape volume. Note that EXEC creates the link entry in your initial working directory (:UDD:USERNAME), not in the directory from which you issued the MOUNT command. When you perform primitive I/O or issue CLI commands against the labeled tape volume, you may substitute the tape's filename for @LMT:volid. After reading or writing to a tape file opened with MOUNT, use the CLI command DISMOUNT to tell the operator to remove the tape from the tape drive.

You can also mount a labeled tape with the CLI command DUMP, or any CLI command that accesses @LMT:volid. When you use this method, EXEC checks to see if the tape is already mounted. If it is not, EXEC directs the operator to mount it. The syntax of the DUMP command is:

```
DUMP @LMT:volid:filename
```

Each time you issue DUMP, EXEC directs the operator to mount and then dismount the tape. Thus, the MOUNT command is usually the more efficient method.

If you mount the labeled tape with the ?OPEN call, offsets ?IFNP and ?IFNL point to the name of the tape volume, which must be in the form

@LMT:volid:filename

The system does not create a link when you use this method, but it does tell the operator to mount the labeled tape volume specified in the pathname. When you issue a ?CLOSE to that tape file, the system directs the operator to dismount the labeled tape volume.

Mounting a labeled tape explicitly with MOUNT is the most efficient way to perform I/O on more than one labeled tape file, since the system does not need to rewind and reposition the tape for each I/O sequence, or direct the operator to mount and dismount the tape for each ?OPEN and ?CLOSE. However, ?OPEN is useful because it gives you the option of creating and opening the tape file at the same time.

When you issue ?READ or ?WRITE to a labeled tape, refer to the tape by one of the following pathnames:

@LMT:volid:filename

where:

@LMT	is the generic filename for a labeled tape.
volid	is the volume ID number.
filename	is the name of the file you wish to access.

:UDD:username:linkname:filename

where:

UDD	is the name of the user directory.
username	is your username.
linkname	is the name of the link entry created by EXEC when the tape was mounted.
filename	is the name of the tape file.

You don't need to cite a specific tape unit number for either of these formats. Use the second format if your current working directory is not :UDD:USERNAME.

EXEC creates the @LMT entry and assigns the file type for labeled tapes (?FGLT). The filename you use must comprise at least 1 but not more than 17 characters from the same character set used for volid.

The volid must contain from one to six characters. If you are referring to a volid or a filename on a labeled tape through a CLI command, then it can contain only the standard AOS filename characters (see *Command Line Interpreter (CLI) (AOS and AOS/VS) User's Manual*).

However, if you are referring to the volid or filename within a program, then any of the ASCII characters in positions 2/0 through 7/14 can be included in a volid or a filename on a labeled magnetic tape. Note that you may use the space character and all of the special characters, and that the system treats uppercase and lowercase letters the same.

Selecting Format and Level

The tape's use determines which format and level of labeling you select. If the tape is to be used on a non-AOS system, then you should select ANSI or IBM format. Select the level according to the level of label support on the system receiving the tape. The default level is the highest (ANSI level 3 and IBM level 2); however, you may select a lower level of labeling by using an extended packet. When writing, AOS will record less information about your data in the labels if you selected a lower level. Therefore, it is best to use the highest level when writing to a labeled tape. If you select a lower level of labeling for a read, AOS will ignore some of the information in the labels. AOS can read a tape of lower level than is requested (i.e., a level 1 ANSI tape can be read if you select level 3 ANSI). Therefore, we recommend that you use the highest level for reading.

Multiprocessor Communications Adapters

The Type 4206 Multiprocess Communications Adapter (MCA) permits full duplex interprocessor communications via the data channel. The MCA is analogous in its software operation to a magnetic tape unit. A process transferring MCA data exists either within a single system, or within up to 14 other distinct systems, or both. Use of a second 4206 MCA, MCA1, permits up to 15 similar additional communications links to be accessed. *

The I/O cycle used with MCAs is the same as with tape files: ?OPEN, ?READ, ?WRITE, and ?CLOSE. Functionally, each MCA is actually two separate devices: an MCAT, which handles transmissions between processors, and an MCAR, which handles receives. You can use MCATs only to transmit, i.e., write data. Likewise, you can use MCARs only to read data from another processor. Thus, the ?READ system call applies to MCARs only, while ?WRITE applies only to MCATs. ?OPEN and ?CLOSE apply to both kinds of MCAs.

Each MCA link corresponds to a pathname in one of the following forms: @MCAT:n or @MCAR:n; or @MCAT1:n or @MACR1:n, where n is the number of the CPU link. Links may be in the range 0-15, but you can use unit number 0 only to read from a link to an MCAR. When unit 0 is used in reading, it means *read from any link.*

Direct MCA I/O, circumventing line protocol, is possible. This is described in Chapter 13.

Character Devices

Character devices are those devices which perform I/O in units of 8-bit bytes. Typical character devices are the console and the paper tape punch. Character devices have the names specified at the beginning of this chapter. Character device names are entered into the peripheral directory, :PER, when the system is bootstrapped.

Character devices require special consideration because they can operate in one of two modes, because of the special meaning attached by them to portions of the I/O parameter packet, and because of the special system calls which are provided for their management.

Each character device can be operated in either the text or binary modes. In *binary mode,* each 8-bit byte is passed without interpretation by the system. In *text mode,* characters are masked to 7 bits on both input and output, with the 8th bit reserved for parity. The system can check for even parity on input, and can force even parity on output (see the device characteristics words). Characters in text mode are subject to interpretation based on specifications in each device's characteristics words. This means, for example, that output of the ASCII sequence CTRL-P, CTRL-I, CTRL-A will not print as expected if characteristics/st (simulate tabs) is enabled. Instead, the output data stream will be CTRL-P followed by from 0 to 7 spaces (depending on which column the CTRL-I occurred in) followed by CTRL-A. Additionally, output character devices in text mode may also be placed into page mode. In *page mode,* output will stop automatically after the number of lines specified in the device characteristics words or on a form feed. To display the next page type CTRL-Q, issue ?READ, or issue ?WRITE with "force output."

There are three characteristics words for each character device. These words specify information such as whether or not the device is an ANSI standard device, whether to perform echoing, whether or not it has hardware tabs and form feeds, etc. The characteristics words are described in detail under system call ?GCHR, which reads these words. System call ?SCHR permits these words to be set or modified.

System call ?SEND transmits a message to a process's console if that console has "message receipt" enabled (?CNRM set to 0 in the device's characteristics word). This method of transmitting a message avoids the need to open or close the device. The advantage of this call is that it permits a process to send a message to any other process's console (if message receipt is enabled). This is particularly useful for a process (like a real-time process) which lacks a console of its own, and provides a convenient way to communicate to the operator's console.

Card Readers

Card readers are character devices, but deserve special consideration for the following reasons:

1. Hollerith to ASCII conversion is done on text ?READs.
2. Once you open it, a card reader is started for input and will *read ahead* as many cards as it can fit into the ring buffer. It will not start again, however, until there is room for one whole card in the ring buffer.
3. The system assumes that all cards are at most 81 columns long. Columns are not checked on input, so a mark-sense card reader is system compatible.

* 4. Device characteristics also allow you to have *trailing blanks suppressed,* so that a NEW LINE follows the last nonblank character on each card. This also allows more cards to fit into the ring buffer.

5. A file read error (ERFIL) is returned if the system encounters a non-Hollerith code on a *text* read.
6. An end-of-file condition is produced by reading a card containing all rows punched in column 1.

Full Duplex Modems

Full duplex modems are also character devices which require special consideration. The following definitions are used in the guidelines listed below.

Abbreviation	Definition
DSR	Data Set Ready.
DTR	Data Terminal Ready.
RTS	Request to Send.
Modem OFF	Both DTR and RTS are OFF.
Modem ON	Both DTR and RTS are ON.
Modem Connected	DSR is ON.
Modem Disconnected	DSR is OFF.

Guidelines

1. During initialization, all lines on the ALM are reset to Modem OFF.
2. Upon execution of the first ?OPEN system call, the status is changed to Modem ON.
3. No I/O takes place until Modem Connected is signaled.
4. I/O terminates with an error return if Modem Disconnected occurs.
5. A remote or dial-up console is logged off when Modem Disconnected occurs.

Assigning Character Devices to Processes

Output to devices can be spooled. If a device has spooling enabled, then its character stream is diverted temporarily to a disk file until the device (file) is closed. The entire spooled output file is then automatically queued to be written out to the device (see ?ENQUE, Chapter 9). If an output character device is being spooled, then more than one process can appear to open and write to it simultaneously.

If a device is not being spooled, then it is assignable. If a device is *assigned* to a process, then only that process may use the device. Device assignments can be made either explicitly or implicitly. Explicit assignment is performed by the ?ASSIGN call, and implicit assignment occurs if a non-spooled device is opened without being explicitly assigned.

If a device has been assigned explicitly by a process, then it must be explicitly deassigned by system call ?DEASSIGN before another process can assign it. If the device has been assigned implicitly, then closing the device or terminating the process will deassign the device. A process can open the same device (file) more than once; the last close will close it. The last close will also deassign the device if it was implicitly assigned.

All son processes can share the father's console. If they do, however, then only the most recently created son sharing the console can issue control calls to it. The following are console control calls: ?OPEN, ?CLOSE, ?ASSIGN, ?RELEASE, ?GCHR, and ?SCHR. Meanwhile, the father process and other sons sharing the console are restricted to issuing only ?READ and ?WRITE to the console.

Line Printer Format Control

If you write to a line printer, both data channel and nondata channel, controlled by the EXEC spooler, you may control the output format with the contents of a User Data Area (UDA). The spooler will use default form parameters to print all files unless you override them. The default parameters will be EXEC's current form settings for that device (i.e., CTRL @EXEC LPP and CTRL @EXEC CPL, or CTRL @EXEC default forms). If the file to be printed was queued with the /FORMS switch, the contents of that form's UDA (in the :UTIL:FORMS directory) will be used for the default instead. All forms entries in the :UTIL:FORMS directory must have forms control UDA's. Attempted use of forms entries without form control UDA's will cause errors. Additionally, you can override the current default by generating a forms control UDA for the file you are going to print.

You can use the AOS utility program FCU (Forms Control Utility) to create, edit, and copy UDA format definitions. See the *AOS Command Line Interpreter User's Manual* for documentation of this utility.

Console Control Characters and Control Sequences

A *control character* is a character that you type while you simultaneously depress the CTRL key. The following control characters have been defined to the operating system; control characters are not passed to the user program.

CTRL-D is an end-of-file character. This control character will terminate the current ?READ in progress from the console keyboard and cause an EOF condition code to be reported.

CTRL-O discards all output directed to the console until the next ?READ from the console is issued, or until the next ?WRITE to the console is issued with force output (?IFOP, specified in the I/O parameter packet). CTRL-O is a toggle switch -- it resets with a second CTRL-O.

CTRL-P causes the next character typed on the console to be accepted literally; that is, the system will not interpret the next character as a control character.

CTRL-S suspends output to the console; for example, it freezes data shown on a CRT screen. Unlike CTRL-O, CTRL-S discards no information; it merely suspends the display of more data until you type CTRL-Q.

CTRL-Q restarts the display of output on a console screen. Thus it resets a previous CTRL-S sequence, or displays a new page if the device is in page mode.

CTRL-U erases the current line of console input.

A *control sequence* is a CRTL-C character followed by any other control character within the range CTRL-A through CTRL-Z. If you follow a CTRL-C with a character other than one of the control characters, the system ignores the CTRL-C. This allows you a way to terminate an inadvertent control sequence.

Specifically, when the system detects a CTRL-C character, it checks the next character you type to determine whether it is CTRL-A, CTRL-B, CTRL-C, or CTRL-E. If it is one of those control characters, AOS takes the appropritate action; otherwise the system ignores the second control character and does not pass it to the program. Note that in all cases the second character in a control sequence disables, that is, terminates, the control sequence.

The control character sequences operate as follows. CTRL-C CTRL-A generates a console interrupt only if a console interrupt task has been defined (see ?INTWT, Chapter 9) by the target process. CTRL-C CTRL-B generates a console interrupt and unconditionally terminates the current process. CTRL-C CTRL-C echoes as ↑C ↑C and does not affect process execution; thus it enables you to determine if the system is up. CTRL-C (followed by anything) drops characters that have not been echoed yet. This allows the next ?READ call to start clean. CTRL-C CTRL-E terminates the current process and creates a *break file,* a memory image of the terminated process.

Using the IPC as a Communications Device

The system supplies a simple I/O procedure by which you can use the IPC facility to provide a full-duplex communications link between two processes, buffered in standard first-in, first-out fashion. The mechanisms for this procedure are the I/O calls ?OPEN, ?READ/?WRITE, and ?CLOSE.

For IPC communications of this kind to occur, AOS requires the following synchronization protocol, first, the calling process creates the IPC entry with an ?OPEN call, specifying ?OFCR in offset ?ISTI of the ?OPEN packet, and file type ?FIPC in offset ?ISTO. The system then issues a global ?IREC call on that entry, signifying that the entry is available for messages from any process. (A global ?IREC is one in which the origin port number in the ?IREC header is 0. See Chapter 4.)

The other process issues a complementary ?OPEN, which the system follows with an ?ISEND to synchronize the communications. The process which creates the IPC entry (by issuing the first ?OPEN) is the owner of that entry.

Thereafter, either process can issue ?READ or ?WRITE calls through the established IPC entry. If one of the communicating processes closes the link or terminates, the system sends the other process an end-of-file signal (exceptional condition EREOF) if it attempts another ?READ.

Note that all ?READs and ?WRITEs are synchronized. Thus, for a process to receive the other's termination signal, it must ?READ it in the proper sequence. Otherwise, the process could attempt to ?WRITE to the closed IPC link indefinitely; there is no exception return in that case.

One way to avoid that condition is to establish interprocess communications through the connection management facility, described in Chapter 11.

System Call Summary

The following system calls let you manage file I/O (you can also use ?OPEN to create data files):

?ASSIGN	Assign a character device to a process.
?CLOSE	Close a file.
?CRUDA	Create a UDA for a file.
?DEASSIGN	Deassign a character device from a process.
?FEOV	Force an end-of-volume condition for a labeled magnetic tape.
?GCHR	Read the characteristics words.
?GNFN	List directory entries.
?GPOS	Get a file pointer position.
?LABEL	Label a magnetic tape.
?OPEN	Open a file.
?RDUDA	Read a UDA.
?READ	Read a record.
?SCHR	Set the characteristics word.
?SDLM	Set the delimiter table.
?SEND	Send a message to a console.
?SPOS	Set the file pointer position.
?STOM	Set time out value.
?TRUNCATE	Truncate a file at current position.
?UPDATE	Flush file descriptor information.
?WRITE	Write a record.
?WRUDA	Write a UDA.

These calls are discussed individually in the following sections.

?ASSIGN

Assign a device to a process.

?ASSIGN
exception return
normal return

Input/Output

Input:

AC0 byte pointer to name of device being assigned.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

ERDAI Device is already assigned.

FILE SYSTEM codes

Description

This call assigns a device for the exclusive use of the calling process. Each assigned device remains assigned until it is explicitly deassigned by ?DEASSIGN or implicitly deassigned when the processs which assigned the device is terminated. No device can be assigned which is currently being spooled.

?CLOSE

Close a file.

?CLOSE *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 address of parameter packet.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERIBM Inconsistency detected in block allocation map upon attempted deallocation of disk blocks.

FILE SYSTEM codes

SYSTEM CALL codes

CHANNEL-RELATED codes

Description

This call closes a file or device. Files must be closed after use to ensure the updating of file status information and to free channels for other use. All files are automatically closed when the process which opened them is terminated.

The portions of the I/O packet meaningful to this call are: the channel number (?ICH) and the packet type (?IPKL, in the specifications word ?ISTI). However, no extended packets are currently defined for the ?CLOSE call. Figure 6-4 shows the ?CLOSE parameter packet.

?CLOSE (continued)

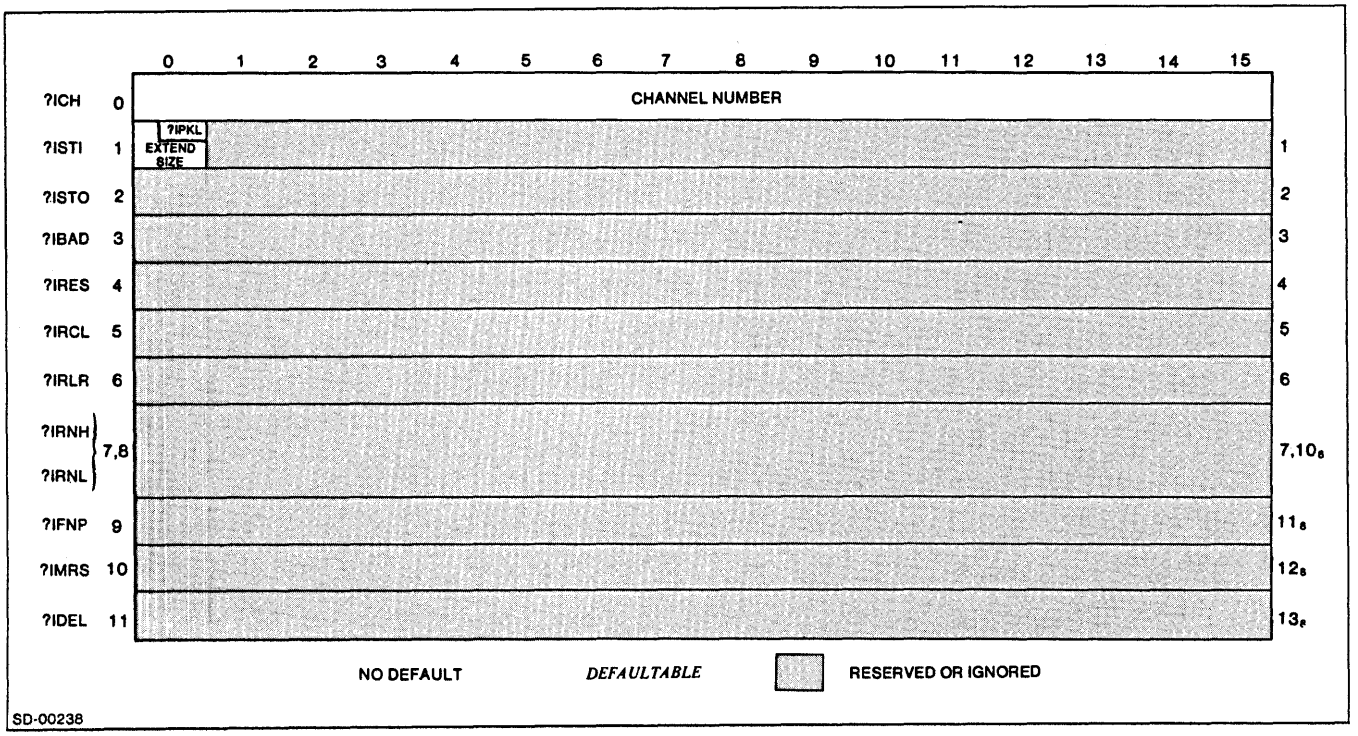


Figure 6-4. ?CLOSE Parameter Packet

?CRUDA

Create a User Data Area (UDA).

?CRUDA
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname of file. *

Output:

AC0 unchanged. *

Exceptional Condition Codes in AC0

ERUAE UDA already exists.

FILE SYSTEM codes

Description

This call allows you to create a 128-word area associated with a file. Once created by ?CRUDA, a UDA cannot be deleted unless you delete the file with which it is associated. You can define the contents and use of any UDA, although the first word in a UDA must be the file type of its associated file. Data General reserves codes in which bit 0 of the first UDA word is zero; codes in which bit zero is set to 1 are available to users. The UDA associated with a file is not affected by normal I/O calls (?WRITE, ?READ, etc.). You use calls ?WRUDA and ?RDUDA for writing to and reading from a UDA.

?DEASSIGN

Deassign a process's device.

?DEASSIGN
exception return
normal return

Input/Output

Input:

AC0 byte pointer to name of device being deassigned.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

ERDAI Device is opened (and thus cannot yet be deassigned).

ERARU Device is already deassigned.

ERARC Attempt to release the process console.

FILE SYSTEM codes

Description

This call explicitly deassigns a device that was previously assigned by ?ASSIGN. The console device (specified in the ?PROC parameter packet) cannot be deassigned, nor can any device which is currently opened.

?FEOV

Force end-of-volume on labeled tape.

?FEOV
exception return
normal return

Input/Output

Input:

AC1 channel number on which the tape was opened.

Output:

AC1 unchanged.

Exceptional Condition Codes in AC0

ERIFT Illegal file type; file is not a labeled tape.

FILE SYSTEM CODES

Description

During a ?WRITE call, the system flushes the current buffer to the tape, and end-of-volume processing occurs just as in the normal end-of-tape case. A swap to the next reel occurs.

During a ?READ call, the system spaces the tape to its logical end. If AOS detects EOV labels, it will swap to the next volume. If AOS detects EOF labels, it will return the error code EREOF.

?GCHR

Read the device characteristics.

?GCHR
exception return
normal return

Input/Output

Input:

AC0	byte pointer to name of device or channel number.
AC1	bit 0=1, AC0 contains a channel number.
	bit 0=0, AC0 contains a byte pointer.
	bit 1=0, get current characteristics.
	bit 1=1, get default characteristics for specified device.
AC2	address of 3-word area to receive the device characteristics.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

INITIALIZATION AND RELEASE codes

Description

This call retrieves the characteristics of any character device which was assigned to the calling process. It copies the three characteristics words associated with the device into the buffer whose pointer you supply in AC2. If, on input, bit 1 of AC1 is 0, this call returns the current characteristics settings. If, on input, bit 1 of AC1 is set to 1, this call returns the default characteristics settings for the specified device. These three characteristics words are defined as shown in Figure 6-5 and described in Table 6-10.

The three word buffer filled in by the ?GCHR call may be used as the input characteristics for the ?SCHR (set characteristics) call, after you make whatever changes are desired. The following discussion should be helpful both for interpreting the output of ?GCHR and setting the input to ?SCHR. Setting bit ?CEPI in word 0 permits even parity only on input. If you do not set ?CEPI, the device will accept either parity on input.

If your terminal is non-ANSI standard, ?CNAS must be set. This may be the default value at your installation, or you may have to set it with the ?SCHR call.

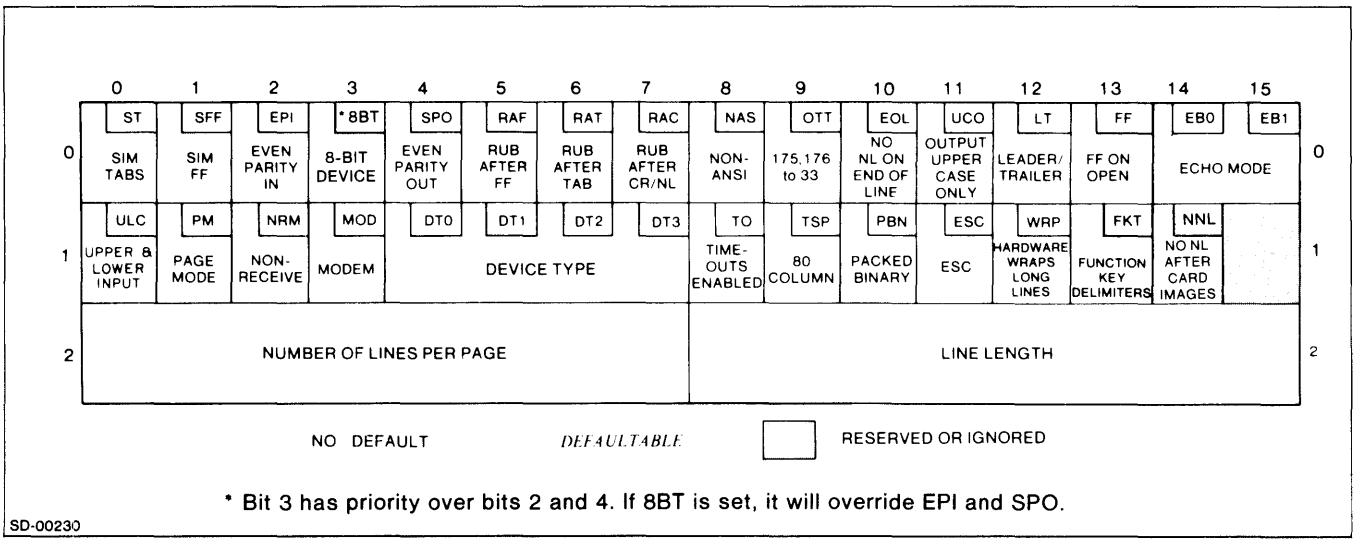


Figure 6-5. Character Device Characteristics Words

An ANSI standard terminal performs both the CR and LF functions when it receives a NEW LINE character. DG Models 6052 and 6053 are ANSI standard terminals. DGC Models 4010, 4010I, 6040, and 6012 are non-ANSI standard. All of our ANSI standard terminals have NEW LINE as a big key and CR as a small key. On the non-ANSI standard terminals, the reverse is true: LF is a small key and CR is a big key.

When your terminals are properly matched to their characteristics, AOS will use the big key to mean NEW LINE and the smaller key to mean CR. Thus, on input from non-ANSI standard terminals, the CR key (big key) translates to NEW LINE, and the LF key (smaller key) translates to CR. On output to non-ANSI standard terminals, AOS always precedes LF with a CR.

On certain old terminals, the ESC character is not octal 33, and thus bit ?COTT must be set. Bit ?CLT is normally set for paper tape punches to provide leader and trailer.

The following echo modes are defined for bits ?CEB0 and ?CEB1.

Echo Mode Value	**Result**
blank	No echoing occurs.
?CEOS	Echoing occurs. Each character is echoed exactly as it was input.
?CEOC	Echoing occurs. Control characters are echoed with up-arrows (for example, CTRL-C as ↑C) and escape characters are echoed as dollar signs.

If bit ?CULC is not set to 1, then input alphabetic characters will be translated to uppercase characters.

If a character device is on a line controlled by a modem, ?CMOD must be set before you can perform I/O. It's possible that the system was generated so that ?CMOD is set by default when you open character devices. If this is not the case at your installation, you must set ?CMOD via the ?SCHR call. In either case, the first read or write operation will block or suspend the calling process or task until the link is established.

Table 6-10. Character Device Characteristics Words

Word	Bit Position	Purpose or Meaning
0	?CST	Simulate tab settings (in every 8th column).
	?CSFF	Simulate form feeds.
	?CEPI	Accept even parity on input.
	?CSPO	Set even parity on output.
	?CRAF	Send 21 rubouts after each form feed.
	?CRAT	Send 2 rubouts after each tab.
	?CRAC	Send 2 rubouts after each carriage return or NEW LINE.
	?CNAS	The device is non-ANSI standard.
	?COTT	Convert octal 175 and 176 to 33 octal.
	?CEOL	Do not automatically output a new line character if the line length is exceeded. That is, truncate the line if its length exceeds the value in word 2.
	?CUCO	Convert lowercase output to uppercase.
	?CLT	On output, write 60 nulls upon open and close. On input, skip leading nulls on open.
	?CFF	Output a form feed upon open.
	?CEB0	Echo mode bit 0.
	?CEB1	Echo mode bit 1.
1	?CULC	Input both upper- and lowercase.
	?CPM	Device is in page mode.
	?CNRM	Prevent this device from receiving messages sent by system call ?SEND.
	?CMOD	Device is attached to a modem interface.
	?CDT0	Device type, bit 0.
	?CDT1	Device type, bit 1.
	?CDT2	Device type, bit 2.
	?CDT3	Device type, bit 3.
	?CTO	Device timeouts are enabled on reads and writes.
	?CTSP	Card reader - retain trailing blanks.
	?CPBN	Card reader - packed binary reads.
	?CESC	ESC character produces a CTRL-C CTRL-A sequence (see Chapter 9).
	?CWRP	Lines which exceed the number of characters in bits 0-7 of Word 2 are continued on the next line of output.
	?CFKT	Data sensitive reads may be delimited with a function key followed by any character. Neither character will be echoed and both characters will enter the buffer.
2	0-7	Lines/page.
	8-15	Line length in characters.

If the link is broken, you will receive error ERDCT, modem disconnect, on the outstanding or subsequent I/O operation on that channel. If you receive error ERDCT on a write operation, you should close and reopen the device (you must also set ?CMOD when you open the device, if the system does not), then rewrite the previous record. If you receive error ERDCT on a read operation and you desire to perform additional reads, you must re-establish the data link as just described. In this case, of course, the decision to transmit the previous record is left to the sender.

You specify device type with a four-bit quantity in word 1, bits ?CDTO through ?CDT3. Devices and their special characters are defined in Table 6-11.

Table 6-11. Devices and their Special Characters

Type	Name	Move Left	Move Right	Erase Line
0	DGC 4010A 6040	n/a	n/a	n/a
1	DGC 4010I	CTRL-Z	CTRL-Y	CTRL-K
2	DGC 6012	CTRL-Y	CTRL-U	CTRL-K
3	DGC 6052 and 6053	CTRL-Y	CTRL-U	CTRL-K
4	Other CRTs	ESC,D	ESC,C	ESC,K
5-15	Undefined	Undefined	Undefined	Undefined

NOTE: For device type 0, each rubout is echoed as SHIFT O (backarrow or underscore) and CTRL-U is echoed as ↑U-NEW LINE. For all other types, rubout is echoed as move left, space, move left and CTRL-U is echoed as carriage return, move right, erase line.

?CTSP, ?CNNL, and ?CPBN are meaningful only for card readers. Without ?CTSP, trailing blanks are suppressed; with ?CTSP, trailing blanks are retained. Without ?CPBN, each column is right justified in a single memory word; with ?CPBN, four columns are packed into three memory words. Without ?CNNL, NEW LINE characters are automatically added after card images; with ?CNNL, they are not added.

?GNFN

List directory entries.

?GNFN *[packet address]*
exception return
normal return

Input/Output

Input:

AC1 channel number on which the directory is opened.

AC2 address of three-word packet.

Output:

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

EREOF No more entries in the directory.

ERIFC Illegal template character. Legal template characters are asterisk (*), dash (-), and plus sign (+).

ERFTL Template exceeds 63 characters.

ERNDR The specified channel is not opened on a directory.

ERCIU Another system call is outstanding on this channel (this error can be received in multitask programs only).

ERFNO The channel is not open.

FILE SYSTEM codes

Description

This call lists the next filename in the directory; i.e., on the first call you will receive the first filename, on the second call you will receive the second filename, etc. You can specify a template in ?NFTP to list the next filename that matches that template. The filename will be copied to the area pointed to by the byte pointer input in ?NFNM.

Before you can issue this call you must ?OPEN the directory (note this is the only use of ?OPEN to open a non-data file). See the description of ?OPEN for directions in using ?OPEN this way. Having opened the directory, you pass the number of the channel used to open the directory in AC1 as input to ?GNFN. In AC2 you pass the starting address of a three-word parameter packet, defined as follows:

Offset	Contents
?NFKY	0 initially; used by the system thereafter.
?NFNM	Byte pointer to the area receiving each filename entry.
?NFTP	Byte pointer to a template; -1 if no template.

When you issue this call initially, set the contents of ?NFKY to zero. After receiving the first filename entry, reissue ?GNFN (by now the system has changed the contents of ?NFKY; therefore, you must not modify these contents). Continue to reissue ?GNFN until control takes the exception return with code EREOF, signaling the last filename entry in the directory.

?GPOS

Get the file pointer position.

?GPOS *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 address of parameter packet.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This system call returns the current file pointer position associated with a given I/O parameter packet. The file pointer position is returned as a double precision value in ?IRNH and ?IRNL of the I/O packet. Position 0 corresponds to the first byte or character in the file, etc. The file pointer is ordinarily incremented when reading or writing, but the pointer can be modified without I/O by setting ?IRNH/?IRNL and ?IPST appropriately in the I/O packet; see ?READ/?WRITE and ?SPOS.

?LABEL

Label a magnetic tape.

?LABEL *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 address of label parameter packet.

Output:

AC0 unchanged.

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERVOL Incorrect labeled tape volume mounted.

FILE SYSTEM codes

Description

This system call allows the user to label a tape under user program control.

Offset	Contents	
?LBFG	Status Bits	
	?LBIM	Create an IBM format label.
	?LBSC	Scratch the tape. (This checks to be sure the volid specified is the tape to be mounted.)
	?LB8	800 BPI.
	?LB16	1600 BPI.
	?LBAM	Automatic density matching.
?LBDV	Byte pointer to magnetic tape unit to be used.	
?LBVD	Byte pointer to volume identifier (up to 6 characters).	
?LBST	Left byte = label level (should be 4). Right byte = number of user volume labels.	
?LBUV	Byte pointer to user volume labels. Each label is up to 76 characters; labels are separated by a single null.	

?LBOI	Byte pointer to owner identifier (or -1 to indicate no owner identifier).
?LPAC	Left byte reserved. Right byte is an accessibility character; if access is not allowed, only superusers can read or write to the volume. (To allow access, set ?LPAC to "space".)
?LBR1	Reserved. Should be set to 0.
?LBR2	Reserved. Should be set to 0.
?LBLN	Length of ?LABEL packet.

?OPEN

Open a file.

?OPEN *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 address of parameter packet.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

SYSTEM CALL codes

CHANNEL-RELATED codes

Description

Before a file can be accessed, it must be opened. Upon opening, the system assigns a channel to the file. You specify information about the file in the parameter packet. This information includes file type, record format, and type of file access permitted. The same packet format is used by ?OPEN, ?READ, ?WRITE, and ?CLOSE. Up to 63 channels can be assigned to each process.

Some information (like record format) can be defaulted to what was specified when the file was created. Other information specified at open time can be changed when I/O is performed; e.g., record format and buffer address.

All reserved words in the parameter packet must be zeroed, unless another system call uses them (i.e., they're ignored for the current call). The parameter packet has the structure shown in Figure 6-6 and described in Table 6-12. Default values are obtained by using the parenthesized settings. (Note that if you close and reopen a file, any default values you placed in the packet will be overwritten by the system, and you will have to reset them before using the packet again.) Each parameter is described at greater length at the end of this section.

Parameters in the File Specifications Word, ?ISTI, represent masks, not bit positions. Thus, to select more than one option sum the masks in this word. For example, to select exclusive open and read only, code ?IEXO+?OFIN in offset ?ISTI of the parameter packet. Unused bits in ?ISTI must be set to zeros.

You can also use the ?OPEN call to read portions of a file into a shared page, provided the file has an element size of four or a multiple of four. To do this, set the ?SHOP bit in offset ?ISTO of the ?OPEN packet. This directs the system to read the specified file portion into a shared page for access by more than one process. The shared file facility guarantees each process the same (and thus the most current) copy of a file.

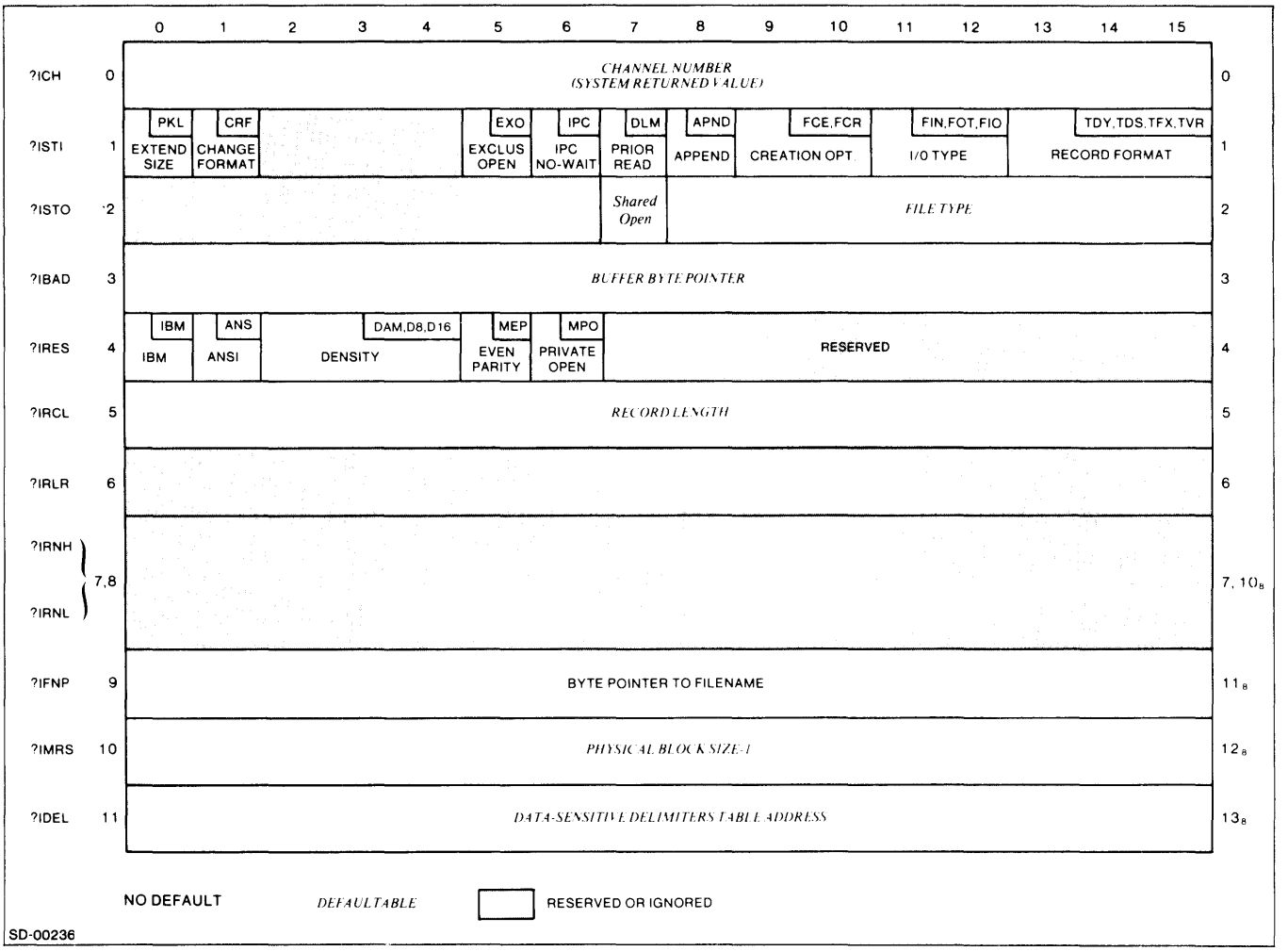


Figure 6-6. ?OPEN Parameter Packet

Table 6-12. ?OPEN Parameter Packet

Offset	Contents	Default
?ICH	Channel number assigned by system.	Assigned by system regardless of the contents of ?ICH.
?ISTI	File Specifications Word:	No default except as noted.
	Packet type ?IPKL - this parameter packet is extended in size to allow extended processing.	(0) Only I/O as described in this manual will be performed. All applications currently require the omission of this field.
	Format select ?ICRF - change format to that in record format field.	(0) The record format selected at ?CREATE is used.
	Absolute file pointer positioning (used only by ?READ/?WRITE).	n/a
	Append ?APND - open the file for appending.	(0) The file pointer is set to the first word in the file.
	Binary I/O (used only by ?READ/?WRITE).	n/a
	Force output (used only by ?READ/?WRITE).	n/a
	Exclusive open ?IEXO - permit no other task, in this process or in any other process, to open the file until it has been closed.	(0) This is a non-exclusive open.
	Priority read ?PDLC - setting this bit opens a file for a priority read. This bit is ignored at ?READ or ?WRITE time.	(0) File is opened with normal priority.
	Creation option ?OFCE - attempt to open the file; if it does not exist, create the file and then open it.	(0) If the file does not exist, do not create it - instead, return an open error.
	?OFCR - create the file and then open it.	
	?OFCR + ?OFCE - delete the file if it exists; create the file and then open it.	
	Input/output ?OFIN - open for input.	none
	?OFOT - open for output.	none
	?OFIO - open for input and output.	none
	Record format ?RTDY - dynamic record type.	none
	?RTDS - data sensitive record type.	none
	?RTFX - fixed length data type.	none
	?RTVR - variable length data type.	none
	?RTUN - undefined record format (read physical blocks).	none

(continues)

Table 6-12. ?OPEN Parameter Packet

Offset	Contents	Default
?ISTO	*Right byte:* File type (see Table 6-13).	(0) If file exists, then whatever you supply here is ignored, and the type is returned by the system. If file does not exist and you select ?OFCE or ?OFCR, or both in ?ISTI, then the type defaults to user data file (?FUDF).
	Left byte: ?SHOP (bit 7) Open this file for system's shared file facility.	(0) Normal ?OPEN.
?IBAD	Byte pointer to record I/O buffer.	(-1) Deferred to ?READ/?WRITE operation.
?IFLG	Bit mask:	
	?OIBM - IBM labeled tape.	(0) AOS labeled tape.
	?OANS - ANSI labeled tape.	
	?IDAM - density automatically selected by system.	(0) Select the density mode set in the AOSGEN dialogue.
	?ID8 - density 800 BPI.	
	?ID16 - density 1600 BPI.	
	?IMEP - select even parity.	(0) Select odd parity.
	?IMPO - private open.	(0) Do not open privately. NOTE: This bit and the ?IEXO bit in the ?ISTI word must be set to select private opening.
?IRCL	Record length.	(-1) Deferred to ?READ/?WRITE operation.
?IRLR	(used only by ?READ/?WRITE).	n/a
?IRNH/ ?IRNL	(used only by ?READ/?WRITE).	n/a
?IFNP	Byte pointer to name of file being opened.	none
?IMRS	Buffer size -1, in bytes. (For magnetic tape, physical block size = buffer size. Also, for magnetic tape, ?IMRS should be odd; that is, physical block size must be even.)	(-1) Buffer size equals 2048 bytes.
?IDEL	Address of data sensitive delimiter table.	(-1) NEW LINE, form feed, carriage return, and null are the data sensitive delimiters.

(concluded)

When you use this mechanism, the system automatically performs an ?SOPEN (shared page ?OPEN) and initiates the standard shared page mechanisms (?SPAGE, ?RPAGE).

Restrictions

The ?SHOP bit mask specifies a shared file open. You must specify ?OFIO or ?OFOT for shared file opens. ?IEXO, ?OFCR, and ?OFCE are illegal options for shared file opens.

Cautions

The shared file facility provides no protection against another user modifying the same record, nor does it protect against modifications to the file by users who obtain their own (unshared) copy using the ?OPEN call.

General Usage Considerations

There are four record formats specifiable in offset ?ISTI. *Dynamic* is a record whose size (in bytes) is specified when an I/O transfer is requested. *Data sensitive* records are records which are delimited by any of a set of characters. If a delimiter table (?IDEL) is set to -1, then the ASCII characters NEW LINE, carriage return, form feed, and null are default delimiters. If some set of characters other than these default characters are desired, then they are specified by setting bits in a delimiter table.

The *delimiter table* consists of sixteen consecutive 16-bit words which form a table of 256 bits. Each bit in this table represents an 8-bit character. Reading from left to right, the first bit (bit 0) of the first word represents the null character (000). Likewise, bit 15 of the last word in this table represents the binary character 377. Setting any bit in this table to 1 indicates that the character represented by this bit will be a delimiter in data-sensitive records transmitted over that channel. Figure 6-7 shows a delimiter table with bits set to make null (000), carriage return (015), and rubout (177) data sensitive record delimiters.

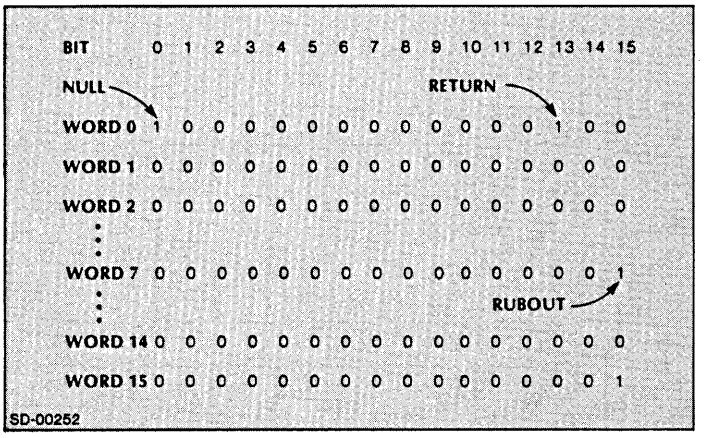


Figure 6-7. Sample Delimiter Table

Fixed length records have a constant length; this length was specified when the file was created (?CCPS), or could be specified (or respecified) at open time (?IRCL), or in the read/write call. *Variable length* records are records of differing lengths whose size is specified by a four-byte ASCII header on each record. AOS writes the header, and it's invisible to you. When writing a variable-length record, the record length is specified as for dynamic I/O.

You can multiply-open any device that is not assigned or exclusively opened. The system assigns a different channel number to the file on the second and subsequent open. You must supply an I/O packet and delimiter table (unless defaulted) for each channel.

The *exclusive open* and *private open* fields permit the caller to specify that he wants no other process or task to access the file. If these fields are not selected, then more than one process or task can open the file concurrently. The exclusive and private open fields are discussed in Chapter 13, under the ?GOPEN system call.

If a caller opens a PMGR file, such as a console, for a *priority read,* and the file has other channels opened on it, all read requests on the priority channel will be placed ahead of requests on the normal channels. This feature may only be selected on ?OPEN.

You can create a user data file at open time by selecting one or both of the file-creation options in offset ?ISTI. The options and their effect are:

?OFCE	Create the file, if it does not exist. If it already exists, ?OFCE is ignored.
?OFCR	Create the file, if it does not exist. If it already exists, an exceptional condition will result.
?OFCE+?OFCR	If the file already exists, delete it, then create the file. (May not be used if opening an IPC-type file.)
omitted	Do not create the file. If it does not exist, an exceptional condition results.

If you attempt to create a device or generic file then control will take the normal return if the generic filename or device already exists in :PER; otherwise control takes the exception return with error ERWAD, write access denied to this directory. Likewise, if you attempt to delete a device name or generic filename and then create and open it (?OFCR+?OFCE), control will take the normal return if the filename exists in :PER, or it will take the exception return if the filename does not exist in :PER. Note that you can specify any valid filename (including names of devices and generic filenames) to be created or deleted/created and opened if you do not specify @ in the pathname prefix (?IFNP). Thus you could create and open a file named MTA0 or LIST in some directory (other than :PER) to which you have write access.

File type occupies the right byte of offset ?ISTO. You set this with one of the byte masks in Table 6-12, or a default of zero. If you are creating the file with this ?OPEN call, and you default the file type, the system will make it a User Data File (?FUDF). If the file was already created before you issued this call, the system will read the existing file type into this field and ignore the contents you provide.

Table 6-13. File Types Which May Be Created with ?OPEN

Mnemonic	Meaning	Comments
?FUDF	User Data File	This is the most common file type, and a zero defaults to this type.
?FTXT	Text File	Should contain ASCII code.
?PRG	Program File	Should contain bound, executable code.
?FDIR	Disk Directory	Note that you can only default the hash frame size, maximum index level, and ACL if you create a directory with ?OPEN.
?FCDP	Control Point Directory	Though AOS will allow you to create a CPD with ?OPEN, there is no utility in doing so. See ?CREATE to create a CPD.
?FIPC	IPC File	Use with ?OFCE in word ?ISTI to create or open an IPC file to allow full communications between exactly two processes. The ?OPEN call will wait until both processes have opened the file unless ?IIPC is set in word ?ISTI.

?OPEN (continued)

The *input/output field* permits the caller to specify whether reading or writing (or both) is to be attempted by the file I/O. Specifying neither reading nor writing will not cause an ?OPEN error, but an error will be signaled if file I/O is attempted.

The *record format field* allows you to change the record format to the type you provide, or default it to the type specified at ?CREATE time (if any). Note that the record format must have been defined by the time a file is read or written.

The *append* field sets the file pointer to the end-of-file; data written to the file will be appended to what is already there.

The meaning of the record length word, ?IRCL, depends upon the record type. For fixed records, this word will specify the length of each record in bytes. For dynamic records and variable-length writes, it indicates the number of bytes which will be transferred upon each I/O request. For data sensitive records, ?IRCL indicates the maximum number of bytes which will be transferred. If a delimiter is encountered before this number, then I/O is terminated.

Word ?IBAD contains a byte pointer to the I/O buffer. This buffer must be large enough to accommodate the largest record which will be read or written, else you will receive a "line too long" error on a record that exceeds the buffer length. If no buffer is specified at open time, then it must be specified with each I/O call.

If you're opening a magnetic tape unit, you can use bits 2 through 4 of word ?IFLG to indicate a "density mode" for the tape unit. The density mode must match the density of the magnetic tape on the unit, else the system will take the exception return for the ?OPEN and return error code ERFTM (File Tape Density Mismatch).

You can choose one of three masks for this word: ?IDAM, which directs the system to select the correct density mode automatically; ?ID8, for a density mode of 800 BPI; and ?ID16, for a density mode of 1600 BPI.

If you default the density bits of ?IFLG, the system uses the density mode chosen by your system manager during system initialization. In this case, you should check to see that the default density mode and your tape's density mode match; otherwise, AOS will return error code ERFTM when you issue the ?OPEN call.

If either the tape unit or the tape itself is damaged, the system will return error code ERITD (Indecipherable Tape Density). If you try to select a density mode for a tape running on a type MTA controller, the system will return error code ERCDN (Controller Does Not Support This Density).

Word ?IMRS contains the size of physical blocks (minus one) which are output to magnetic tape; this word is used in the same way by the IPC (when it is treated as a device), and by the MCA. It is ignored for disk I/O.

To establish an IPC communications link, two processes must open the same IPC file. Control returns to the normal return of each process only after both processes have opened this file, unless the no-wait option is selected. If you set the no-wait bit, your process will take the exception return if the other process has not already opened the IPC file. Note that a process can't use the ?OPEN call to open a communications path to itself. You may, however, use IPC primitive calls to establish an IPC loop. See Chapter 4 for details.

To open a directory for using system call ?GNFN, place a byte pointer to the entry buffer in ?IFNP, place -1 in offsets ?IBAD, ?IRCL, ?IMRS, and ?IDEL, and place a zero in all remaining packet offsets. Only system call ?GNFN should be used with directories; other calls produce unpredictable results.

Labeled Magnetic Tape

AOS can write/read ANSI Level 1, 2, and 3 and IBM Level 1 and 2 labeled tapes. Set the flag ?OANS or ?OIBM in the offset ?IFLG (formerly called ?IRES) of the ?OPEN packet to indicate either ANSI or IBM labeled tapes. AOS generates its own labeled tape format if neither flag is set. All labeled tapes written prior to REV 3.04 use AOS format. Note that selecting IBM labeled tapes does not cause the data to be written in EBCDIC. This is done by selecting the field translation packet on ?READ or ?WRITE.

You can use the ?OPEN call to open both an existing file and a new file on a labeled magnetic tape. However, you cannot select the following options when using labeled tapes.

- Append.
- Both read and write.
- Positioning with a ?READ or ?WRITE.
- System calls ?SPOS and ?GPOS.

The system will return error ERIOO (Illegal Option for Open Type) if you select any of these options.

When writing, AOS uses a few fields in the ?OPEN packet to transmit information to the labels. Because the labels are created at open time, these fields must contain valid information else the tape might be unreadable by another computer.

- Record format is set in offset ?ISTI; the flag ?ICRF must also be set.
- Record length is set in offset ?IRCL.
- Block length "Block Length -1" is set in offset ?IMRS.

Similarly, AOS uses a few fields in the ?OPEN packet to transmit information from the labels when reading. If you ignore this information, the data on the labeled tape may not be read properly.

- Record format is returned in ?ISTI. Do not select ?ICRF if the format value is to be returned.
- Record length is returned in ?IRLR.
- Block length is returned in ?IMRS. This field must contain a -1 if the block length value is to be returned.

If you specify ?ICRF at ?READ or ?WRITE time, the record format specified must not differ from that established at ?OPEN time.

At ?OPEN time, "record length" is the maximum record length in the file. The maximum record length is the length of each record if the file is in fixed-length record format. At ?READ or ?WRITE time, record length is the length of the record being read or written (or maximum record length when writing data-sensitive record format). Undefined and IBM fixed record formats must be written with an even record length. A -1 in ?IRCL specifies the default record length of ?DFLL (136.) bytes.

At ?OPEN time, "block length" is the maximum physical block length in bytes. A -1 in ?IRMS specifies the default block length of 2048. bytes. For IBM fixed record format, the block length must be an integral multiple of the record length. For undefined record formats, the maximum block length is the maximum record length (each record is one block). AOS ignores the value of ?IRMS in a ?READ or ?WRITE packet in place of the block length established at open time.

The system might delete files existing on tape when it is writing the file to labeled tape. The expiration date is the determining factor. If the file has expired (today's date is greater than or equal to the expiration date), then AOS will overwrite it, and thus delete it. Note that this also deletes all files following the overwritten file. It is assumed that the following files have expired if the current file has expired.

Remove the write ring to assure files are not destroyed inadvertantly. The default retention period is 90 days. This assures that the file on a labeled tape cannot be overwritten for 90 days from its creation time, unless a previous file on the tape has an earlier expiration date.

You must mount labeled tapes via EXEC before using them. For more information on how to mount tapes, see the discussion of file I/O using labeled magnetic tapes in this chapter.

Labeled Tape Extended Packet

You can use the labeled tape extended packet to change the field default values in the labels, or to determine the field values in the labels. (For each explanation of the labels used and the fields in each label, see the labeled tape section in this chapter.)

You can choose the extended packet by setting bit ?IPKL in word ?ISTI of the ?OPEN packet, and following the ?OPEN packet by 16 contiguous words. Table 6-2 shows the structure of packet. Set the flag bit and the address of the labeled tape extended packet at offset ?ETLT.

When reading the labels at ?OPEN time, AOS updates the extended packet to contain the information indicated on the labels of the opened file. When writing the labels at ?OPEN time, AOS uses the data in the extended packet to create the labels for the file written. If you want the default value for all fields, do not use the extended packet.

Table 6-14 explains the location and values of the fields in the extended packet. If the file was opened for output, AOS writes the header/trailer labels at ?OPEN/?CLOSE time. If the file is opened for input, AOS reads the user header labels at ?OPEN time. If the file was opened for input and all data was read (e.g., an EOF error occurred on a read), then AOS reads the user trailer labels at ?CLOSE time.

You can have up to nine user header and nine user trailer labels. Each user label is 76. characters long. The byte pointers in offsets ?ELUH and ?ELUT each point to an area containing from one to nine user labels. When writing, the system writes the first 76. bytes of the user label area in the first user label; it writes the second 76. bytes in the second user label, etc. If the system encounters a null (0) as the 76. bytes are prepared for the user label, it will ignore the null and the remainder of the 76. bytes and will fill the user label with spaces. When reading, the system returns the data from the first label in the first 76. bytes of the area, the second label in the second 76. bytes of the area, etc.

Table 6-14. Labeled Magnetic Tape Extended ?OPEN Packet

Offset	Contents	Default	
		With Pkt.	W/O Pkt.
?ELVL	Volume identifier (always returned) taken from the VOL1 label (6 bytes).	n/a	n/a
?ELGN	Generation number of the file (from 0 to 9999). -1 means do not check or return the generation number if reading; use the default if writing.	0001	
?ELVR	Generation version number of the file (from 0 to 99). -1 means do not check or return the version number if reading; use the default if writing.	00	
?ELCR	Creation date in standard system format. 0 means use default.	Current date	
?ELRE	Retention period in days.	n/a	90 days
?ELCT	Right byte: number of user header labels.	n/a	No user labels
	Left byte: number of user trailer labels.	n/a	
	If ?ELEP is set in this word it means the packet contains offsets ?ELAC, ?ELFS, ?ELR1, ?ELR2.		
?ELUH	Byte pointer to area containing user header labels.	n/a	
?ELAC	Left byte: level of labeling; values ?ELL1 level 1 (ANSI or IBM) ?ELL2 level 2 (ANSI or IBM) ?ELL3 (ANSI) 0 means use default Right byte: access byte. This byte is written as the access byte; 0 means use default.	If ANSI format, ANSI level 3 used. If IBM format, IBM level 2 used. If ANSI format " ". If IBM format "0".	
?ELFS	Byte pointer to file set ID. If 0 use default. If -1 ignore file set ID when reading; use default when writing.	VOLID of 1st volume in file set.	
?ELR1	Reserved word, must be 0.		
?ELR2	Reserved word, must be 0.		

?RDUDA/?WRUDA

Read/write a User Data Area (UDA).

?RDUDA or ?WRUDA
exception return
normal return

Input/Output

Input:

AC0 byte pointer to pathname of file.

AC2 word address of a 128-word buffer area.

Output:

AC0 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERNUD UDA does not exist.

FILE SYSTEM codes

Description

Use ?RDUDA or ?WRUDA to read from or write into a UDA. You must first create a UDA with a ?CRUDA call. You can determine whether or not a file has an associated UDA by examining bit ?FUDA in offset ?SSTS of the file status packet (?FSTAT).

The first word of a UDA is always a file type code for the file you want to pass to a system utility. Data General reserves codes where bit 0 equals zero; codes in which bit 0 is set to 1 are available to users.

?READ/?WRITE

Perform record I/O.

?READ or ?WRITE *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 address of parameter packet, if not in line.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERITP Illegal translation parameter (extended packet only).

FILE SYSTEM codes

SYSTEM CALL codes

CHANNEL-RELATED codes

Description

After a file has been opened on a channel, record I/O may be performed on the file with the ?READ or ?WRITE calls. Certain information specified when the file was created or opened can be either defaulted to earlier values or changed when record I/O is performed. This information includes the buffer address, record type, record length, and record number.

Parameters in the File Specifications Word, ?ISTI, represent masks, not bit positions. Thus to select more than one option (e.g., format select and binary output), sum the masks in this word (e.g., ?ICRF+?IBIN). Unused bits in ?ISTI must be set to zero.

The structure and contents of the parameter packet are shown in Figure 6-8 and described in Table 6-15.

?READ/?WRITE (continued)

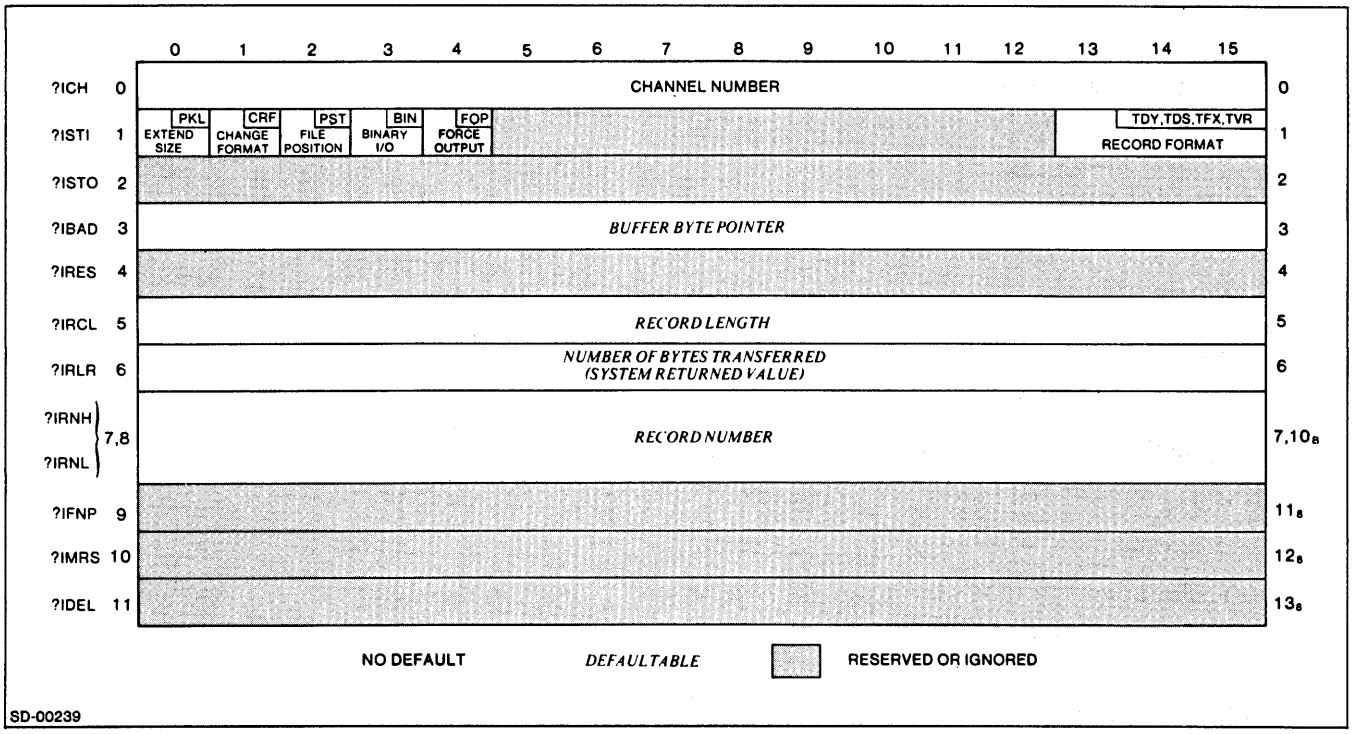


Figure 6-8. ?READ/?WRITE Parameter Packet

Table 6-15. ?READ/?WRITE Parameter Packet

Offset	Contents	Default
?ICH	Channel number.	No default
?ISTI	File Specifications Word	
	Packet type ?IPKL - this parameter selects extended processing.	(0) Only I/O features in the initial packet may be selected.
	Format select ?ICRF - change format to that in record format field.	(0) The record format specified at ?OPEN is used.
	Absolute file pointer positioning ?IPST - ?IRNH/?IRNL is an absolute record number for fixed records, an absolute byte offset for other record types.	(0) ?IRNH/?IRNL is interpreted as a relative record offset for fixed records, relative byte offset for other types.
	Append (used only by ?OPEN).	n/a
	Binary I/O ?IBIN - character device I/O is in binary mode.	(0) Character I/O proceeds in text mode.
	Force output ?IFOP - write out buffer from dynamic system area to device.	(0) Normal system buffering occurs.
	Exclusive open (used only by ?OPEN).	n/a
	IPC primary (used only by ?OPEN).	n/a
	Creation option (used only by ?OPEN).	n/a
	Input/output (used only by ?OPEN).	
	Record format ?RTDY - dynamic (see ?IRCL).	none
	?RTDS - data sensitive (see ?IDEL and ?IRCL).	none
	?RTFX - fixed length (see ?IRCL).	none
	?RTVR - variable length.	none
	?RTUN - undefined record format (read physical blocks).	none
?ISTO	File type (used only by ?OPEN).	n/a
?IBAD	Byte pointer to record I/O buffer.	(-1) Use I/O buffer specified at ?OPEN time.
?IFLG	Used by ?OPEN.	n/a
?IRCL	Record length. Number of bytes to read or write for dynamic and fixed record formats; maximum number of bytes to read or write for data sensitive and variable length record formats.	(-1) Use record length specified at ?OPEN time.

(continues)

Table 6-15. ?READ/?WRITE Parameter Packet

Offset	Contents	Default
?IRLR	Actual number of bytes transferred.	(Returned by the system.)
?IRNH/ ?IRNL	These fields are the high-order and low-order portions of a double-precision record number. If you select ?IPST and ?RTFX in offset ?ISTI of this parameter packet, ?IRNH/?IRNL is the absolute record number. If you select ?RTFX but not ?IPST, ?IRNH/?IRNL is a record offset relative to the current file pointer position. For all other record types (?RTDY, ?RTDS, and ?RTVR), if you select ?IPST, ?IRNH/?IRNL is the absolute byte count from the beginning of the file, and if you do not select ?IPST, ?IRNH/?IRNL is a byte offset relative to the current file pointer position. Note that you can reposition the file pointer without data transfer by setting ?IPST in offset ?ISTI and both ?IRNH/?IRNL to 0 for the beginning of the file or to -1 for the end of the file.	Next sequential record, if ?IPST is not selected. If ?IPST is selected, there is no default value.
?IFNP	(used only by ?OPEN).	n/a
?IMRS	(used only by ?OPEN).	n/a
?IDEL	(used only by ?OPEN).	n/a

(concluded)

If you select ?ICRF to change the format to that specified in the record format field, then the new format persists until either the channel is closed or you specify ?ICRF again. Note that one of the following cases must be true:

1. The record format was specified on ?CREATE and defaulted or changed on ?OPEN and ?READ/?WRITE; or
2. The record format was postponed on ?CREATE, then specified on ?OPEN and defaulted or changed on ?READ/?WRITE; or
3. The record format was postponed on ?CREATE and ?OPEN, and specified on ?READ/?WRITE.

If you do none of the above then you have failed to provide a record format, and the system will return an error on ?READ/?WRITE.

Also note that the format ?RTUN (undefined record format) may only be used on a ?READ call. You will need this format to read unlabeled mag tapes of unknown block size; it causes the system to read the actual physical blocks.

Before each read or write, the file pointer indicates the next record; the pointer may be moved to any position in the file. Values in offsets ?IRNH/?IRNL and ?IPST position the file pointer as follows:

?IPST	?IRNH/ ?IRNL	File Pointer Position
0	0	At the next record.
0	n (n is any integer)	At the relative record offset n (forward or backward from the current file pointer position) for fixed records, or at relative byte offset n for all other record types.
1	0	At the beginning of the file.
1	-1	At the end of the file.
1	n (n is any integer)	At the nth byte or record in the file, depending on record type. The first byte or record is number 0.

In essence, setting ?IPST to 0 makes ?IRNH/?IRNL a relative value, while setting ?IPST to 1 makes ?IRNH/?IRNL an absolute value.

For console devices, flag ?IFOP resets the CTRL-O flag and writes this record to the device. For disk, magnetic tape, MCA, and IPC devices, the current buffer is forced to be written out from the dynamic system area to the device.

If you are reading data from a character device and binary mode input is on, the system will ignore rubout and all control sequences, and pass all data in uninterpreted bytes. You exit from binary mode input by typing BREAK on a terminal, communicating via an ALM-8 or -16 communication line, issuing a non-binary mode ?READ, or terminating the process. Note that none of this affects binary mode output on a ?WRITE call.

Screen Management Primitives

This feature allows you to access certain screen management primitives normally controlled only by the system program PMGR (Peripheral Manager Process). On a ?READ, this feature allows you to display an initial string, to ignore typed-ahead characters, and to use control characters to delete, insert, and overwrite characters within the line being input. (These control characters are the same as those defined for the SED utility, documented in the *AOS/VS SED Text Editor User's Manual*). Additionally, on both ?READ and ?WRITE, this feature allows you to suppress the echoing of delimiters and to specify an initial cursor position (in column and row number format) where the cursor will be moved before the I/O takes place.

To use this feature, you must set up an extended parameter packet for setting a ?READ or ?WRITE call. This is explained earlier in this chapter. Briefly, you must:

- Set bit ?IPKL of offset ?ISTI in the normal packet.
- Supply a 16-word address block following that packet.
- In word ?ETSP of the address block, supply the address of the extended packet in bits 1-15, and use bit zero as a flag bit. (You must set this bit to one to get the packet read. After each extended processing call, the system resets this bit to zero.)
- In the location pointed to by word ?ETSP, supply the screen management primitives extended packet.

The structure of the screen management primitives extended packet is given in Table 6-16.

Table 6-16. Screen Management Primitives Extended Packet

Offset	Contents
?ESFC	Flag word, with any of these bit masks: ?ESSE — Screen edit; i.e., enable SED control characters. ?ESRD — Redisplay previously read data (?READ only). This is valid only when ?ESSE is also set. ?ESNR — Drop typed ahead characters; i.e., flush the ring buffer before beginning input (?READ only). ?ESED — Do not echo delimiters. ?ESCP — Initial cursor position specified in offset ?ESCR of this packet.This is valid only when ?ESSE is also set. ?ESDD — If this bit is on after a ?READ, the ?READ was terminated in a double character delimiter (from function keys). ?ESRP — If this bit is on, the cursor position will be returned in offset ?ESCR. If the last ?READ did not specify an initial cursor position (?ESCP clear), then the cursor position returned will be relative to the cursor position before the last ?READ was performed. If the last ?READ did specify initial cursor position (?ESCP set), then the cursor position returned is an abolute position on the screen. The cursor position returned never has a row value greater than LPP - 1 (lines per page minus one) or a column value greater than CPL (columns per line).
?ESEP	If masks ?ESSE and ?ESRD are set in flag word ?ESFC, then this word will be used as the relative position within the data at which the cursor will be left. The first character is position zero.
?ESCR	The relative column (left byte) and row (right byte) to position the cursor to before doing any I/O. The first column of the top row on the console screen is column 0, row 0. Note that this feature will not function correctly unless the characteristics for the device are also correctly set. The absolute cursor position will be returned in this word if ?ESRP is set in word ?ESFC.

Selected Field Translation

This feature allows you to translate data in various ways as you perform a ?READ or ?WRITE. Note that, since the translation tables are not bidirectional, you must redefine translation type before reversing I/O direction. For example, if you are reading an EBCDIC file, translating it to ASCII as you go, and you want to write to the file from the I/O buffer and have it go back in as EBCDIC, you must replace ?EBAS with ?ASEB in ?EFTY, the translation type field. The translation is done in the order you specify and is cumulative. You may specify fields in any order, and they need not be mutually exclusive.

You accomplish field translation with an extended parameter packet. The process of setting up such a packet is explained earlier in this chapter. Briefly, you must:

- Set bit ?IPKL of offset ?ISTI in the normal packet and supply a contiguous 16-word address block following that packet.
- In word ?ETFT of the address block, supply the address of the extended packet in bits 1-15, and use bit zero as a flag bit; you must set this bit to one to get the extended packet read. After each extended processing call, the system resets this bit to zero.
- In the location pointed to by word ?ETFT, supply the selected field translation extended packet.

The selected field translation packet consists of a header word and one or more subpackets. Each subpacket comprises three words: ?EFFP, ?EFLN, and ?EFTY. The right byte of the header word contains the value N--the number of translations to be performed each time a ?READ or ?WRITE is called. The header word must be followed by N subpackets, each describing the Nth translation to be done. The fields described by the subpackets may be in any order, and need not be mutually exclusive.

AOS performs translations in the same order as the subpackets occur. Thus, when a field is translated from EBCDIC to ASCII and then odd parity added, the first subpacket should contain information for EBCDIC to ASCII translation, and the second should specify the same field for odd parity addition. Note that the translation tables are not bidirectional. This means your program must change the value in ?EFTY of the subpackets each time the direction of the I/O is changed. For example, if you are translating a file from EBCDIC to ASCII as you read it, ?EBAS should be placed in ?EFTY. Before writing the same record out in EBCDIC, you should set ?EFTY to ?ASEB.

The structure of the selected field translation extended packet is given in Table 6-17.

Table 6-17. Selected Field Translation Extended Packet

Offset	Contents	Default
?EFNF	Right byte: Number of fields to be translated.	n/a
	Left byte: (returned) Actual number of fields translated.	n/a
?EFFP	Byte offset to field.	n/a
?EFLN	Field length.	(0) Remainder of record.
?EFTY	Translation type:	n/a
	?ASEB - ASCII to EBCDIC.	
	?EBAS - EBCDIC to ASCII.	
	?ALAU - ASCII lowercase to ASCII uppercase.	
	?AUAL - ASCII uppercase to ASCII lowercase.	
	?SPAR - Strip parity bits.	
	?AEPR - Add even parity.	
	?AOPR - Add odd parity.	
?EFMAX	Length of packet.	3
	Or, instead of the above masks, a byte pointer to a 256-byte translation table.	n/a

Line Printer Format Control

You may control the horizontal and vertical format of any device driven by the EXEC spooler. If the device is not a data channel line printer, EXEC will simulate the vertical format by outputting multiple NEW LINE characters as needed. The EXEC gets format information for a spooled device as follows:

1. If the file was printed with a special form specified, that form entry in directory :UTIL:FORMS must have a User Data Area (UDA) that contains form control information. If the form entry does not have a UDA with form control information, the system will print an error message instead of the file. Otherwise, the form entry's form control information will be used instead of EXEC's current values (i.e., @EXEC LPP and @EXEC CPL) for the system form parameters.

2. If the file was printed without a special form specified, EXEC's current values (i.e., CTRL @EXEC LPP and CTRL @EXEC CPL) will be used for the system form parameters.

3. You have the ability to override the system form parameters (i.e., /FORMS or @EXEC's, whichever are in use) by specifying form control parameters in the UDA of the file you want printed. To do this, however, the form length must exactly match the system form length, and the line length must not be longer than the system line length.

You can create, edit, and copy form control UDA's with the system utility program FCU (Forms Control Utility), documented in the *Command Line Interpreter (CLI) (AOS and AOS/VS) User's Manual.*

The following examples illustrate some possible applications of the ?READ/?WRITE calls. Table 6-18 indicates parameters and values which would be selected to produce the various examples.

Table 6-18. Examples of ?READ/?WRITE Operations

Ex. No.	Format Select	Record Pos.	Binary Output	Flush Buffer	Record Format				I/O Buffer	Record Length	Record No.	
	?ICRF	?IPST	?IBIN	?IFOP	?RTDY	?RTDS	?RTFX	?RTVR	?IBAD	?IRCL	?IRNH	?IRNL
1		X							-1	m	n	n
2									-1	m	n	n
3	X				X				-1	m	n	n
4	X				X				-1	m		
5	X						X		-1	m		
6							X		-1	m		
7	X					X			-1	m		
8	X							X	-1	m		

In example 1, after positioning the file pointer to record or byte n, m bytes (or a record m bytes long) are transferred to a buffer specified to ?OPEN.

Example 2 spaces forward (n > 0) or backward (n < 0) n records/bytes, and then transfers m bytes to/from the buffer specified to ?OPEN.

Example 3 spaces forward n bytes, reads or writes the next m bytes and transfers them to the buffer specified to ?OPEN. Record format specified in ?OPEN is overridden by ?ICRF and ?RTDY.

Example 4 reads or writes m bytes starting at the file pointer, regardless of the record format specified to ?OPEN. The file pointer is positioned after these bytes; dynamic record format persists.

Example 5 changes record format to fixed and reads or writes the next fixed length record, and positions the file pointer after this record. The fixed length record is m bytes long.

Example 6 is equivalent to example 7 if the file was opened with ?RTFX specified. It is likewise equivalent to example 7 if no record format was specified to ?OPEN. If ?OPEN specified a record format other than ?RTFX, however, then that record type prevails as the record is read or written.

Example 7 changes record format to data sensitive and reads or writes the next data sensitive record; no more than m bytes will be transferred, and the record is delimited by any character(s) specified to ?OPEN in the delimiter table. If no table was specified at ?OPEN, the default delimiters are selected.

Example 8 reads or writes the next record. Upon a write, a record header specifying length m, followed by m bytes of data, is written. Upon a read, this record is read. In both cases, the record type is changed to variable.

?SCHR

Set the device characteristics.

?SCHR
exception return
normal return

Input/Output

Input:

AC0	byte pointer to name of device or channel number.
AC1	bit 0=1, AC0 contains a channel number.
	bit 0=0, AC0 contains a byte pointer.
	bit 1=0, change characteristics
	bit 1=1, define default characteristics for the specified device.
AC2	address of three-word characteristics specification.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

INITIALIZATION AND RELEASE codes

Description

This call sets the three characteristics words associated with any character device. These words are illustrated under ?GCHR in this chapter.

If bit 1 of AC1 is set to 0 on input, AOS copies the three-word buffer you point to in AC0 to the associated device. If bit 1 of AC1 is set to 1, the three-word characteristic you point to in AC0 will override the bit setting specified at AOSGEN time. Only the operator process (PID 2) can define new default device characteristics.

?SDLM

Set a delimiter table.

?SDLM
exception return
normal return

Input/Output

Input:

AC0	byte pointer to name of device or channel number.
AC1	masks:
	?SDDN AC0 contains a byte pointer to device name.
	?SDCN AC0 contains a channel number.
	?SDTO Set delimiter table for output.
	?SDTI Set delimiter table for input.
	?SDTP Set delimiter table for priority read.
AC2	Word address of the 16. word delimiter table.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERICN	Illegal channel.
ERIFT	Illegal file type.
ERMPR	System call parameter address error.
ERPRE	Invalid system call parameter.

Description

This system call sets a delimiter table for a file with data-sensitive records, or for a character device, while the file or device is open. This call is an alternate to the delimiter table feature in the ?OPEN packet. If you define a delimiter table in the ?OPEN packet and then issue ?SDLM, the new delimiter table overrides the one defined in the ?OPEN packet. Thus, you could use this call to redefine the delimiter table you set up at ?OPEN time.

If you're issuing ?SDLM for a character device, such as a console, you can define one of three types of delimiter tables in your input to AC1. For an output delimiter table (a table defining delimiters for output to the device), specify mask ?SDTO in AC1. For an input delimiter table (defining delimiters for input to the device), specify mask ?SDTI in AC1. If you've opened the device, specify mask ?SDTI in AC1. If you've opened the device with the priority read option, specify ?SDTP in AC1 to designate the table as a priority input delimiter table.

?SEND

Send a message to a console.

?SEND
exception return
normal return

Input/Output

Input:

AC0 byte pointer to name of console receiving the message; PID of process whose console is receiving the message; or byte pointer to name of process whose console is receiving the message.

AC1 byte pointer to message which will be sent.

AC2 bits 8-15, byte length of message.

bits 6-7:

0 AC0 contains PID.

1 AC0 contains byte pointer to process name.

2 AC0 contains byte pointer to console name.

3 (undefined).

Output:

AC0 unchanged.

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERNAC Named device is not a console device.

ERMRD Device has characteristics ?CNRM set (i.e., reception of ?SEND messages has been disabled).

ERPRH Attempt to refer to a process that is not in the process tree.

FILE SYSTEM codes

Description

This call outputs a message on a console if that console has message receipt enabled (i.e., ?CNRM of device characteristics equals zero). The maximum length of the message is 255 bytes.

The message is output with a header:

<NEW LINE>
FROM PID xxx: message

xxx is the PID of the process issuing the ?SEND call.

Note that the name of a process is of the form username:process-name.

?SPOS

Set the file pointer position.

?SPOS *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 I/O parameter packet address, if not passed in line.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This system call repositions the file pointer associated with the channel named in the I/O parameter packet; no I/O is performed. Offsets ?IRNH/?IRNL, ?IPST and ?IRCL in the packet must be set appropriately to effect the pointer position change. ?IBAD is ignored by this call.

Offset ?IRCL indicates record length. If the record type is fixed length, you should set this offset to the record length, or to -1, if the record length was previously defined. If the record type is other than fixed, you must set ?IRCL to zero. Setting ?IPST to 0 makes ?IRNH/?IRNL a relative value, while setting ?IPST to 1 makes ?IRNH/?IRNL an absolute value. Thus these parameters are handled as they are by the ?READ/?WRITE calls:

?IPST	?IRNH/?IRNL	File Pointer Position
0	0	At the next record.
0	n (n is any integer)	At the relative record offset n (forward or backward from the current file pointer position) for fixed records, or at relative byte offset n for all other record types.
1	0	At the beginning of the file.
1	-1	At the end of the file.
1	n (n is any integer)	At the nth byte or record in the file, depending on record type. The first byte or record is number 0.

?STOM

Set time out value.

?STOM
exception return
normal return

Input/Output

Input:

AC0	byte pointer to name of device or channel number.
AC1	bit 0=0, AC0 contains a byte pointer.
	bit 0=1, AC0 contains a channel number.
	rest of word is reserved and should be set to 0.
AC2	timeout value in seconds.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This call specifies how long the system should wait before giving the ERDTO (Device Time Out) error message. The purpose of the time out is to recognize when a device has not responded to I/O within a specified period of time. For example, if paper tape is being punched and a character is sent to the punch, the punch should respond within a few seconds that it is ready for another character. If it does not respond, the punch has timed out.

The shortest time out value allowed is 2 seconds. If a smaller value is received, the time out value will be set to 2. The system uses the value for both input and output, if the device allows both directions of communication.

Once a caller issues a ?STOM call, you (or your program) should issue a set characteristics call (?SCHR) to change the characteristic that checks for time out (?CTO). If the time out value is set to a -1, the default time out will be used. A list of devices and their default time out values follows.

Device Type	Default Value (seconds)
all consoles	-1 (means time out value isn't checked)
PTR	2
CDR	2
PTP	3
LPA	30
PLT	2
LPC	30

?TRUNCATE

Truncate a file at current position.

?TRUNCATE
exception return
normal return

Input/Output

Input:

AC1 channel number on which the file was ?OPENed.

Output:

AC1 unchanged.

Exceptional Condition Codes in AC0

ERIFT Illegal file type.

FILE SYSTEM codes

Description

The ?TRUNCATE system call truncates disk or magnetic tape files at the current file pointer position. File types other than disk or unlabeled magnetic tape are illegal. Truncating @NULL produces no action and takes the normal return. You can truncate magnetic tape files by writing two end-of-file marks at the current position.

?UPDATE

Flush file descriptor information.

?UPDATE
exception return
normal return

Input/Output

Input:

AC1 bit zero: reserved, must be zero.

bits 1-15: channel number.

Output:

AC1 unchanged.

Exceptional Codition Codes in AC0

FILE SYSTEM codes

Description

This call forces any file descriptor information that AOS may be keeping in main memory out to disk. ?UPDATE has the same effect as ?GCLOSE followed by ?GOPEN, but without the associated overhead of those calls. The ?UPDATE call guarantees file integrity for all writes dones up to the point of completion of the ?UPDATE call. The ?UPDATE call is useful to any application which can afford to lost little or no file data in the case of a system crash.

The file that you are issuing ?UPDATE to must have a file type between the values ?DMIN and ?UMAX, which includes all DGC and user file types. The system ignores attempts to ?UPDATE a system file type between ?DMIN and ?UMAX and, in this case, always takes the normal return.

When using the ?UPDATE call, note that no file data is written to the disk by this call. You are responsible for forcing this data out. If the data is being written with ?READ/?WRITE calls, then you must set bit ?IFOP in the packet to force output, If the file is shared, the ?RPAGE calls must be used with the flush bit set.

Examples

This section illustrates common I/O calls in two separate programs. Figure 6-9 shows the first of these, WRITE.

This program outputs lines you type on the console to the line printer, @LPT. Since WRITE needs to know the name of the console you are using, you type the command:

```
XEQ WRITE console name
```

to activate WRITE. WRITE then issues a ?GTMES call (described in Chapter 9) to read the name of the console you are using. After activating WRITE, you proceed to type lines of text, following each with a NEW LINE.

To terminate WRITE, enter a line containing nothing but a number sign (#). The lines you typed on the console are not output to the line printer until the file is closed, since the line printer is spooled. For a more detailed description of WRITE's operation consult *Learning to Use Your Advanced Operating System.*

DLIST, shown in Figure 6-10, outputs to @LPT the names of all files in a specified directory.

To run DLIST, type

```
XEQ DLIST directory pathname
```

DLIST issues ?GTMES to read the directory pathname, opens the directory and line printer, and then issues ?GNFN to read the filenames. After reading each one, DLIST outputs the filename to the line printer.

Between each name, DLIST outputs a NEW LINE character to the printer.

When control finally takes the ?GNFN exception return, DLIST checks to see whether AC0 indicates that the last filename entry has been read or whether an error condition has arisen. If there was a bona fide error, control goes to ERROR (described in Chapter 2). Otherwise, DLIST simply sets the good return flag and returns to the CLI.

```
 0001  WRITE     AOS ASSEMBLER REV 01.08          17:49:13 12/08/78
01                ;WRITE AOS ASSEMBLER
                                      .TITL   WRITE
03                                    .ENT    START
04                                    .NREL
05                              START:  ?GTMES  GTMPK ;GET ORIGINAL CLI COMMAND
06 00004'000447                       JMP     ERTN  ;EXCEPTION RETURN
07                                    ?OPEN   CONSOLE ;OPEN CON.FOR READING
08 00011'000442                       JMP     ERTN  ;REPORT ERROR
09                                    ?OPEN   LINPT ;OPEN LNE.PTR.FOR WRITING
10 00016'000435                       JMP     ERTN
11
12                              CONT:   ?READ   CONSOLE  ;READ A RECORD
13 00023'000430                       JMP     ERTN  ;REPORT ERROR
14 00024'020434                       LDA     0,TERM  ;CHECK FOR "#"
15 00025'024545                       LDA     1,CONSOLE+3 ;GET BYTE PNTR.
16 00026'126710                       LDB     1,1   ;GET 1ST.CHAR.
17 00027'106415                       SUB #   0,1,SNR ;SKIP IF 1ST CHAR NOT #
18 00030'000407                       JMP     CLOSE ;#, CLOSE DEVICES
19                                    ?WRITE  LINPT  ;WRITE RECORD TO PRNTR.
20 00035'000416                       JMP     ERTN  ;REPORT ERROR
21 00036'000761                       JMP     CONT ;READ ANOTHER RECORD
22
23                              CLOSE:  ?CLOSE  CONSOLE ; CLOSE CON.
24 00043'000410                       JMP     ERTN    ;REPORT ERROR

25                                    ?CLOSE  LINPT  ;CLOSE LINE PRNTR.
26 00050'000403                       JMP     ERTN  ;REPORT ERROR
27 00051'152620                       SUBZR   2,2  ;SET GOOD RTN FLAG
28 00052'000402                       JMP     .+2  ;SKIP ERROR FLG.
29
30 00053'030404                 ERTN:   LDA     2,FLAGS ;GET ERROR FLGS.
31                                    ?RETURN       ;RETURN TO CLI
32 00056'000775                       JMP     ERTN  ;TRY TO REPORT ?RETURN ER.
33
34 00057'150000                 FLAGS:  ?RFEC+?RFCF+?RFER ;RETURN FLAGS
35 00060'000043                 TERM:   043     ;ASCII CODE FOR "#"
36 00061'000074                 IBUF:   .BLK    60.  ;120.CHARS.MAX.
37                        ;PARAMETER PACKETS
38                        ;?GTMES PACKET
39 00155'000004                 GTMPK:  .BLK    ?GTLN
40       000155'                        .LOC    GTMPK+?GREQ
41 00155'000003                         ?GARG
42       000156'                        .LOC    GTMPK+?GNUM
43 00156'000001                         1                     ;GET CONSOLE NM.
44       000157'                        .LOC    GTMPK+?GSW
45 00157'000000                         0
46       000160'                        .LOC    GTMPK+?GRES
47 00160'000342"                        CON*2    ;BYTE PNT. TO ADRS.OF CON. NM.
48       000161'                        .LOC    GTMPK+?GTLN ;PACKET LENGTH
49                        ;END OF ?GTMES PACKET
50 00161'000006                 CON:    .BLK    6; AREA TO RECEIVE CON.NM.
51
52                        ;READ PACKET
53 00167'000014                 CONSOLE:.BLK    ?IBLT
54       000167'                        .LOC     CONSOLE+?ICH
55 00167'000000                         0
56       000170'                        .LOC     CONSOLE+?ISTI
57 00170'040022                         ?ICRF+?RTDS+?OFIN ;DATA SENS.READS, INPT
58       000171'                        .LOC     CONSOLE+?ISTO
59 00171'000000                         0
```

Figure 6-9. WRITE (continues)

```
10002  WRITE
01
02                              ;WRITE
03                              ;READ PACKET CONTINUED
04          000172'                         .LOC    CONSOLE+?IBAD
05  00172'000142"                           IBUF*2  ;BYTE PNTR.TO READ BUFFER
06          000173'                         .LOC    CONSOLE+?IRES
07  00173'000000                            0
08          000174'                         .LOC    CONSOLE+?IRCL
09  00174'000170                            120.    ;120.CHARS./LINE MAX.
10          000175'                         .LOC    CONSOLE+?IRLR
11  00175'000000                            0
12          000177'                         .LOC    CONSOLE+?IRNL
13  00177'000000                            0
14          000200'                         .LOC    CONSOLE+?IFNP
15  00200'000342"                           CON*2
16          000201'                         .LOC    CONSOLE+?IMRS
17  00201'177777                            -1
18          000202'                         .LOC    CONSOLE+?IDEL
19  00202'177777                            -1
20          000203'                         .LOC    CONSOLE+?IBLT ;PACKET LENGTH
21                              ;END OF READ PACKET
22                              ;START OF WRITE PACKET
23  00203'000014                      LINPT:  .BLK    ?IBLT
24          000203'                         .LOC    LINPT+?ICH
25  00203'000000                            0
26          000204'                         .LOC    LINPT+?ISTI
27  00204'040012                            ?ICRF+?RTDS+?OFOT ;DATA SENS.,FOR OUTPT
28          000205'                         .LOC    LINPT+?ISTO
29  00205'000000                            0
30          000206'                         .LOC    LINPT+?IBAD
31  00206'000142"                           IBUF*2  ;BYTE PNTR.TO RECORD BUFFR.
32          000207'                         .LOC    LINPT+?IRES
33  00207'000000                            0
34          000210'                         .LOC    LINPT+?IRCL
35  00210'000170                            120.    ;WRITE 120.CHARS.MAX.
36          000211'                         .LOC    LINPT+?IRLR
37  00211'000000                            0
38          000213'                         .LOC    LINPT+?IRNL
39  00213'000000                            0
40          000214'                         .LOC    LINPT+?IFNP
41  00214'000436"                           LINP*2
42          000215'                         .LOC    LINPT+?IMRS
43  00215'177777                            -1
44          000216'                         .LOC    LINPT+?IDEL
45  00216'177777                            -1
46          000217'                         .LOC    LINPT+?IBLT ;PACKET LENGTH
47                              ;END OF WRITE PACKET
48  00217'040114                      LINP:   .TXT    "@LPT"
49          050124
50          000000
51                                          .END    START

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS
```

Figure 6-9. WRITE (continued)

```
  0003 WRITE

CLOSE 000037'        1/18    1/23
CON   000161'        1/47    1/50    2/15
CONSO 000167'        1/08    1/13    1/15    1/24    1/53    1/54    1/56
                     1/58    2/04    2/06    2/08    2/10    2/12    2/14
                     2/16    2/18    2/20
CONT  000017'        1/12    1/21
ERTN  000053'        1/06    1/08    1/10    1/13    1/20    1/24    1/26
                     1/30    1/32
FLAGS 000057'        1/30    1/34
GTMPK 000155'        1/06    1/39    1/40    1/42    1/44    1/46    1/48
IBUF  000061'        1/36    2/05    2/31
LINP  000217'        2/41    2/48
LINPT 000203'        1/10    1/20    1/26    2/23    2/24    2/26    2/28
                     2/30    2/32    2/34    2/36    2/38    2/40    2/42
                     2/44    2/46
START 000000' EN     1/03    1/05    2/51
TERM  000060'        1/14    1/35
?CLOS 004350  MC     1/23    1/25
?GTME 004545  MC     1/05
?OPEN 004315  MC     1/07    1/09
?READ 004403  MC     1/12
?RETU 004600  MC     1/31
?WRIT 004436  MC     1/19
?XCAL 000001         1/06    1/08    1/10    1/13    1/20    1/24    1/26
                     1/32
```

Figure 6-9. WRITE (concluded)

```
 0001  DLIST    AOS ASSEMBLER REV 01.08          15:14:49 12/04/78
01               ;001 DLIST AOS ASSEMBLER
                                .TITL  DLIST
03                              .EXTN  ERROR
04
05                           ;THIS PROGRAM LISTS THE CONTENTS OF A DIRECTORY ON
06                           ;THE LINE PRINTER. TO INVOKE THE PROGRAM, TYPE
07                           ;"DLIST 'DIRECTORY-NAME' (NEW LINE)".
08                              .NREL
09                           LIST:   ?GTMES MSGPK ;GET DIR.NAME FROM COMMAND LN.
10 00004'002446                      JMP @.ERROR
11                                   ?OPEN DIR      ;OPEN DIR. FOR ENTRY LISTING
12 00011'002441                      JMP @.ERROR
13                                   ?OPEN LINPT    ;OPEN LINE PTR. FOR LISTING
14 00016'002434                      JMP @.ERROR
15 00017'024443                    CONT: LDA 1, DIR     ;GET THE DIRECTORY CHANNEL
16 00020'030433                      LDA 2,.GFNPK   ;GET THE GNFN PACKET
17                                   ?GNFN
18 00023'000420                      JMP EOFTST     ;END OF LIST OR AN ERROR?
19                                   ?WRITE LINPT   ;WRITE ENTRY NAME TO LPT
20 00030'002422                      JMP @.ERROR
21 00031'020464                      LDA 0,.EOL
22 00032'040447                      STA 0,LINPT+3  ;OUTPUT NEW-LINE CHAR.
23                                   ?WRITE LINPT
24 00037'002413                      JMP @.ERROR
25 00040'020415                      LDA 0,.ENTRY   ;RESTORE ENTRY BUFFER
26 00041'040440                      STA 0,LINPT+3
27 00042'000755                      JMP CONT
28
29 00043'024411                 EOFTST: LDA 1,.EREOF
30 00044'106414                      SUB# 0,1,SZR
31 00045'002405                      JMP @.ERROR    ;NOT AN END-OF-FILE
32 00046'152620                      SUBZR 2,2   ;END-OF-FILE CONDITION,GOOD RTN
33                                   ?RETURN
34 00051'002401                      JMP @.ERROR
35
36 00052'000000$                .ERROR: ERROR
37 00053'000117'                .GFNPK: GFNPK
38 00054'000030                 .EREOF: EREOF          ;SYSTEM END-OF-FILE CODE
39 00055'000244"                .ENTRY: ENTRY*2
40
41                           ;GET-MESSAGE PARAMETER PACKET
42 00056'000004                 MSGPK:  .BLK ?GTLN
43       000056'                         .LOC MSGPK+?GREQ
44 00056'000003                          ?GARG
45       000057'                         .LOC MSGPK+?GNUM
46 00057'000001                          1                ;GET DIRECTORY NAME
47       000061'                         .LOC MSGPK+?GRES ;BYTE PNTR.TO DIRECTORY
48 00061'000654"                         DIRNM*2          ;NAME BUFFER
49       000062'                        .LOC MSGPK+?GTLN
50
```

Figure 6-10. DLIST (continues)

```
10002  DLIST
01
02                ;0002 DLIST
03
04                              ;PACKET TO OPEN DIRECTORY FOR ?GNFN
05 00062'000014                 DIR:    .BLK ?IBLT
06       000062'                        .LOC DIR+?ICH
07 00062'000000                         0
08       000063'                        .LOC DIR+?ISTI
09 00063'000000                         0
10       000064'                        .LOC DIR+?ISTO
11 00064'000000                         0
12       000065'                        .LOC DIR+?IBAD
13 00065'177777                         -1
14       000066'                        .LOC DIR+?IRES
15 00066'000000                         0
16       000067'                        .LOC DIR+?IRCL
17 00067'177777                         -1
18       000070'                        .LOC DIR+?IRLR
19 00070'000000                         0
20       000071'                        .LOC DIR+?IRNH
21 00071'000000                         0
22       000072'                        .LOC DIR+?IRNL
23 00072'000000                         0
24       000073'                        .LOC DIR+?IFNP   ;BYTE PNTR. TO
25 00073'000654"                        DIRNM*2           ;OPENED FILE
26       000074'                        .LOC DIR+?IMRS   ;DEFAULT BLOCK SIZE
27 00074'177777                         -1
28       000075'                        .LOC DIR+?IDEL   ;DEFAULT DATA SENS.
29 00075'177777                         -1                ;DELIMITERS
30       000076'                       .LOC DIR+?IBLT
31
32                              ;LINE PRINTER I/O PACKET
33       000076'                LINPT:  .LOC LINPT+?ICH
34 00076'000000                         0
35       000077'                        .LOC LINPT+?ISTI
36 00077'040012                         ?ICRF+?RTDS+?OFOT  ;DATA SENS. OUTPUT RECS.
37       000100'                        .LOC LINPT+?ISTO
38 00100'000000                         0
39       000101'                        .LOC LINPT+?IBAD
40 00101'000244"                        ENTRY*2            ;POINTER TO ?GNFN BUFFER
41       000102'                        .LOC LINPT+?IRES
42 00102'000000                         0
43       000103'                        .LOC LINPT+?IRCL
44 00103'000204                         132.               ;MAX. ENTRY SIZE
45       000104'                        .LOC LINPT+?IRLR
46 00104'000000                         0
47       000105'                        .LOC LINPT+?IRNH
48 00105'000000                         0
49       000106'                        .LOC LINPT+?IRNL
50 00106'000000                         0
51       000107'                        .LOC LINPT+?IFNP
52 00107'000224"                        LPTPR*2       ;BYTE PTR.TO LINE PTR. NAME
53       000110'                        .LOC LINPT+?IMRS
54 00110'177777                         -1
55       000111'                        .LOC LINPT+?IDEL
56 00111'177777                         -1
57
58 00112'040114                 LPTPR:  .TXT "@LPT"
59       050124
60       000000
```

Figure 6-10. DLIST (continued)

```
 0003  DLIST
01                ;0003 DLIST
02
03 00115'000234"              .EOL:    .+1*2
04 00116'005000               EOL:     .TXT "<012>"
05
06                            ;GET-FILE-ENTRY PACKET
07 00117'000003               GFNPK:   .BLK ?NFLN
08       000117'                       .LOC GFNPK+?NFKY   ;USED BY SYSTEM
09 00117'000000                        0
10       000120'                       .LOC GFNPK+?NFNM
11 00120'000244"                       ENTRY*2            ;BUFFER POINTER
12       000121'                       .LOC GFNPK+?NFTP
13 00121'177777                        -1                 ;NO TEMPLATE
14       000122'                       .LOC GFNPK+?NFLN
15
16 00122'000204               ENTRY:   .BLK   132.
17 00326'000200               DIRNM:   .BLK   128.
18                                     .END   LIST

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS



  0004 DLIST

CONT   000017'        1/15    1/27
DIR    000062'        1/12    1/15    2/05    2/06    2/08    2/10    2/12
                      2/14    2/16    2/18    2/20    2/22    2/24    2/26
                      2/28    2/30
DIRNM  000326'        1/48    2/25    3/17
ENTRY  000122'        1/39    2/40    3/11    3/16
EOFTS  000043'        1/18    1/29
EOL    000116'        3/04
ERROR  000001   XN    1/03    1/36
GFNPK  000117'        1/37    3/07    3/08    3/10    3/12    3/14
LINPT  000076'        1/14    1/20    1/22    1/24    1/26    2/33    2/35
                      2/37    2/39    2/41    2/43    2/45    2/47    2/49
                      2/51    2/53    2/55
LIST   000000'        1/09    3/18
LPTPR  000112'        2/52    2/58
MSGPK  000056'        1/10    1/42    1/43    1/45    1/47    1/49
.ENTR  000055'        1/25    1/39
.EOL   000115'        1/21    3/03
.EREO  000054'        1/29    1/38
.ERRO  000052'        1/10    1/12    1/14    1/20    1/24    1/31    1/34
                      1/36
.GFNP  000053'        1/16    1/37
?GNFN  003245   MC    1/17
?GTME  004545   MC    1/09
?OPEN  004315   MC    1/11    1/13
?RETU  004600   MC    1/33
?WRIT  004436   MC    1/19    1/23
?XCAL  000001         1/10    1/12    1/14    1/18    1/20    1/24    1/34
```

Figure 6-10. DLIST (concluded)

End of Chapter

Chapter 7
Creating and Managing a Multitask Process

Introduction to Task Concepts

A *task* is the ultimate unit which uses system resources, including CPU control. A *program* is the current executable contents of a process's address space, and contains the code paths executed by tasks. A process contains only one program at any given time, but may change the current program many times during its lifetime. A *program file* is a program residing on disk.

Single-task programs are already familiar to users of high-level languages such as BASIC and FORTRAN. A single-task program has a single unified path connecting all its program logic, no matter how complex the branches.

There is no compelling reason why a process should be limited to performing only one task. Indeed, many tasks can be directed to run in a single code path, provided that path is re-entrant (i.e., does not modify its own contents during execution), and the state of each task can be preserved and restored as required. AOS permits multitasking of up to 32 concurrent tasks.

Multitasking provides a priority-oriented, structured facility for processing asynchronous events within a single process. It is a convenient way to handle complex parallel events which would otherwise be extremely difficult to manage.

Multitasking requires priority assignments for single tasks or groups of tasks, and allows unique identification numbers to be assigned to each individual task. Priority assignments assure that the highest priority task that is ready for execution will be scheduled to run; the unique identification number permits each task to be specified without ambiguity.

Applications such as process control frequently need to handle asynchronous events. Typical examples of these are:

- time-out routines
- alarm routines
- overlapped I/O
- asynchronous processing of data, interrupts, etc.

Task States and Priorities

Each user process will have one task, called the *initial task*, established for it when it is created. If your process needs more tasks, it must initiate at least the first of them by issuing an appropriate task call from the initial task. As more tasks are initiated, each of them may in turn initiate others. Each process may initiate up to 32 concurrent tasks (including the initial task).

Each task is assigned a priority when it is first initiated; priorities range from 0 (the highest) through 255 (the lowest). Priorities may be changed during the execution of the program. If more than one task exists at the same priority level, each task in that level will receive CPU control in round-robin fashion.

The initial task is created at the highest priority, and since it is the only active task in the program initially, it receives control. Other tasks which are as yet uninitiated are said to be in the *dormant* state. When the initial task establishes the second or subsequent task, the task scheduler places the highest priority task within the process into *execution.* Other tasks that have been initiated but are of lower priority are *ready* to run.

Among processes that are eligible to run, it is the highest priority process which receives control of the processor. The system's task scheduler always gives control to the highest priority ready task in the executing process; this transfer of control is called *scheduling.* Scheduling can occur at any time. The system guarantees only that the ready task with the highest priority will gain control of the CPU.

Task management calls executed by modules bound into the user context do not cause scheduling to occur unless they suspend the executing task explicitly (e.g., ?SUS, ?PRSUS). As for other system calls, those which go to completion without delay do not cause task scheduling; all other system calls do cause task scheduling. For example, ?RCALL to a currently resident overlay does not cause scheduling, but ?RCALL to a currently nonresident overlay does cause scheduling.

Several different events can cause an executing or ready task to be *suspended.* One of these events is the issuing of a system call. A suspended task cannot be scheduled or placed into execution. Each task has a number of event flags, corresponding to the events which can cause suspension. A task remains suspended as long as one or more of its event flags is set.

Memory Resources

Each task in the system must have a stack of at least 30 (decimal) words. Link will automatically reserve a minimum stack of 30 words for the initial task unless a larger stack is requested (/STACK argument switch).

You will usually need to allocate a larger stack if you issue any user runtime call (i.e., one bound from URT.LB) or if you issue one or more SAVE instructions. The system's user runtime management requires from decimal 20 to 22 stack words, and the frame size is 5 for each SAVE. Bear in mind that whenever a task issues a resource call (see Chapter 3), the system appends a two-word suffix to the caller's temporaries area. Thus, whenever you are calculating the total required stack size, you must consider the following:

- your use of resource calls;
- the number of SAVES (frame size equals 5 for each SAVE) and their nested depth;
- the amount of temporary storage (SAVE arguments);
- the number of program tasks.

All calls described in this chapter are located in modules bound into the user context; this makes their execution more efficient. Each of these modules is contained in a system library named URT.LB. If this library is not mentioned in the Link command line, then modules required for program execution will be bound into the shared context. If, however, you have no shared code or data in your program, Link will realize this and link URT.LB modules unshared. Thus, it will not create a shared area just for URT.LB modules. Appendix D explains how to obtain the titles and entry points of all system runtime modules which can be loaded into the user context.

Any reference in a task call to an address outside the process's address space will cause a hardware trap to occur and the process to be terminated. If the CLI created the offending process, then the CLI will output a trap message on the console (see the *AOS CLI User's Manual* for details). If a process other than the CLI created the offending process, the father process can receive an IPC message indicating the trap by issuing ?IREC from port ?SPTM.

TCB Queues

There are several variables unique to each task that the system maintains so that each task can be run independently of the others. This information is called the *user task state*. The system maintains each task's user task state within a table called a Task Control Block (TCB). TCBs reside and are maintained in the user context, and there are as many TCBs as there are tasks requested for use by the program. Information in the user task state includes the following: accumulator contents, PC, carry, the contents of location ?USP, and the stack locations, 40 through 43. ?USP, Unique Storage Position, is location 16; this is a single ZREL location that is maintained for each task.

You may specify other information to be maintained as part of the user task state: the floating point unit state, and additional contiguous ZREL locations. Appendix C, which describes TCB structure, explains how to specify this additional user task state information.

The system manages two queues of TCBs for each program: an active queue and an inactive queue. The active queue consists of a linked list of TCBs for the executing task and other tasks that are either suspended or ready for execution. TCBs in this queue are connected by the link words of each TCB (?TLNK).

TCBs can be allocated by the pseudo-op .TSK in a source module, or by using the /TASKS switch in the Link command line.

User Status Table

Another table found in each program's user context is the *User Status Table* (UST). This table contains a collection of information pertinent to the execution of each program, and contains (among other things) information about the program's TCB queues.

TCB and UST structures are defined parametrically in the User Parameter File, PARU.LS (reproduced in Appendix D) and are described in Appendix C.

Task Initiation

Before any task other than the initial task can run, it must be made known (initiated) to the system. The system provides the initial task; but you must use the ?TASK call to initiate all other tasks. This call permits either a single task or multiple tasks to be initiated for execution. Execution can occur either on a strict priority basis or it can occur by priority at a certain time of day. Tasks initiated for execution based on a time of day can also be queued for repetitive execution at specific time intervals.

If the queued task facility is used, a queued task manager must be initialized by system call ?IQTSK. Queued tasks are dequeued by system call ?DQTSK.

After a TCB is allocated for the initiated task (but before that task is entered into the active queue), the system calls a task initiation routine, ?UTSK. If you have not provided a routine with this name, then the system calls and returns from the default ?UTSK routine (bound from URT.LB) which performs no action. Appendix C describes ?UTSK at greater length.

Event Flags

Each task has a number of event flags associated with it. If one or more of its event flags are set, then the task is in the suspended state. A task is ready if all its event flags are clear. The names of these event flags, and the events which may set them, are as follows:

Event Flag	Event
?TSPN or ?TSIG	Task has issued a system call and is awaiting the completion of its execution in system space.
?TSSG	Task is waiting for an overlay area or shared routine, or has issued an ?XMTW call and the message has not been received, or is awaiting a message (?REC).
?TSSP	Task has been suspended by one of the following calls: ?IDSUS, ?PRSUS, or ?SUS.
?TSRC	Task has issued ?TRCON and is awaiting a message from the console.
?TSIW	System is preparing this task for execution.

Event flags ?TSGS and ?TSAB are used by the operating system. ?TSUF is used by high-level languages running under the operating system.

?IDSTA returns a specific task's status word, indicating the states of each of its event flags.

Changing Task Priorities

The system provides two calls for changing task priorities:

?IDPRI	Change the priority of a task by ID.
?PRI	Change the calling task's priority.

?IDPRI permits the calling task to change its or another task's priority. The target task is named by specifying its unique ID. ?PRI permits the calling task to change its own priority only.

Readying and Suspending Tasks

The system provides a variety of ways to suspend one or more tasks, i.e., to set their ?TSSP event flags. Corresponding to these calls is another set of calls which readies tasks by clearing their ?TSSP event flags. These calls are as follows:

?IDRDY	Ready a task specified by ID.
?IDSUS	Suspend a task specified by ID.
?PRRDY	Ready all tasks of a given priority.
?PRSUS	Suspend all tasks of a given priority.
?SUS	Suspend the calling task.

?IDRDY and ?IDSUS will ready or suspend a single task specified by its ID number. ?PRRDY and ?PRSUS ready or suspend all tasks which have the same priority. ?SUS suspends the calling task only.

Killing and Aborting Tasks

Since each task has a stack, each task module may begin and end with SAVE and RTN instructions. Upon execution of RTN, the task is killed automatically and the system releases its resources. The TCB and any overlay or shared routine if one was specified in the ?TASK call that initiated the task are resources that are released on RTN.

In addition to this implicit means for killing a task, the system provides explicit calls for terminating tasks or aborting their current activity:

?IDGOTO	Stop a task's current activity, abort its system call (if any), and transfer its control elsewhere.
?IDKIL	Kill a task specified by ID.
?KILAD	Define a kill-processing routine.
?KILL	Kill the calling task (self-termination).
?PRKIL	Kill all tasks of a single priority.

?IDGOTO unconditionally stops the target task's activity by aborting any outstanding system call it may have, or by removing its suspension caused by ?SUS, ?PRSUS, IDSUS, ?REC, or ?XMTW. Additionally, this call directs the target task to run at a new location when it gains control of the CPU. The priority of this task remains unchanged.

A typical use of ?IDGOTO is to provide CTRL-C CTRL-A interrupt processing (described in Chapter 9). In essence, a normally suspended console interrupt task could gain control and issue an ?IDGOTO call to halt the main task's outstanding activity and output a termination message.

Any task can be killed by issuing ?KILL or RTN. Upon each task termination, any resources it acquired by a general procedure call are released. Killing a task does not release its stack or other resources in an orderly fashion, however. To permit an orderly release of resources, the system provides two mechanisms: a special kill-processing routine for each task, and a general kill-processing routine for all tasks within the process. Each task can define a kill-processing routine for itself by issuing ?KILAD. You may specify a general kill-processing routine, ?UKIL, gaining control upon each task's termination. This facility is described in Appendix C.

On either ?PRKIL or ?IDKIL, the system examines the event flags of each task to be terminated. If a target task was suspended by either ?SUS, ?PRSUS, ?IDSUS, ?REC, or ?XMTW, then readying the target task immediately lifts the suspension. If its ?TSPN event flag is set, then its outstanding system call is aborted. If the task were in suspension due to some other event, the associated event flag would have to be reset before the task could be readied.

In any case, after suspension has been lifted from the target task, it is raised to the ready state and one of two actions occurs. First, if no kill-processing address is provided, the task is simply killed. Alternatively, a task can specify a special kill-processing routine for itself with the ?KILAD call. If such an address has been defined, then when the target task receives control, it goes to this routine. A task entering its kill-processing routine can perform an orderly release of its resources if it so chooses. It will not actually be killed until it issues a ?KILL call in this routine.

When a task is killed its TCB is relinquished to the inactive TCB queue for possible use in the initiation of other tasks. Abnormal program termination results if the last task in the active TCB queue is terminated with an ?IDKIL, ?PRKIL, or ?KILL command.

Intertask Communications

The operating system provides a mechanism for transmitting and receiving one-word messages between tasks. These messages are deposited in message locations called *mailboxes*. The contents of each mailbox are set to zero when it contains no message. The following standard intertask message calls are defined:

?REC	Receive a message.
?RECNW	Receive a task message without waiting.
?XMT	Transmit a message.
?XMTW	Transmit a message and await its receipt.

Two other system calls, ?IXMT and ?IMST, transmit a message from a user device interrupt service routine, and receive that message. These calls are described in Chapter 8, *User Device Support*.

A task wishing to receive a message issues a ?REC task call, specifying a mailbox, then the task becomes readied immediately as a sign that the message is available; otherwise the task remains suspended (?TSSG) until transmission occurs. A task wishing to transmit a message specifies a message and a mailbox by either ?XMT or ?XMTW. If it issues ?XMT, then it remains ready after issuing the call; if ?XMTW, then it becomes suspended (?TSSG) until reception of the message by a different task issuing a ?REC call. If several tasks attempt to receive a message from the same address, only the highest priority task will receive the message unless the transmitting task specifies that the message should be *broadcast* to all receivers, regardless of priority.

The ?REC and ?XMT/?XMTW calls can lock and unlock a critical region. A critical region is a procedure or database which is shared by several tasks, yet can be assessed by only one task at a time. To protect a critical region, the program must define a synchronization word, the mailbox; all concerned tasks then attempt to receive a message from the mailbox. The task in control of the locked critical region then issues a transmit, ?XMT without broadcast, to the mailbox when the region is to be made available to the other waiting tasks. The highest priority task waiting to receive word at the mailbox is then readied and gains unique control of the region. This task, in turn, retains use of the critical region until it unlocks the region by issuing its own ?XMT, etc.

This technique requires that the locking facility be initialized before any task uses it. Initialization can be performed either by setting the mailbox initially to a nonzero value, or by having an initialization task issue an ?XMT to the synchronization mailbox.

A critical region can also be protected by disabling the scheduling of the task environment (see the following section). However, since this method can potentially disrupt task management, the use of ?XMT/?REC is preferred in most situations.

Calls Affecting an Entire Process

Calls discussed in this section differ from calls discussed previously in that these calls perform actions affecting the entire process rather than a single task. These calls have been implemented in modules that are loaded into the user context for a faster response than would be the case if they were executed in the operating system's space.

In some instances, it may be desirable to suspend briefly the scheduling facility. That is, the system's normal rule that CPU control be given to the highest priority ready task can be overridden. For example, you might suspend scheduling to control race conditions between tasks which are competing for a critical resource. Since the disablement of scheduling, even briefly, is potentially disruptive, it must be performed with caution.

Additionally, you must realize that disabling scheduling will not affect system activities such as spooling and interrupt service. The scheduling function is disabled by system calls ?DRSCH and ?DFRSCH, and it is re-enabled by system call ?ERSCH.

Another series of task calls test and alter the LEF mode within a process. A user within a process may wish to use the *load effective address* instruction, LEF. However, this instruction can be issued only when the CPU is in the LEF mode (bit 9 set in the user map status word). Moreover, when the CPU is in the LEF mode, no I/O instruction can be issued because it would be interpreted as a LEF instruction.

Users can determine whether or not a process is in LEF mode and can set or reset this mode by means of the following calls:

?LEFD Disable LEF mode.

?LEFE Enable LEF mode.

?LEFS Get the LEF mode status.

Initially each process has LEF mode enabled. The system saves the state of LEF mode upon entry to each user device interrupt service routine and restores that state upon exit from the routine. (User device interrupt service routines are described in Chapter 8.)

System Call Summary

The following system calls manage a multitask environment:

?DFRSCH Disable scheduling in this program and return the previous state.

?DQTSK Dequeue a task previously queued.

?DRSCH Disable scheduling in this program.

?ERSCH Enable scheduling in this program.

?IDGOTO Redirect a task.

?IDKIL Kill a task specified by ID.

?IDPRI Change the priority of a task specified by ID.

?IDRDY Ready a task specified by ID.

?IDSTAT Get the status of a task specified by ID.

?IDSUS Suspend a task specified by ID.

?IESS Initialize the extended state save area.

?IQTSK Name a queued task manager.

?KILAD Define a kill-processing routine.

?KILL Kill the calling task.

?LEFD Disable LEF mode.

?LEFE Enable LEF mode.

?LEFS	Determine the LEF mode status.
?MYTID	Return ID and priority of the calling task.
?PRI	Change the priority of the calling task.
?PRKIL	Kill all tasks of a given priority.
?PRRDY	Ready all tasks of a given priority.
?PRSUS	Suspend all tasks of a given priority.
?REC	Receive a task message.
?RECNW	Receive a task message without waiting.
?SUS	Suspend the calling task.
?TASK	Initiate one or more tasks for immediate or periodic execution.
?TRCON	Read a task message from a process console.
?XMT	Transmit a task message.
?XMTW	Transmit a task message and wait for its receipt.

?DFRSCH

Disable scheduling in this program and return the previous state.

?DFRSCH
exception return
normal return

Input/Output

Input:

none.

Output:

AC0	flag bit:
	bit ?DSCH = 0, scheduling not already disabled.
	bit ?DSCH = 1, scheduling already disabled.
AC1	undefined.
AC2	undefined.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call prevents scheduling from occurring in the current program until scheduling is re-enabled explicitly (?ERSCH). ?DFRSCH should be issued only with caution, since it disrupts the system's ordinary management of multitasking within this process; the task that issues this call will retain control even though other tasks within this process become ready.

When ?DFRSCH succeeds, it returns flag bit ?DSCH in AC0, which indicates whether or not multitask scheduling was enabled before ?DFRSCH took action. A value of 1 in AC0 indicates that multitask scheduling was disabled before you issued ?DFRSCH.

You can use ?DFRSCH to lock all other tasks out of a critical region and still retain in memory a copy of the previous state of multitask scheduling. Internally called subroutines that need to disable scheduling would use ?DFRSCH rather than ?DRSCH to restore previous multitask scheduling environments before exiting from the subroutine.

If scheduling has already been disabled, it will simply remain disabled and control will take the normal return. The system re-enables scheduling when a task issues the ?DFRSCH call and then terminates itself with a ?KILL call.

?DQTSK

Dequeue one or more tasks.

?DQTSK *[task definition packet]*
exception return
normal return

Input/Output

Input:

AC2 address of ?TASK Definition Packet (TDP).

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

This call dequeues a task or tasks queued for execution. If the task(s) are currently active, they will remain active. After becoming inactive (dormant), they can be queued again only by a call to ?TASK.

Note that the TDP passed to ?DQTSK must be the same packet (not a copy) that was passed to ?TASK to queue the task(s). After issuing a ?TASK call for queued tasks, you are not free to alter or delete the TDP until after you issue ?DQTSK.

?DRSCH

Disable scheduling.

?DRSCH
exception return
normal return

Input/Output

Input:

none.

Output:

unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call prevents scheduling from occurring in the current program until scheduling is re-enabled explicitly (?ERSCH). ?DRSCH should be issued only with caution, since it disrupts the system's ordinary management of multitasking within this process; the task that issues this call will retain control even though other tasks within this process become ready.

If scheduling has already been disabled, it will simply remain disabled and control will take the normal return. The system re-enables scheduling when a task issues the ?DRSCH call and then terminates itself with a ?KILL call.

?ERSCH

Enable scheduling.

?ERSCH
exception return
normal return

Input/Output

Input:

none.

Output:

unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

Initially task scheduling is enabled within a program. If task scheduling has been suspended by a call to ?DRSCH, then scheduling can be reactivated by issuing ?ERSCH. If scheduling has already been enabled, it will simply remain enabled and control will go to the normal return.

?IDGOTO

Redirect a task.

?IDGOTO
exception return
normal return

Input/Output

Input:

AC0 address of new code sequence.

AC1 target task ID.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

TASK EXCEPTION codes

Description

This call directs the target task to transfer to a new code sequence, instead of resuming its normal activity, when it next gains control of the CPU. If the target task has an outstanding system call, that will be aborted. Likewise, if the target task is suspended due to a ?IDSUS, ?PRSUS, ?REC, ?SUS, or ?XMTW, the suspension will be lifted. Note that, unlike ?IDKIL, ?IDGOTO does not change the target task's priority. You cannot use this call to transfer control within the calling task; i.e., AC1 cannot contain the ID of the caller.

?IDKIL

Kill a task specified by ID.

?IDKIL
exception return
normal return

Input/Output

Input:

AC1 ID of task to be killed.

Output:

AC1 unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

The task targeted by this call is readied immediately at the highest priority if it was suspended by event flags ?TSSG or ?TSSP. If it was suspended due to the remaining event flags, then these must be reset before the task becomes readied. If the task has issued a system call (event flag ?TSPN) or ?TRCON (event flag ?TSRC), then the call will be aborted and the flag will be reset; control will not go to the aborted call's exception return.

After all its event flags have been reset, the target task is ready and one of two actions occurs. First, if no kill-processing routine is provided, then the task is killed. Alternatively, if the target task had provided such a routine, then it receives control at that routine.

Note that the system generates the error message "Last Task Was Killed," and aborts the program when the ?IDKIL call (or any other "kill" task call) terminates the last active task.

?IDPRI

Change the priority of a task specified by ID.

?IDPRI
exception return
normal return

Input/Output

Input:

AC0 new task priority.

AC1 ID of task to undergo priority change.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

This call changes the priority of that task whose ID was specified.

?IDRDY

Ready a task specified by ID.

?IDRDY
exception return
normal return

Input/Output

Input:

AC1 ID of task to be readied.

Output:

AC1 unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

This call readies the task specified by ID only if that task's ?TSSP event flag has been set, i.e., ?IDSUS, ?PRSUS, or ?SUS calls have been issued.

?IDSTAT

Get the status of a task.

?IDSTAT
exception return
normal return

Input/Output

Input:

AC1 ID of task whose status is to be obtained.

Output:

AC0 status word.

AC1 unchanged.

AC2 base address of the target task's TCB.

Exceptional Condition Codes in AC0

TASK codes

Description

This call returns a word which describes the target task's priority and its status, specifically, ?TSTAT of the task's TCB. The interpretation of this word is as follows:

Event Flag	Event
?TSPN or	Task has issued a system call whose execution is taking place in system space.
?TSIG	
?TSSG	Task is waiting for an overlay area or shared routine, or has issued an ?XMTW call and the message has not been received, or is waiting a message (?REC).
?TSSP	Task has been suspended by one of the following calls: ?IDSUS, ?PRSUS, or ?SUS.
?TSRC	Task has issued ?TRCON and is awaiting a message from the console.
?TSIW	System action is occurring in preparation for this task's execution.

Event flags ?TSGS and ?TSAB are used by the operating system. ?TSUF is used by high-level language running under the operating system.

If none of these flags are set (i.e., if the word contains zero), then the task is in the ready state.

?IDSUS

Suspend a task specified by ID.

?IDSUS
exception return
normal

Input/Output

Input:

AC1 ID of task to be suspended.

Output:

AC1 unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

This command suspends a task by setting its ?TSSP flag.

?IESS

Initialize the extended state save area.

?IESS
exception return
normal return

Input/Output

Input:

AC0 size of save area.

AC1 page zero starting address.

AC2 destination address of storage area for this task.

Output:

e

AC0 unchanged.

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call sets up both the User Status Table (UST) to indicate the size and location of memory locations to be saved on a per task basis, and the task's Task Control Block (TCB) to point to the place to save task information for this task.

The system maintains page zero locations, Unique Storage Position (16), and the stack pointers (40-43) on a per task basis, and saves them in the TCB when the task is not executing (see earilier TCB discussion). If your tasks need more page zero locations maintained on a per task basis, you should group the tasks together and have each task using the locations issue an ?IESS system call.

Note that each task doing an ?IESS call must specify the same size area in AC0 and the same starting address in AC1. The end of the area must not exceed 377.

Use of ?IESS can lengthen task scheduling time substantially.

?IQTSK

Create a queued task manager.

?IQTSK
exception return
normal return

Input/Output

Input:

AC0 priority to give to the task manager (0 if the caller's priority).

AC1 ID to give to the task manager.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

Before queued tasks can be specified in a ?TASK call, a queued task manager must be established by a call to ?IQTSK, and one TCB must be allocated for this manager. ?IQTSK may be issued only once in each program.

?IQTSK specifies the priority of the queued task manager; if zero is specified, then the caller's priority is used. ?IQTSK also assigns an ID to the queued task manager, since this ID is required for the system's control of the manager. After ?IQTSK has been issued, the system will use the queued task manager as required.

?KILAD

Define a kill-processing routine.

?KILAD
exception return
normal return

Input/Output

Input:

AC0 routine address.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call permits a special address to be specified which will receive control the first time that the termination of the task issuing this call is attempted. (?KILL and RTN terminate the task without going to the kill-processing address.)

The kill-processing address lets a task release its system resources before termination. Resources such as overlay areas acquired with ?OVLOD, channels, and user devices must be explicitly released. Having released these resources and performed any other desired functions, the task then issues a ?KILL to terminate itself.

A task in a kill-processing routine is executing at the highest priority. Thus, the task will retain control until it relinquishes it, either by a task state transition or by a priority change.

?KILL

Kill the calling task.

?KILL

Input/Output

Input:

none.

Output:

not applicable.

Exceptional Condition Codes in AC0

No exceptional condition codes are possible.

Description

There is neither an exception nor a normal return from this call, since at the completion of this call the task is terminated. When the task is terminated, its TCB is removed from the active queue and is placed in the inactive queue. The calling task is the only task that can be terminated by this command.

If the ?KILL call directs the last active task to terminate itself, the system will generate the error message "Last Task Was Killed," and will abort the program.

?LEFD

Disable LEF mode.

?LEFD
exception return
normal return

Input/Output

Input:

none.

Output:

AC0 contents lost.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call disables LEF mode so that LEF instructions cannot be issued. If the process is currently running with LEF mode disabled, the call is effectively nonoperational.

?LEFE

Enable LEF mode.

?LEFE
exception return
normal return

Input/Output

Input:

none.

Output:

AC0 contents lost.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call enables LEF mode, so LEF instructions can then be issued. If a process is currently running with LEF mode enabled, then this call is effectively nonoperational.

?LEFS

Get the current LEF mode status.

?LEFS
exception return
normal return

Input/Output

Input:

none.

Output:

AC0 user map status word (bit 9 indicates LEF mode status).

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call determines whether or not the process is running with LEF mode enabled.

?MYTID

Return the ID and priority of the calling task.

?MYTID
exception return
normal return

Input/Output

Input:

none.

Output:

AC0	task ID of caller.
AC1	priority of calling task.
AC2	unchanged.

Exceptional Conditional Codes in AC0

No exceptional condition codes are currently defined

Description

This call returns the task ID and priority of the calling task in AC0 and AC1, respectively.

?PRI

Change the priority of the calling task.

?PRI
exception return
normal return

Input/Output

Input:

AC0 priority.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call changes the priority of the calling task to the value contained in AC0. If other tasks already exist with this new priority, then the caller will be ranked last in the priority class for round-robin scheduling. That is, all other ready tasks within the same process which have the same priority will be given control before the task issuing the ?PRI call gains control.

If a priority greater than 255 is put into AC0, the value will be truncated to the least significant eight bits.

?PRKIL

Kill all tasks of a given priority.

?PRKIL
exception return
normal return

Input/Output

Input:

AC0 priority of tasks to be killed.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

The tasks targeted by this call are readied immediately with the highest priority, if they were suspended by events that set flags ?TSSG or ?TSSP. If one or more were suspended by events which set the remaining event flags, then these flags must be reset before the task becomes readied. If such tasks issued system calls whose execution occurs in system space (event flag ?TSPN) or call ?TRCON (event flag ?TSRC) then the calls will be aborted and the flag(s) will be reset. Control will not go to the aborted calls' exception returns.

After all their event flags have been reset, the target tasks are ready and one of two actions occurs. If no kill-processing routines are provided for any of the tasks, then they are simply killed. Alternatively, if any task provided such a routine, then it receives control at that routine.

If ?PRKIL terminates the last active task, the system will generate the error message, "Last Task Was Killed," and will abort the program.

?PRRDY

Ready all tasks of a given priority.

?PRRDY
exception return
normal return

Input/Output

Input:

AC0 priority of tasks to be readied.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call readies all tasks suspended previously by ?PRSUS, ?SUS, or ?IDSUS whose priority is given as input to this call. Tasks with this priority that are suspended for other reasons (e.g., an outstanding system call) will be readied only when these other conditions are removed. That is, this call resets event flag ?TSSP but leaves other event flags unchanged.

If no task exists with the specified priority, no action is taken and control goes to the normal return.

?PRSUS

Suspend all tasks of a given priority.

?PRSUS
exception return
normal return

Input/Output

Input:

AC0 priority of tasks to be suspended.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This command suspends all tasks whose priority is given as input. That is, it sets their ?TSSP event flags. Each task suspended with ?TSSP set will remain suspended until the flag is reset. The flag is reset when a task becomes the target of either ?PRRDY or ?IDRDY. The calling task itself will become suspended if its priority is specified by AC0.

If no task exists with the specified priority, no action is taken and control goes to the normal return.

?REC

Receive a message.

?REC
exception return
normal return

Input/Output

Input:

AC0 mailbox address.

Output:

AC0 unchanged.

AC1 message.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call returns a message that another task sent by a transmit command (see ?XMT and ?XMTW), and restores the contents of the message address (mailbox) to zeros. The mailbox address must not have bit zero set.

If the ?REC call has been issued but no transmitter has yet sent a message, then the receiving task remains suspended (event flag ?TSSG) until the message is sent. If the message was already sent and if the receiving task was not suspended by ?PRSUS or ?IDSUS (event flag ?TSSP), then the task is readied.

If several tasks attempt to receive the same message, only the highest priority task will receive the message unless it was broadcast to all receivers.

?RECNW

Receive a task message without waiting.

?RECNW
exception return
normal return

Input/Output

Input:

AC0 mailbox address.

Output:

AC0 unchanged.

AC1 message.

AC2 unchanged.

Exceptional Conditional Codes in AC0

ERNMW No message waiting.

Description

This call returns a message that another task has sent, and resets the mailbox to zero. Unlike ?REC, the receiving task returns an error if the message has not already been sent when the ?RECNW call is issued.

?SUS

Suspend the calling task.

?SUS
exception return
normal return

Input/Output

Input:

none.

Output:

unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call places the calling task in the suspended state by setting its ?TSSP event flag. The suspended task remains suspended until it is readied by an ?IDRDY or ?PRRDY call.

?TASK

Initiate one or more tasks.

?TASK *[task definition packet]*
exception return
normal return

Input/Output

Input:

AC2 task definition packet (TDP) address.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

This call initiates one or more tasks to create a multitask program. Code for these tasks may exist in a root context, overlay, or shared routine.

Before passing control to the task code, the system will check for the presence of a user-supplied routine named ?UTSK, and if this exists it will receive control. ?UTSK can be used to manage special task resources; ?UTSK is described in Appendix C.

When control is passed to the task code, AC3 contains the return address. Therefore, if you wish to be able to return to this address in the calling task, the created task's code must begin with a SAVE instruction and end with an RTN instruction. Overlay or shared routine procedures *must* have this format. Except for the initial task, the outermost RTN of a task functions as an effective ?KILL, and if that task is a shared routine or overlay, it is released. (Nested SAVE/RTN pairs within a task do not do this.)

Do not begin and end the initial task with a SAVE/RTN pair; this will cause unpredictable behavior.

Information specifying the task initiation is passed in a Task Definition Packet (TDP). There are two versions of the TDP: standard and extended. The extended packet contains all information found in a standard packet, with extra words to define the time of day for task creation, the number of times for creation to occur, and the elapsed time between each creation.

Task queueing is obtained by specifying repetitive or future task creation. Note that you are not free to alter or delete an extended TDP until after you issue a corresponding ?DQTSK.

The structure of the standard TDP is shown in Figure 7-1 and described in Table 7-1.

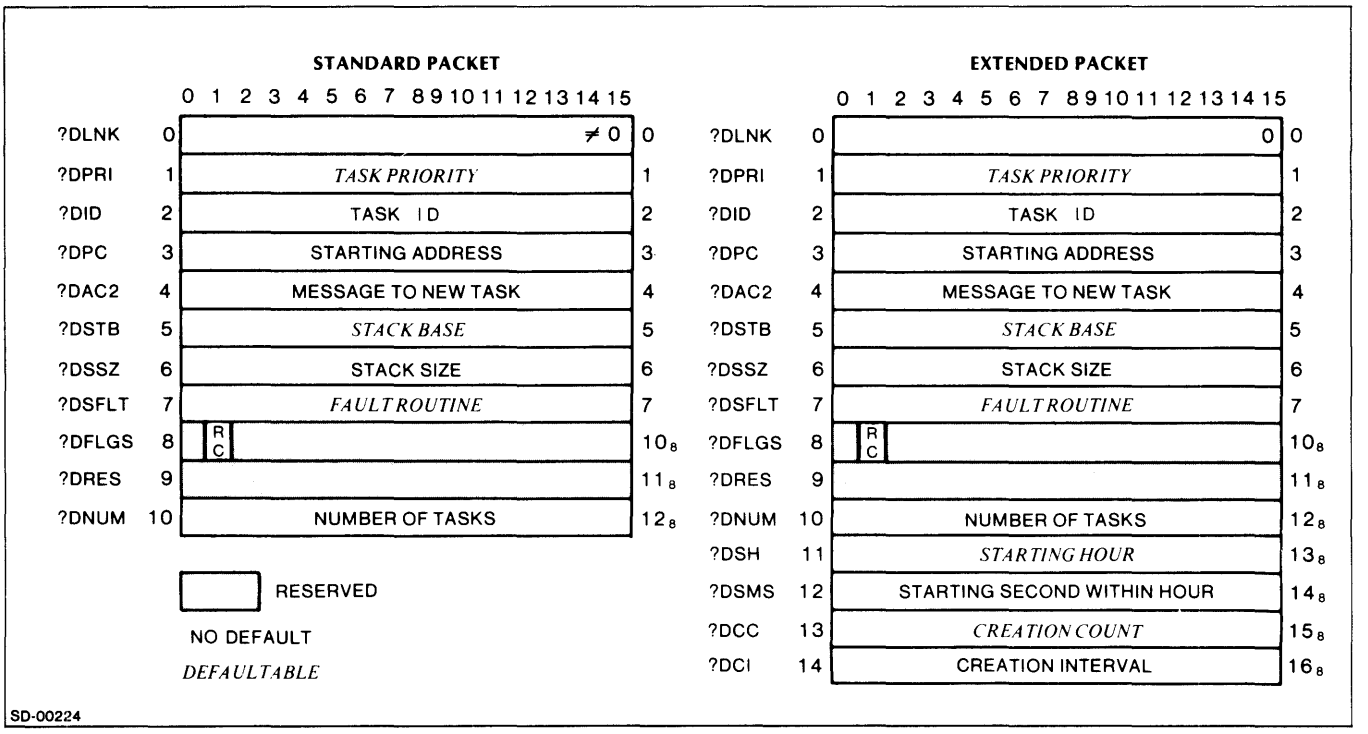


Figure 7-1. Task Definition Packets

Table 7-1. Standard Task Definition Packet Structure

Offset	Contents
?DLNK	Packet type: 0 specifies extended, nonzero specifies standard.
?DPRI	Priority; 0 specifies the caller's priority.
?DID	Task ID; 0 specifies no ID.
?DPC	Starting address or procedure entry.
?DAC2	Message to new task(s) AC2.
?DSTB	Stack base (or -1, user allocates the stack).
?DSSZ	Stack size in words.
?DSFLT	Fault routine: -1 specifies caller's stack fault routine address.
?DFLGS	Task specification flags: ?DFLRC - Task resides in a procedure (see Chapter 3).
?DRES	Reserved.
?DNUM	Number of tasks to create.

The extended TDP (see Figure 7-1) consists of a standard TDP, with ?DLNK containing 0, and a suffix of the words described in Table 7-2.

Table 7-2. Extended Task Definition Packet Suffix

Offset	Contents
?DSH	Starting hour; 0 or -1 specifies immediate execution. When ?DSH occurs before the current time of day, the system queues the task for the next day. When ?DSH is greater than 24:00 hours and less than 48:00 hours, the system queues the task for the next day. When ?DSH is equal to (24.*D)+H, the system queues the task in D days. To queue for midnight, enter the hour 24.
?DSMS	Starting second in the hour; ignored if (?DSH) = -1.
?DCC	Number of times to create task(s); -1 specifies creation of tasks without limit.
?DCI	Number of seconds between each creation.
NOTE: You should be aware that if your process is blocked at the time specified by ?DSH and ?DSMS, the queued task will not begin executing until the process is unblocked.	

The extended packet suffix follows ?DNUM immediately.

The value of ?DID must not be 1, as this is the number of the initial task. If in ?DID a task ID is specified and there are multiple tasks to be created, the initial ID is assigned to the first task and this value is incremented for succeeding tasks.

The location where task execution will begin can be an address in the root context, an address or an offset in an overlay, or a shared routine entry. This location is specified in ?DPC; task specification flags indicate whether it is an overlay address or offset, a shared routine entry, or neither.

A one-word message is sent to each new task's AC2. This message is specified in ?DAC2 if only one task is being initialized. For multiple task initialization, ?DAC2 contains the address of a table of messages which will be placed in each task's AC2. The system will pass to each task's AC3, the address of an appropriate termination module. Thus if SAVE/RTN is issued, the task will be killed and -- if the task was not in the root -- the shared routine or overlay will be released.

If a stack base is specified, (i.e., ?DSTB is not -1), then this value will become the task's stack pointer, SP, and the task's frame pointer, FP. The task's stack limit equals the sum of (?DSTB) and (?DSSZ). If the number of tasks to be created (?DNUM) exceeds one, then ?DSTB must specify a base area whose size equals (?DNUM)*(?DSSZ). Each task has its own stack, and the common task code must be re-entrant. See Figure 7-2.

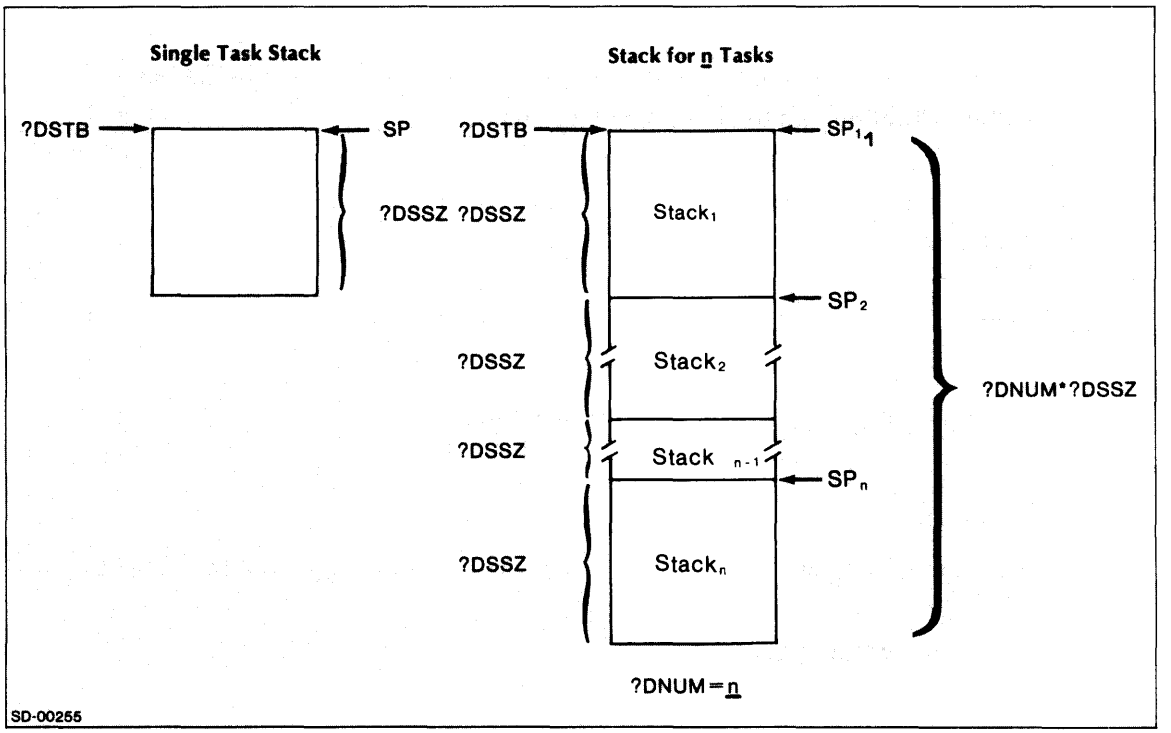


Figure 7-2. Stack Parameters in Single- and Multitask Initializations

If task initialization is to occur at a time of day, or if task creation is to occur more than once (or both), then an extended task definition packet is required. Specifying that task initialization is to occur more than once (offset ?DCC), possibly at some future time, implies task queuing; a task queue manager must have been specified by a previous call to ?IQTSK. If several tasks are queued by a single ?TASK call, then ?DQTSK dequeues all of them. If there are not enough TCBs to satisfy the request to initiate several tasks, no tasks will be initiated.

?TRCON

Read a task message from a process console.

?TRCON
exception return
normal return

Input/Output

Input:

AC0 byte pointer to message area (must be word-aligned).

Output:

AC0 unchanged.

AC1 message byte count (including NEW LINE terminator, excluding task ID and comma delimiter).

Exceptional Condition codes in AC0

TASK codes

Description

This call prepares the calling task to receive a message that will be transmitted from the process console keyboard. More than one task within a program may issue this call (the system uses task IDs to differentiate between receivers). When ?TRCON is used to read task messages from the process console, ?READ cannot be used with that console.

?TRCON will fail if it is issued by a task that has no ID.

Before issuing this call for the first time, be sure that there is a TCB available for use by the system. This TCB will be used to create a message manager task whose function is to monitor the keyboard for ?TRCON messages. A > prompt is issued on the process console to indicate that the message manager task is ready to receive a message.

The task issuing ?TRCON takes the normal return when it has received the message. Each message must begin with the decimal ID of the receiving task, followed by a comma delimiter and the message itself. The last character typed in the total message must be NEW LINE. The length of the total message, including NEW LINE, can be up to 132 characters. The format of an input message is:

task ID ,message<NEW LINE>

The system outputs two diagnostic messages if errors occur in messages intended for tasks. These messages and their meanings are:

****TID NOT FOUND***

No task with the specified ID was found to be awaiting a ?TRCON message.

****INPUT ERROR***

Non-numeric character in task ID.

?XMT

Transmit a task message.

?XMT
exception return
normal return

Input/Output

Input:

AC0	mailbox address.
AC1	message.
AC2	-1 to broadcast message to all waiting receivers; otherwise, only highest priority receiver.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

This call lets a task send a one-word, nonzero message to a mailbox whose contents are currently zero. If the message is broadcast (AC2 set to -1), all receivers at this mailbox get the message; otherwise, only the highest priority receiver gets the message.

?XMTW

Transmit a task message and wait.

?XMTW
exception return
normal return

Input/Output

Input:

AC0	mailbox address.
AC1	message.
AC2	-1 to broadcast message to all waiting receivers; otherwise, only to highest priority receiver.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

TASK codes

Description

This call lets a task send a one-word, nonzero message to a mailbox whose contents are currently zero. The task issuing this call is suspended (i.e., event flag ?TSSG will be set) until the message is received by another task. If a ?REC is already outstanding for this message when the ?XMTW call is made, then the transmitting task is not suspended.

If the message is broadcast (AC2 set to -1), all receivers at this mailbox get the message; otherwise, only the highest priority receiver gets the message.

Examples

This section illustrates the ?KILL, ?TASK, and ?TRCON calls. Figure 7-3 shows the first program, NEWTASK.

In NEWTASK, the initial task creates a new task and then kills itself. An examination of the task parameter packet shows the following specifications for this new task: a standard packet, caller's (i.e., initial task's) priority, task ID of 1, new task's starting address is NEWTASK, no message to the task, a 30-word STACK, uses the default fault routine, no task specification flags, task does not reside in an overlay, and only one task is initiated. We did not need to specify an ID for this task; we did so simply because we will use this same packet in the next illustration program, and in that program we will need an ID.

After NEWTASK is initiated it terminates the process and outputs a message indicating that it has done so.

Figure 7-4 shows ECHO, a program illustrating the use of ?TRCON.

Just as in NEWTASK, the initial task initiates a second task and then kills itself. This new task then waits for a console message from us, outputting a > prompt indicating that it is ready for a message. When we receive the prompt, we type on the console a message of the form:

```
2,message
```

where 2 indicates that the receiving task's ID is 2. (The initial task's ID was 1.) The task then terminates itself and outputs message via the ?RETURN call.

```
 0001  NEWTA     AOS ASSEMBLER REV 01.09          16:27:57 12/05/78
01
                                    .TITL   NEWTASK
03                                  .EXTN   ERROR
04
05                 ;THIS PROGRAM HAS TWO TASKS: THE INITIAL TASK, AND A NEW
06                 ;TASK WHICH THE INITIAL TASK CREATES. AFTER CREATING THE
07                 ;NEW TASK, THE INITIAL TASK TERMINATES ITSELF. THE NEW
08                 ;TASK THEN TERMINATES THE PROCESS, SIGNALING SUCCESS
09                 ;WITH A SHORT ?RETURN MESSAGE.
10
11                                  .NREL
12        000002                    .TSK 2
13
14                         START:   ?TASK   PKT      ;CREATE THE NEW TASK
15 00004'002410                     JMP     @.ERROR
16                                  ?KILL   ;THE INITIAL TASK KILLS ITSELF
17 00007'024444            NEWTSK:  LDA     1,MSG    ;GET INITIAL'S SWAN SONG
18 00010'030462                     LDA     2,FLAGS   ;AND THE ?RETURN FLGS.
19                                  ?RETURN           ;AND TERM.THE PROCESS
20 00013'002401                     JMP     @.ERROR
21 00014'000000$           .ERROR:  ERROR
22
23 00015'000036            STACK:   .BLK    30.      ;RESERVE A 30-WORD STACK
24 00053'000130"           MSG:     .+1*2
25 00054'052110                     .TXT    "THIS IS NEWTASK SIGNING OFF"
26        044523
27        020111
28        051440
29        047105
30        053524
31        040523
32        045440
33        051511
34        043516
35        044516
36        043440
37        047506
38        043000
39
```

Figure 7-3. NEWTASK (continues)

```
!0002  NEWTA
01
02 00072'100041          FLAGS:  ?RFCF+33.        ;MSG.IN CLI.FORMAT,33
03                                                ;CHARS. LONG
04                ;STANDARD TASK PACKET
05        000073'         PKT:    .LOC    PKT+?DLNK  ;PACKET EXTENDED?
06 00073'000001                   1                  ;NO.
07        000074'                 .LOC    PKT+?DPRI  ;USE THE CALLER'S
08 00074'000000                   0                  ;PRIORITY
09        000075'                 .LOC    PKT+?DID   ;TASK I.D. IS
10 00075'000002                   2                  ;2
11        000076'                 .LOC    PKT+?DPC   ;TASK'S STARTING
12 00076'000007'                  NEWTSK             ;ADDRESS
13        000077'                 .LOC    PKT+?DAC2  ;NO MSG. TO TASK
14 00077'000000                   0                  ;IN AC2
15        000100'                 .LOC    PKT+?DSTB  ;START OF
16 00100'000015'                  STACK              ;STACK
17        000101'                 .LOC    PKT+?DSSZ  ;STACK LENGTH IS
18 00101'000036                   30.                ;30 WORDS
19        000102'                 .LOC    PKT+?DSFLT ;NO SPECIAL
20 00102'177777                   -1                 ;FAULT ROUTINE
21        000103'                 .LOC    PKT+?DFLGS ;NO TASK
22 00103'000000                   0                  ;SPECIFICATION FLGS
23        000104'                 .LOC    PKT+?DRES  ;OVERLAYS?
24 00104'000000                   0                  ;NO.
25        000105'                 .LOC    PKT+?DNUM  ;INITIATE ONLY
26 00105'000001                   1                  ;ONE TASK
27                ;END OF TASK PACKET
28
29                                .END    START

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS


  0003 NEWTA

ERROR 000001  XN     1/03    1/21
FLAGS 000072'        1/18    2/02
MSG   000053'        1/17    1/24
NEWTS 000007'        1/17    2/12
PKT   000073'        1/15    2/05    2/07    2/09    2/11    2/13    2/15
                     2/17    2/19    2/21    2/23    2/25
STACK 000015'        1/23    2/16
START 000000'        1/14    2/29
.ERRO 000014'        1/15    1/20    1/21
.KILL 000003  XN     1/17
.TASK 000002  XN     1/15
?KILL 005672  MC     1/16
?RETU 004600  MC     1/19
?TASK 005613  MC     1/14
?XCAL 000001         1/15    1/17    1/20
```

Figure 7-3. NEWTASK (concluded)

```
 0001  ECHO      AOS ASSEMBLER REV 01.09          16:58:37 12/05/78
01
                                    .TITL   ECHO
03                                  .EXTN   ERROR
04
05                  ;THIS PROGRAM USES ?TRCON TO READ A CONSOLE MSG.,
06                  ;AND THEN OUTPUTS THIS SAME MESSAGE TO THE CONSOLE
07                  ;IN A ?RETURN MESSAGE. AFTER INVOKING THIS PROGRAM,
08                  ;TYPE A MESSAGE IN THE FORM:
09                  ;       2,<MESSAGE><NEW LINE>
10                  ;(THE TASK'S I.D. IS 2). DO NOT EXCEED 100 CHARS
11                  ;OR 1 LINE IN THE MESSAGE.
12
13                                  .NREL
14        000003                    .TSK    3
15
16                          START:  ?TASK PKT       ;CREATE THE ECHO TASK
17 00004'002414                     JMP     @.ERROR
18                                  ?KILL           ;KILL THE INITIAL TASK
19 00007'020413             NEWTSK: LDA     0,MSG
20                                  ?TRCON          ;WAIT FOR THE MESSAGE
21 00012'002406                     JMP     @.ERROR
22 00013'024407                     LDA     1,MSG   ;NOW ECHO IT AND
23 00014'030405                     LDA     2,FLAGS ;TERMINATE THE PROCESS
24                                  ?RETURN
25 00017'002401                     JMP     @.ERROR
26
27 00020'000000$            .ERROR: ERROR
28 00021'100144             FLAGS:  ?RFCF+100.
29 00022'000046"            MSG:    .+1*2
30 00023'000144                     .BLK    100.    ;RESERVE ROOM FOR LONGER
31                  ;LINE THAN WILL HAVE ON THE CONSOLE,SO THAT TRAILING
32                  ;NULLS (INITIALIZED BY ASSEMBLER) WILL NOT INTERFERE
33                  ;WITH OUTPUT MESSAGE.
34
35                  ;TASK PACKET
36        000167'           PKT:      .LOC    PKT+?DLNK ;EXTENDED PKT?
37 00167'000001                     1                 ;NO.
38        000170'                   .LOC    PKT+?DPRI ;USE THE CALLER'S
39 00170'000000                     0                 ;PRIORITY
40        000171'                   .LOC    PKT+?DID  ;TASK I.D. IS
41 00171'000002                     2.                ;2
42        000172'                   .LOC    PKT+?DPC  ;STARTING ADDRESS
43 00172'000007'                    NEWTSK            ;OF NEW TASK
44        000173'                   .LOC    PKT+?DAC2 ;MSG.TO NEW TASK'S
45 00173'000000                     0                 ;AC2
46        000174'                   .LOC    PKT+?DSTB  ;START OF
47 00174'000202'                    STACK             ;STACK
48        000175'                   .LOC    PKT+?DSSZ ;STACK SIZE IS
49 00175'000036                     30.               ;30 WORDS
50        000176'                   .LOC    PKT+?DSFLT ;NO SPECIAL
51 00176'177777                     -1                ;FAULT ROUTINE
52        000177'                   .LOC    PKT+?DFLGS ;NO TASK
53 00177'000000                     0                 ;SPECIFICATION FLGS
54        000200'                   .LOC    PKT+?DRES  ;OVERLAYS?
55 00200'000000                     0                 ;NO.
56        000201'                   .LOC    PKT+?DNUM  ;INITIALIZE ONLY
57 00201'000001                     1                 ;ONE TASK
58 00202'000036             STACK:  .BLK    30.
59                                  .END    START

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS
```

Figure 7-4. ECHO (continues)

```
  0002 ECHO

ERROR 000001   XN     1/03     1/27
FLAGS 000021'         1/23     1/28
MSG   000022'         1/19     1/22     1/29
NEWTS 000007'         1/19     1/43
PKT   000167'         1/17     1/36     1/38     1/40     1/42     1/44     1/46
                      1/48     1/50     1/52     1/54     1/56
STACK 000202'         1/47     1/58
START 000000'         1/16     1/59
.ERRO 000020'         1/17     1/21     1/25     1/27
.KILL 000003   XN     1/19
.TASK 000002   XN     1/17
.TRCO 000004   XN     1/21
?KILL 005672   MC     1/18
?RETU 004600   MC     1/24
?TASK 005613   MC     1/16
?TRCO 006200   MC     1/20
?XCAL 000001          1/17     1/19     1/21     1/25
```

Figure 7-4. ECHO (concluded)

End of Chapter

Chapter 8
User Device Support

Servicing User Interrupts

The operating system permits a resident process to receive control upon the receipt of certain hardware interrupts. Since this ability directly conflicts with a secure multiprogramming system, users must be privileged (?PVDV) to use this facility.

On detecting an interrupt request, the system will dispatch control to the interrupt service routine by using the Device Interrupt Vector Table. In this table are pointers to Device Control Tables (DCTs) for devices established when the system was generated and pointers to DCTs built by the user for servicing interrupts generated by devices which were not established when the system was generated. (Consult *How to Load and Generate AOS* for a discussion of system generation.)

The user DCT interfaces with the system and user interrupt service. The structure and mnemonic assignments of this five-word table are:

Offset	Purpose
?UDVXM	TCB address of task that issued ?IMSG (maintained by system).
?UDVMS	Interrupt service mask.
?UDVIS	Address of interrupt service routine.
?UDVBX	Reserved system word for use in ?IXMT and ?IMSG calls. Set to 0 when you build the DCT.
?UDDRS	Address of the powerfail restart routine (if ?IDEF specifies an extended DCT), otherwise -1.

A user device and its associated interrupt service routine are defined by system call ?IDEF and removed by system call ?IRMV. The system will automatically disable both LEF mode and I/O protection on the return from the ?IDEF call; this is necessary for the user process to execute I/O instructions, since the bit pattern is the same for LEF instructions as it is for I/O instructions. if the process that issued the ?IDEF either terminates or ?CHAINs, the system will automatically remove the device.

If the user device employs either the data channel or BMC, the process must also issue the system call ?STMAP to set up the appropriate map slots.

When a user device interrupt occurs, an OR operation is performed between the interrupt mask and the mask contained in ?UDVMS of the user DCT to produce the current interrupt mask. The current interrupt mask establishes which devices, if any, can interrupt the interrupting device. While in the service routine, you (or your program) can modify the contents of the current mask with the MSKO instruction. (See *Programmer's Reference Manual, ECLIPSE® -Line Computers*, for a description of machine-level I/O concepts.)

On transferring control to the user interrupt service routine, the system will place the DCT address in AC2 and the current system mask in AC0. The system will always save the state of LEF mode and I/O protection when transferring control to the user interrupt service routine, and will disable both LEF mode and I/O protection before executing the routine.

Once in the interrupt service routine, you have complete control of the hardware. If the interrupt service routine uses floating point instructions, it is the responsibility of the user interrupt service routine to preserve and restore the state of the FPU. The only system calls that may be issued from the service routine are ?IXMT and ?IXIT; all other system calls will result in a user database trap, thereby causing the process to terminate.

The system call ?IXMT can be used to transmit a one-word message from the interrupt service routine to another task outside interrupt service.

To exit from the user interrupt service routine, issue the system call ?IXIT. Setting the contents of AC1 to zero will suppress task scheduling; setting it to a nonzero value will select rescheduling. The system will automatically restore the pre-interrupt state of LEF mode and I/O protection.

Communicating from a Service Routine Call

All multitask activity ceases at the moment a user device interrupt occurs. It is possible for a user to communicate a message to a task from an interrupt service routine. If the task in question has been expecting such a message from an ?IMSG call, it is in the suspended state. Issuing an ?IXMT from the service routine will ready that task. If no task has issued an ?IMSG, the ?IXMT call simply posts the message for the first task to issue ?IMSG. Message broadcasting by ?IXMT is not allowed, since one of the parameters specified by ?IMSG is the DCT address of the interrupt service routine which will send the message and the address of the ?IMSG task is written in ?UDVXM of that DCT by the system.

Enabling and Disabling Access to All Devices

Users with privilege ?PVDV may want to issue I/O instructions by tasks (i.e., not from user interrupt service routines). The system provides two calls to permit the task-level access of all devices, and then to disable this access to all devices. ?DEBL enables access to all devices, and ?DDIS disables access to all devices.

When access is enabled, I/O instructions can be issued to both system and user devices, but LEF mode is disabled. If you attempt to execute a LEF instruction with LEF mode disabled, results will be unpredictable. When access is disabled LEF mode is enabled, and any attempt to directly access devices will erroneously execute as a LEF instruction. This is because direct I/O (e.g., DIA) and LEF instructions use the same 16-bit patterns; only by the state of the LEF mode can the CPU distinguish them. By default, access is disabled.

Powerfail/Auto Restart

AOS will also restart user devices after a powerfail, provided you have specified an extended DCT within the ?IDEF system call. The DCT extension (offset ?UDDRS) points to a powerfail auto-restart routine. When a powerfail occurs, the system transfers control to the auto-restart routine, with the DCT address in AC2, and the current system mask in AC0.

If auto-restart is enabled (see *AOS Operator's Guide*, AOSGEN), then AOS will check to see if there are any user-defined devices that have associated powerfail restart routines. AOS will restart these devices according to their user restart routines.

(AOS enables interrupts and masks out all devices during the auto-restart sequence, so that it will recognize only powerfail interrupts.) The system transfers control to the auto-restart routine with a system mask of -1, which cannot be changed. The states of both the devices and the data channel map are undetermined after a powerfail.

Since a powerfail interrupt cannot be masked out, it is possible for a powerfail-restart sequence to occur while the system is servicing a user device interrupt. Take this into account when you write code for servicing interrupts.

AOS restores the map state of slots allocated to data channel devices. For BMC devices, the user's restart routine is responsible for restoration.

You should not disable interrupts for more than 500 milliseconds. If you do, the processor could halt abnormally before the system is able to acknowledge a powerfail interrupt and initiate an orderly shutdown.

As in the case of other interrupt service routines, you may issue only the system calls ?IXMT and ?IXIT within the auto-restart routine. The user interrupt service routine must save and restore the state of the FPU if it issues floating point instructions. When you issue ?IXIT to exit from an auto-restart routine, the contents of AC1 will not affect the task scheduler one way or another.

Data Channel and BMC Considerations

When a user device is on the data channel or BMC, you must provide additonal information in ?IDEF and ?STMAP to describe how the system should map the device to the I/O buffers in the user process address space. The ?IDEF call reserves the map slots and the ?STMAP call provides the mapping information.

You can use two methods to provide this mapping information: by using a single group of map slots allocated from the data channel A map, or by suppling an optional map definition table (MDT).

If you use the group of map slots allocated from the data channel A map, then with the ?IDEF call you need supply only the number of map slots required by the device. The system will not return the map slots which have been assigned for the device, and ?STMAP simply maps the user buffer in the allocated slots.

If you set up an MDT in your user address space, you can allocate up to eight groups of map slots from any of the four data channel maps, or from the BMC. The system will return the actual map slots allocated. Each MDT entry is two words long; the maximum size for the MDT is 16 words. If you want less than 8 entries, the last entry must be followed by a -1.

During input, the first word in each MDT entry specifies from which map the slots are to be allocated, and the first acceptable slot within that map. If the first map slot is already in use, the system tries to allocate map slots higher in the map (increasing addresses). The second word of the entry specifies how many slots are to be allocated within this group of slots. for mapping on the BMC, this number is rounded up to a mutlitple of 32 slots. On return from ?IDEF, the first word of each MDT entry contains the first map slot that the system actually allocated.

The format of the first word in an MDT is as follows:

BMC mapping:	?UDBM + first map slot
DCH A:	?UDDA + first map slot
DCH B:	?UDDB + first map slot
DCH C:	?UDDC + first map slot
DCH D:	?UDDD + first map slot

where the "first map slot" ranges from 0 to 31 (data channel) or 0 to 991 (BMC), and is the lowest map slot the system can assign.

Regardless of which method you use, the allocated map slots must be contiguous. If the system cannot find enough contiguous map slots in the specified map, it returns error ERDCH (Data Channel Map is full). ?IDEF and ?STMAP must use the same method of ampping for results to be consistent.

System Call Summary

The following calls manage user device interrupt service and provide device access:

?DDIS	Disable process access to all devices.
?DEBL	Enable process access to all devices.
?IDEF	Identify a user device interrupt service routine.
?IMSG	Receive a message from an interrupt service routine.
?IRMV	Remove user device interrupt service.
?IXIT	Exit from an interrupt service routine.
?IXMT	Transmit a message to a task outside the interrupt service routine.
?STMAP	Set the data channel map.

Modules for the execution of some of these calls are loaded into the user context. Only the ?IXMT system call can be issued from interrupt service routines, and it may not be issued outside these routines.

?DDIS

Disable access to all devices.

?DDIS
exception return
normal return

Input/Output

Input:

none.

Output:

none.

Exceptional Condition Codes in AC0

USER DEVICE codes

Description

This call disables access to all devices (enables I/O protection). The process will trap if it tries to execute any I/O instructions when the device access is disabled and the LEF mode is off. ?DEBL restores access to all I/O devices. Note that ?DDIS does not re-enable LEF mode.

Exercise discretion when using this call. Callers must have privilege ?PVDV.

?DEBL

Enable access to all devices.

?DEBL
exception return
normal return

Input/Output

Input:

none.

Output:

none.

Exceptional Condition Codes in AC0

USER DEVICE codes

Description

This call disables LEF mode, and permits I/O instructions to be issued by tasks to all system and user devices. Use discretion when using this call; callers must have privilege ?PVDV.

This call is not necessary for accessing a user device if the device has been identified to the system by ?IDEF, and I/O instructions are issued only by the user interrupt service routine.

?IDEF

Define a user device.

?IDEF
exception return
normal return

Input/Output

Input:

AC0	bit 0=0, regular length DCT.
	bit 0=1, extended length DCT (powerfail restart).
	bit 1=0, MDT not used.
	bit 1=1, MDT used for data channel or BMC.
	bits 2-15, device code (0-63 decimal).
AC1	bit 0=0, device doesn't use data channel or BMC; AC2 is ignored.
	bit 0=1, device uses data channel or BMC; AC2 has mapping information.
	bits 1-15, address of the DCT.
AC2	bits 0-15, MDT address if MDT is used (bit 1 of AC0 is 1); number of map slots if MDT is not used (bit 1 of AC0 is 0).
	AC2 is used for data channel or BMC devices only.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	modified by system.
MDT	Beginning of each group of slots is returned in the first slot field of each MDT entry. (See Data Channel and BMC Considerations in this chapter.)

Exceptional Conditional Codes in AC0

USER DEVICE codes.

?IDEF (continued)

Description

The ?IDEF call identifies a user device and its interrupt service routine to the system so that AOS can provide service for that device. The Device Control Table (DCT), the MDT, and the ACs store the device description that ?IDEF passes to the system.

An ?IDEF call with bit 0 of AC0 set specifies a "powerfail restart routine" at offset ?UDDR of the DCT. If the address of the routine is -1, then no restart is attempted (see "Powerfail/Auto Restart" in this chapter).

An ?IDEF call with bit 0 of AC1 set means the user device will use either data channel or BMC resources. Bit 1 of AC0 provides the format of the data channel or BMC request. If bit 1 is 0, then AC2 has the number of map slots requested for the data channel map. If bit 1 is 1, then AC2 has the address of the MDT (see "Data Channel and BMC Considerations" in this chapter).

This call enables you to issue I/O instructions to all devices and to receive interrupts generated by the device at the device code specified in AC0.

The number of map slots required on the data channel or BMC map is equal to the number of physical pages within the user process's address space that contain buffers involved in the I/O transfer.

Issue ?IXIT to exit from the user interrupt service routine. To specify task scheduling, set AC1 to a nonzero value when issuing ?IXIT (setting AC0 to zero suppresses scheduling).

NOTE: Upon return from the ?IDEF call, the process will have LEF mode and I/O protection turned off.

?IMSG

Receive an interrupt service message.

?IMSG
exception return
normal return

Input/Output

Input:

AC2 DCT address of device whose routine is to issue a message.

Output:

AC1 message.

AC2 unchanged.

Exceptional Condition Codes in AC0

USER DEVICE codes

Description

This call returns a message that an interrupt service routine has sent with an ?IXMT call. Additionally, ?IMSG resets the contents of the message address to all zeros; the message address must not have bit zero set. Device access must be enabled; i.e., LEF mode must be disabled.

WARNING: It is possible to enable both LEF mode and device access when you issue an ?IMSG. We caution you against this, since the results may be unpredictable.

If ?IMSG has been issued but no transmitter has yet sent a message, then the calling task remains suspended (event flag ?TSSG) until the message is sent by ?IXMT. If the message has already been sent and if the receiving task has not been suspended by ?PRSUS or ?IDSUS (event flag ?TSSP), then the task will be readied.

An interrupt service routine must be identified by ?IDEF before a task can issue ?IMSG to receive a message from that routine.

?IRMV

Remove user device interrupt service.

?IRMV
exception return
normal return

Input/Output

Input:

AC0 user device code.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

USER DEVICE codes

Description

This call removes the device's entry from the system's interrupt vector table. The system will no longer recognize interrupts from that user device, and will simply clear the interrupts where possible.

?IXIT

Exit from interrupt service.

?IXIT

Input/Output

Input:

AC1 0, suppress task scheduling.

Output:

AC1 unchanged.

Exceptional Condition Codes in AC0

No codes are defined, since there is no exception return.

Description

Issue this call to return from a user interrupt service routine. If you set the contents of AC1 to 0 before issuing this call, then task scheduling will be suppressed. Otherwise task scheduling will occur.

?IXMT

Transmit an interrupt message.

?IXMT
exception return
normal return

Input/Output

Input:

AC1 message.

AC2 DCT address of device whose routine is to issue a message.

Output:

AC1 unchanged.

AC2 unchanged.

The input state of carry is lost.

Exceptional Condition Codes in AC0

ERXMT Message address is already in use.

ERXMZ Attempt to send a zero message.

Description

This call transmits a one-word message from an interrupt service routine to a task issuing ?IMSG. The message must be nonzero, and is deposited in the mailbox specified by ?UDVBX of the DCT specified in AC2.

This call can be issued only from an interrupt service routine.

?STMAP

Set the data channel map

?STMAP
exception return
normal return

Input/Output

Input:

AC0	bit 0 must be 0.
	if bit 1 is 1 then device is using a map definition table.
	bits 2-15 hold the device code.
AC1	bit 0 must be 0.
	bits 1-15 are the user's buffer address.
AC2	contains the following flag bits (only if bit 1 of AC0 is 1):
	bits 0-3, the MDT entry number (range: 0-7).
	bits 4-9, first map slot.
	bits 10-15, number of map slots.

Output:

AC0	unchanged.
AC1	for a BMC device:
	bit 0 is 1.
	bits 1-15 are the low order 15 bits of the BMC address.
	for a data channel device:
	data channel address.
AC2	for a BMC address:
	AC2 is the high order part of the BMC address.
	AC2 is not used for a data channel address.

Exceptional Conditional Codes in AC0

USER DEVICE codes

?STMAP (continued)

Description

This call initializes slots in the data channel or BMC map for the device specified in AC0. ?STMAP also returns the first logical address of the device buffer.

?STMAP works in conjunction with ?IDEF. ?IDEF defines a user device, its I/O channel (BMC or data channel), and optionally, its map slots. ?STMAP initializes those map slots.

If you are using an MDT, then set bit 1 of AC0; AC2 will hold the information relating to the particular MDT entry. If bit 1 of AC0 is 0, then AC2 will not be used (see the section on Data Channel and BMC Considerations). Bits 0-3 and 10-15 of AC2 correspond to the entry number and the total number of map slots you defined in the MDT (with ?IDEF).

Bits 4-9 of AC2 must contain an offset from the device's first map slot. The first map slot is the number the system returned to the MDT (first acceptable slot field) as output to the ?IDEF call. Thus, if you want to initialize all the device's map slots, set bits 4-9 to 0. if you want to initialize all slots except the first slot, set bits 4-9 to 1.

End of Chapter

Chapter 9
Miscellaneous System Calls

This chapter describes those system calls which were not discussed in preceding chapters, either because they do not fall into one of the broad categories covered by those chapters or because their effect extends to more than one of these categories.

How to Get Actual Memory Locations

The system call ?AMAP allows you to get the actual physical memory pages assigned a process, should you need this information for diagnostic programs or other applications.

How to Verify a Labeled Tape Volume

The system call ?CKVOL allows you to make sure the correct labeled tape volume is mounted by comparing the volume identifier (volid) that you specify with the volid of the labeled tape that is mounted on a particular unit.

Console Interrupt Management

As described in Chapter 6, a process can allow a console operator to interrupt the execution of a program by depressing the CTRL and C keys followed by another control character on a process console. (The process console is specified in ?PCON of each process's parameter packet.) The result of the control sequence depends upon whether or not this sequence is currently enabled or disabled.

At the beginning of a program's execution, the program is interruptable by a control sequence. The program remains interruptable unless system call ?ODIS or ?INTWT is issued. These calls prevent CTRL-A sequences from having an effect until a complementary system call, ?OEBL, is issued to re-enable these interrupts.

If the operator types a CTRL-A sequence (CTRL-C followed by CTRL-A) when terminal interrupts are enabled, then one of two actions occurs, depending upon whether or not an interrupt processing task was defined. If none has been defined, then the CTRL-A has no effect. However, system call ?INTWT defines a task to monitor the process console keyboard for CTRL-A control sequences. When such a sequence is detected, this task is readied at the ?INTWT normal return; further CTRL-A sequences are ignored until a new ?INTWT call is issued.

Design of the interrupt-processing procedure is left up to the programmer. A typical procedure might issue ?IDGOTO to halt the main task's outstanding activity, output a termination message and perform miscellaneous interrupt processing, and then issue ?INTWT to re-enable and re-monitor CTRL-A sequences.

If you issue a CRTL-C CTRL-B sequence, then the process is terminated, whether or not it has been issued ?INTWT.

A CTRL-E sequence is the same as a CTRL-B sequence, except that CTRL-E creates a break file before terminating. A break file contains the memory image of the process that was interrupted.

Clock/Calendar Calls

Each system maintains a 24-hour clock and a calendar. The system maintains the current time as hours, minutes, and seconds; hours range from 0 (midnight) to 23 (11 PM). The system maintains the current date as month, day, and year. The year is expressed as the difference between the current year and 1900; thus the octal value for 1976 would be 114.

The system call to set the time of day is ?STOD; ?GTOD returns the current time of day. System call ?SDAY sets the system calendar; ?GDAY reads the calendar. Only the operator process, or one at that level, can issue ?STOD and ?SDAY.

The system maintains the clock and calendar at any real-time clock frequency, and some programming applications will need to know that frequency. System call ?GHRZ returns the current real-time clock frequency. This frequency can be set only when the system is generated.

System call ?DELAY suspends a task for a period of time expressed in milliseconds.

The system provides two calls to convert time and date values, expressed as scalar quantities, into hour, minute, second and day, month, year notations. The two system calls are ?CTOD (convert scalar to time of day) and ?CDAY (convert scalar to date). Scalar quantities for these values are returned in the file status parameter packet (?STCH/?STCL, ?STAH/?STAL, and ?STMH/?STML) by system call ?FSTAT, described in Chapter 5. The inverses of these calls are ?FTOD and ?FDAY, which convert times and dates to scalar quantities.

The ?ITIME system call returns a 48-bit value suitable for use as a timestamp.

Read a CLI Message

Each time a new process is created by a CLI command line, the CLI sends an initial IPC message to this process. ?GTMES returns CLI command line information contained in this message. This includes the number of arguments, the name of a specific argument, and switch information.

?GTMES also provides a way to access messages, sent from a creating process other than the CLI to its offspring, which are not in CLI format.

Read an Error Message File

There are 200_8 groups of error or exceptional condition codes used by the operating system, utilities and other programs running in the system. Groups 0 through 77_8 are reserved for definition by Data General Corporation; the remaining groups, numbered 100_8 through 177_8, are available for definition by users.

Each error code is a 16-bit word with two fields: a group field and an error code field. Associated with each code is a text string describing the code; the CLI uses this facility to display error messages for exceptional condition codes received during the execution of commands.

A single system file, ERMES, lists all the currently assigned code groups, and contains all the codes and their associated textual messages. System call ?ERMSG provides a convenient way to find a code in the file and retrieve its text message.

You may obtain the source version of your installation's ERMES, allocate a currently unused group (or add codes to an already allocated group), and insert a series of codes and their associated messages. Alternatively, it is possible to create a new error message file, structured like ERMES but with different contents. The structure of ERMES is described in the discussion of ?ERMSG.

Get a Program's Pathname

?GPRNM returns the pathname of the program file from which a process was initially loaded. A process can use it to get its own or another's program name.

Queue a File Entry

Several functions done system-wide, such as spooling, need to queue requests for service. If your system is running the EXEC process, this queuing is done automatically by the operating system. However, if your system is not running EXEC, it's possible to queue a file entry explicitly for spooled output (giving the name, assigned by the operator, of the spoolable device). The means for explicit queuing is system call ?ENQUE. In addition, you may queue requests with the ?EXEC call, described below, if you desire more control over the queuing parameters.

Enter the Debugger

Programs may transfer control to the debugger at any time by issuing system call ?DEBUG. Applications for this call include using it as a breakpoint, or using it to transfer control from an error processing routine to the debugger upon detection of an unknown error condition. Control can then be sent from the debugger back to the program by using the debugger P command.

Examining and Changing the Bias Factors

The operating system has a minimum and a maximum bias factor. The minimum bias factor is the number of noninteracting swappable jobs that the system tries to keep in memory at all times. The maximum bias factor is the number of noninteractive swappable jobs that the system will normally allow in memory at one time. The bias range may influence the response time of interactive swappable processes in the same system. Bias range selection criteria are described in the *AOS System Manager's Guide*. System call ?GBIAS returns the current bias range; ?SBIAS sets a new bias range. Only the system operator can issue ?SBIAS.

Network Calls

The ?HNAME and ?RNAME system calls are useful only if your AOS system is part of a XODIAC network (see *XODIAC Network Management System User's Manual)*. The ?HNAME call allows you to obtain host ID and name information about both local and remote hosts in your network. You can use ?RNAME to determine the network location of a filename.

Issuing a Request to EXEC

The EXEC utility provides a means for controlling user access in a time-sharing or batch environment. The *AOS Operator's Guide* describes EXEC in detail. User programs needing to request a service from EXEC must issue system call ?EXEC. The CLI, for example, issues ?EXEC to implement its MOUNT command. Services currently provided by ?EXEC are:

- assign a logical name to a tape unit (to be accessed as an uninitialized whole), and issue an operator mount message;
- deassign a logical name so that the operator can dismount a tape reel;
- place a request into a queue;
- hold, unhold, or cancel a queue request;
- get EXEC status information.

Using the Floating Point Unit

Before any task uses the floating point unit (FPU), that task must initialize the FPU by reserving an 18-word area to save the FPU state. This area is reserved by issuing system call ?IFPU. You can reserve this area, known as the floating point status register, by issuing the system call ?IFPU. Even single-task programs must use ?IFPU before using the floating point unit, since this system call also clears any existing data in the floating point status register. If any task using FPU fails to issue ?IFPU, FPU results obtained by this task will be indeterminate.

Managing the System Logging File

The operating system maintains a special file, with pathname :SYSLOG and its own file type ?FLOG, in which messages are logged by the EXEC (if it is running), and the system operator. The system calls ?SYLOG and ?LOGEV are provided to aid in managing this file. The system logging file is further explained in the *AOS Operator's Guide* and the *Command Line Interpreter (CLI)(AOS and AOS/VS) User's Manual.*

Current AOS Status

Since the Advanced Operating System may go through several revisions between each major release, it is useful to be able to determine which revision is currently running. The system call ?SINFO enables you to get this and other selected information.

System Call Summary

The following list summarizes the system calls described in this chapter:

?AMAP	Get actual memory allocated to a process.
?CDAY	Convert scalar to date.
?CKVOL	Check volume identifier of a labeled magnetic tape.
?CTOD	Convert scalar to time.
?DEBUG	Enter the debugger.
?DELAY	Suspend a task for a specific interval of time.
?ENQUE	Queue an entry.
?ERMSG	Read an error message.
?EXEC	Request a service from EXEC.
?FDAY	Convert date to scalar.
?FTOD	Convert time of day to scalar.
?GBIAS	Get the bias range.
?GDAY	Get the current date.
?GHRZ	Get the current real-time clock frequency.

?GPRNM	Get program name.
?GSID	Get the system identifier.
?GTMES	Read a CLI message.
?GTOD	Get the time of day.
?HNAME	Obtain host ID/name information.
?IFPU	Initialize the floating point unit.
?INTWT	Define a console interrupt task.
?ITIME	Return a 48-bit timestamp.
?LOGEV	Log an event in the log file.
?ODIS	Disable subsequent control sequences.
?OEBL	Enable subsequent control sequences.
?RNAME	Determine network location of a file.
?SBIAS	Set the bias range.
?SDAY	Set the system calendar.
?SINFO	Get selected information on the current AOS.
?SSID	Set the system identifier.
?STOD	Set the time of day.
?SYLOG	Manipulate the system log file.

?AMAP

Get actual memory allocated a process.

?AMAP
exception return
normal return

Input/Output

Input:

AC0	byte pointer to process name, process ID, or -1 if self.
AC1	0 if AC0 contains process ID. -1 if AC0 contains a byte pointer to process name.
AC2	bit zero is a context indicator: 0 if primary context, 1 if ghost context. bits 1 through 15: word pointer to a 32-word array.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERMPR	System call parameter address error.
ERPNM	Illegal process name.
ERPRM	Attempt to refer to a process not in the hierarchy.

Description

This system call allows a process to determine the actual physical memory pages allocated to itself or another process. This is intended primarily as an aid to writing diagnostic programs.

The 32-word array pointed to by AC2 will be filled in with the physical pages of memory allocated to the corresponding logical pages of the specified process. Logical pages that do not have physical pages allocated will be indicated by -1 (invalid).

The target process of this call should be resident at the time the call is made. If this call is made on a swappable process which is swapped out, then no memory is assigned to the process and all page numbers will be returned as invalid. Remember that the physical pages assigned to a swappable process may vary every time that that process is swapped out and back in.

?CDAY

Convert a scalar to a date.

?CDAY
exception return
normal return

Input/Output

Input:

AC0 number of days since December 31, 1967.

Output:

AC0 day in month.

AC1 month in year.

AC2 year minus 1900.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call accepts a scalar value denoting a number of days since December 31, 1967 and coverts this into the equivalent day, month, and year values. The scalar value is returned in the ?FSTAT parameter packet, offsets ?STCH, STAH, and ?STMH.

?CKVOL

Check volume identifier of a labeled magnetic tape.

?CKVOL
exception return
normal return

Input/Output

Input:

AC0	byte pointer to the unit's pathname.	
AC1	byte pointer to the volume identifier (volid) string.	
AC2	flag word:	
	?XMFI=1,	labels are in EBCDIC
	?OPDL=1,	density mode is 800 bpi
	?OPDM=1,	density mode is 1600 bpi
	?OPAM=1,	automatic density mode selection

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERVOL Incorrect labeled tape volume mounted
FILE SYSTEM codes

Description

?CKVOL checks the volume identifier (volid) of a labeled magnetic tape against a volid that you (the user) supply.

?CKVOL compares the volid that AC1 points to with the volid of the labeled tape that is mounted on the unit whose pathname AC0 points to. If these two volids are not the same, AOS returns error code ERVOL in AC0. If the volids are the same, AOS takes the normal return.

?CKVOL allows you to make sure the correct labeled tape volume is mounted before you read from or write to the tape.

?CTOD

Convert a scalar to a time of day.

?CTOD
exception return
normal return

Input/Output

Input:

AC0 half the number of seconds since midnight.

Output:

AC0 seconds.

AC1 minutes.

AC2 hours (0-23).

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call accepts a scalar value denoting half the number of seconds since midnight and converts this into the equivalent hour, minute, and second. The scalar value is returned in the ?FSTAT parameter packet, offsets ?STCL, ?STAL, and ?STML.

?DEBUG

Enter the debugger.

?DEBUG
exception return
normal return

Input/Output

Input:

none.

Output:

unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This call transfers control to the debugger. You can return control to this call's normal return by issuing a Debug *proceed* (P) command.

?DELAY

Suspend a task for a specified time.

?DELAY
exception return
normal return

Input/Output

Input:

AC0, AC1 — delay interval in milliseconds.

Output:

AC0, AC1 — modified (used by system to maintain delays).

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This system call suspends the calling task for an interval of time in milliseconds, specified as a double precision number.

CAUTION: If the specified number of milliseconds is not a multiple of the real-time clock period, then the delay interval will be rounded up to the next higher multiple. For example, if the clock frequency is 10 Hz (period equals 100 ms), and a 120 ms. delay is specified, the actual delay will be 200 ms.

The actual clock frequency can be found by issuing system call ?GHRZ.

?ENQUE

Queue an entry.

?ENQUE
exception return
normal return

Input/Output

Input:

AC0 byte pointer to name of queue.

AC1 byte pointer to specifications string.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This call queues a file entry to a queue of another process using the IPC mechanism. For example, you can queue an entry to a spooler output queue on a system without the EXEC process. To do so, you create a specifications string which names the entry to be queued, along with switches modifying the way it is to be output. The format of this string is as follows:

pathname *[/switch]...*

The string can be up to 512 characters long, and the terminating character must be a null. Any switch argument such as a space, which would ordinarily terminate the switch, can be passed literally by setting the high order bit in this character.

Table 9-1 lists defined switches; the absence of a switch produces the effects which are listed as default:

Table 9-1. Spooler Specification Switches

Switch	Effect	Default
/B	Output in binary mode.	Output in text mode.
/D	Delete file after output.	Do not delete after output.
/H	Output header at top of each page.	No page header is output.
/MES= message	Output specified message to operator console.	No message output.
/P	Pause for operator response before outputting the file.	No pause.

If /P (and no /MES) is specified, then before output is started, PAUSE FROM *spoolable devicename* is displayed on the operator console.

If /MES (and no /P) is specified, then before output is started, the message defined by the switch is displayed on the operator console, prefixed FROM *spoolable devicename.*

If both /MES and /P are specified, then before output is started, the message defined by /MES is displayed on the operator console prefixed by PAUSE FROM *spoolable devicename:* A typical message might be PAUSE FROM LPA: TWO-PART FORMS.

The operator must respond with an appropriate control command, such as **CONTROL @SPOOL CONTINUE**, to start output of the file. These responses are described in the *AOS Operator's Guide.*

?ERMSG

Read an error message file.

?ERMSG
exception return
normal return

Input/Output

Input:

AC0	error code.
AC1	left byte: byte size of message buffer. right byte: channel number; 377 specifies system default.
AC2	byte pointer to message buffer.

Output:

AC0	byte length of message.
AC1	channel number.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERTXT Buffer too small for message, no message received.

FILE SYSTEM codes

Description

This call returns the textual message associated with a given error code in an error message file. If this message is found in ERMES, the standard error message file provided with the system, then specify 377 as a channel number (the system assigns a channel). If a file other than ERMES is desired, then it must be opened and its channel number given in AC1. The actual channel used is indicated in AC1 on output. If an undefined error code is passed to ?ERMSG, control takes the normal return and the message UNKNOWN ERROR CODE *n* is placed in the buffer. All error codes and message files must have the same structure as ERMES (described in Figure 9-1).

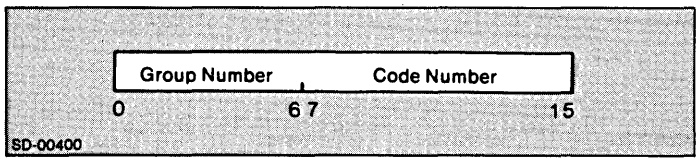


Figure 9-1. Error Code Structure

Groups range from 0 to 177_8 ; error codes range from 0 to 777_8. Groups 0 through 77_8 are reserved for definition by Data General Corporation; remaining groups 100_8 through 177_8 are open for user definitions.

Error (or exception) codes and their messages are removed or created by editing, assemblng, and binding the error message file, ERMES.SR. Two macros, GRP and CODE let you add new codes or groups to the file. The GRP macro defines the start and size of a new error group. The format of this macro is as follows:

```
GRP m n
```

where m is the group number and n is the maximum number of error codes in that group.

Thus GRP 0 200 defines a maximum of 200 error codes for error group zero.

Error codes and their associated text messages are inserted into each group using the CODE macro, which has the following format:

```
CODE code mnemonic
.TXT *message string*
```

Thus the following macro,

```
CODE ERIFC
.TXT *ILLEGAL FILENAME CHARACTER*
```

associates the error message ILLEGAL FILENAME CHARACTER with error condition ERIFC.

?EXEC

Request a service from EXEC.

?EXEC *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 address of ?EXEC parameter packet.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERESO Attempt to issue this call from a process that was not a son of EXEC.

ERRBO The operator refused this request.

* ERWMT Attempt to dismount a reel of tape that was not mounted.

ERXNA EXEC not available in this system.

ERXUF You requested an unknown function (packet offset ?XRFNC).

FILE SYSTEM codes

IPC SYSTEM codes

Description

This system call requests EXEC to perform one of several functions for the calling process, which must be a son created by EXEC. The ?EXEC call can currently perform five functions:

1. mount a tape (labeled or unlabeled);
2. dismount a tape (labeled or unlabeled);
3. place a request into a queue;
4. hold, unhold or cancel a queue request;
5. obtain EXEC status information.

There is a unique parameter packet for each of these functions. The first word of the packet, ?XRFNC, signals which type it is to the system.

Mounting and Dismounting a Tape

To mount an unlabeled tape, you must set the first word of the ?EXEC packet (?XRFNC) to either ?XFMUN or ?XFXUN. To mount a labeled tape, you must set the first word to either ?XFMLT or ?XFXML. Then, to dismount either a labeled or an unlabeled tape, set the ?XRFNC word to ?XFDUN. The logical name of the tape, to which the packet points, must be a simple, unique name within the caller's initial working directory. EXEC will take this name and create a link entry of the form :UDD:<username>:<linkname> in the caller's initial working directory. The resolution of this link is either @MTxxx for unit mounts, or @LMT:volid for labeled mounts.

The offsets ?XMUT and ?XMLT are byte pointers to a required operator request text string. The request string can be a maximum of 80 characters long, including the delimiter. The delimiter can be either a new line, carriage return, form feed, or null. You should include some or all of the following types of information in the operator request text:

- the name of the tape reel (if other than the volume name);
- the location where the tape reel is stored;
- the type of unit to use (high density, 7 or 9 track, etc.).

Tables 9-2 and 9-3 illustrate the ?EXEC packet for two unlabeled mount functions.

Table 9-2. ?EXEC Packet for ?XFMUN and ?XFXUN Unlabeled Mount Functions

Offset	Contents
?XRFNC	Function code, either ?XFMUN or ?XFXUN.
?XMUL	Byte pointer to the logical name to be created in the user's initial directory.
?XMUT	Byte pointer to the operator request text.

Table 9-3. Additional Packet for ?XFXUN Function

Offset	Caption
?XMUQ	Reserved.
?XMUR	Reserved.
?XMUF	Flags word - see description.
?XMUS	Reserved.
?XMUE	Reserved.

Tables 9-4 and 9-5 illustrate the ?EXEC packet for the labeled mount functions ?XFMLT and ?XFXML.

Table 9-4. ?EXEC Packet for ?XFMLT and ?XFXML Labeled Mount Functions

Offset	Contents
?XRFNC	Function code, either ?XFMLT or ?XFXML.
?XMLL	Byte pointer to the logical name to be created in the user's initial directory.
?XMLT	Byte pointer to the operator request text.
?XMLV	Byte pointer to list of volids, each separated by a <0>. The list is terminated by <0><0>.

Table 9-5. Additional Packet for ?XFXML

Offset	Contents
?XMLR	Reserved.
?XMLF	Flags word -- see description.
?XMLS	Reserved.
?XMLE	Reserved.

Table 9-6 illustrates the bits of the flag word for ?EXEC functions ?XFXUN and ?XFXML.

Table 9-6. Functions ?XFXUN and ?XFXML (for ?EXEC) Bits

Bit	Description
0	?XMFC - reserved.
1	?XMFI - if set, the tape is in IBM format.
2 3 4	?OPD0, ?OPD1, ?OPD2 - density to use. This is only applicable to tape drives capable of software selection of the density. See ?OPEN for the setting of the bits.
5	?XMFR - if set, ?EXEC will assume the tape to be mounted is to be read only and will set the ACL to USER, RE.
6-15	Reserved.

Table 9-7 illustrates the packet used to dismount a tape.

Table 9-7. EXEC Packet for Dismounting a Tape

Offset	Contents
?XRFNC	?XFDUN - dismount function code.
?XDUL	Byte pointer to the logical name or unit to be dismounted.
?XDUT	Byte pointer to a text string explaining to the system operator what to do with the dismounted tape reel. 0 indicates no message for the operator.

As with mount packet, the string passed by ?XDUT cannot exceed 80 characters (including the delimiter). You can specify no message by setting ?XDUT to zero.

Placing a Request into a Queue

The queue request packet is illustrated in Table 9-8. Note that the first word, ?XRFNC, can be any one of four values given by the indicated mnemonics, depending on the type of queue desired. Unless ?XFSUB is chosen here, the queue name pointed to in offset ?XTYP must match the type chosen in ?XRFNC.

Table 9-8. ?EXEC Packet for Placing a Request into a Queue

Offset	Contents
?XRFNC	One of four values: ?XFSUB Place an entry into any type of queue. ?XFLPT Place an entry into a print queue. ?XFPTP Place an entry into a punch queue. ?XFPLT Place an entry into a plot queue. ?XFHAM Place an entry into a HAMLET queue. ?XFFTA Place an entry into an FTA queue.
?XTYP	Byte pointer to a queue name. Unless you specified ?XFSUB, the queue must be of the type indicated above.
?XDAT	(Returned.) Date queued in file system format.
?XTIM	(Returned.) Time queued in file system format.
?XLMT	Resource limit. If yours is a print entry, this indicates the maximum number of pages to print. If you default to zero, the EXEC will estimate for you by taking the number of bytes in the file, dividing by 1000 decimal, discarding the remainder and adding four. If yours is a batch entry, the maximum number of biseconds (i.e., number of seconds divided by two) of total CPU time allowed this batch, including the CLI and all other processes created by the the batch job. If you default to zero, the EXEC will estimate a value of 30 decimal, i.e., 1 minute. These limits must be supplied but will only be enforced if the operator has enabled the "limit" feature. For all other queue types, this word must be set to zero.
?XPRI	Entry priority. The priority must be between 255 (lowest priority) and the maximum as specified in your user profile. You may default with a -1, in which case the EXEC will assign you a priority in the middle of your range and return that value here.
?XFGS	Flag word. Possible flags are: ?BMBT Print file in binary mode. ?BMDA File specified in offset ?XPBP to be deleted after processing completes. You must have owner and/or write access to the file. ?BMFO Print file with "fold long lines" option (see QPRINT command in the *Command Line Interpreter (CLI) (AOS and AOS/VS) User's Manual).* ?BMNO Send a message back to your console when the entry has been completed. If you log off and then you or another user logs on, or if your console is not controlled by the EXEC, no message will be sent. ?BMNR Entry not to be restarted if the system or the EXEC crashes while the entry is being processed. ?BMOP Entry to be started only if there is a system operator on duty. This flag should always be used if the entry is for a batch job which uses the mount/dismount facility. ?BMRA This flag determines how the offsets XAFD and ?XAFT will be interpreted (see below). ?BMSH Entry to have "sequence number held by user" set. ?BMTI Print file with "title" option (see QPRINT command in the *Command Line Interpreter (CLI) (AOS, AOS/VS) User's Manual).*

(continues)

Table 9-8. ?EXEC Packet for Placing a Request into a Queue

Offset	Contents
?XSEQ	(Returned.) Sequence number assigned to the entry by the EXEC.
?XRES	Reserved - must be zero.
?XFBP	If the queue type is batch, byte pointer to a jobname. For other queue types, byte pointer to a forms type. Jobnames need not be unique. Forms types must have been defined and made accessible to you by the system operator.
?XPBP	Byte pointer to the pathname of a file to be processed. The pathname must be complete, beginning from the root, i.e., it must begin with a colon. For FTA queues: byte pointer to full pathname of the source file.
?XAFD and ?XAFT	Interpretation depends on flag bit ?XFRA in offset ?XFGS. If ?XFRA *was not* set: ?XAFD read as the date (in file system format) after which the entry may be processed. ?XAFT read as the time (in file system format) after which the entry may be processed. If ?XFRA *was* set: ?XAFD read as a number of whole days after which entry may be processed. ?XAFT read as a number of bi-seconds (two second intervals) after which entry may be processed. EXEC adds the time periods specified in ?XAFD and ?XAFT, starting from the time the queue request was made, and begins processing the entry after the designated time period has elapsed. The system clears the ?XFRA bit and returns the absolute time in ?XAFT and ?XAFD.
?XXW0	For print queues: if nonzero, the print spooler will print the pathname of the file, the date and time of printing, and a page number on the first printable line of each page. The second printable line of the page will be skipped, and actual printing will begin on line three. If this word is zero, printing will begin with line one of each page. For all other queue types: this word must be zero. For batch queues: byte pointer to the full pathname of an existing file to be used as the generic output file. EXEC will neither print nor delete this file. Use zero to specify the *default* generic output file. EXEC creates, prints, and then deletes the default file :QUEUE:USER.OUTPUT.SEQ.EXEC omits the printing step if the file is empty. For HAMLET queues: the ?XXWO value, when in the range 0-7, specifies the HAMLET stream number. Otherwise, EXEC uses the first available stream. For HASP II emulator queues: the ?XXW0 value specifies the HASP II stream number if it is in the range 0-7; otherwise, EXEC uses the first available stream. For FTA queues: byte pointer to full pathname of the destination file.
?XXW1	For print queues: if nonzero, the print spooler will continue onto a new line all lines which are too long to fit onto the form. If zero, any line which doesn't fit will be truncated. For batch queues: byte pointer to the full pathname of an existing file to be used as the generic list file. EXEC will neither print nor delete this file. Use zero to specify the *default* generic list file. EXEC creates, prints, and then deletes the default file :QUEUE:USER.LIST.SEQ. EXEC omits the printing step if the file is empty. For HAMLET queues: the XW1 value specifies the stream type where the value of 1 is of type line, 2 is of type sequential, 3 is of type EBCDIC, and 4 is of type binary. For all other queue types: this word must be zero.

(continued)

Table 9-8. ?EXEC Packet for Placing a Request into a Queue

Offset	Contents
?XXW2	For print queue types number of copies to be made by the appropriate spooler. If 0, 1 is assumed. For batch queues: this word must be 0. For HAMLET queues: this word must be 0.
?XXW3	For print and batch queues: byte pointer to a text string (destination). This text string will appear in block letters at the top of any header or trailer pages. Use a zero to specify the default username as the destination. For HAMLET queues: this word must be 0.

(concluded)

A maximum of 256 queue entries is allowed at any one time. When the EXEC queue becomes full, the system will not honor subsequent queue requests, and will return the error code message, The Queue Is Full.

On a print or batch queue request, the operator may have enabled the "limit" feature, requiring the queue request to specify a maximum number of pages for a print job or a maximum amount of CPU time for a batch job. Offset ?XMLT must supply this value, or else a default of zero is used which causes the EXEC to estimate the value for you. For print jobs, the limit is computed by the length of the file in bytes divided by 1000. + 4. For batch jobs, the limit is set to 60. CPU seconds.

Note that in flag word ?XFGS, whether or not bit ?XFRA is set determines how words ?XAFD and ?XAFT are interpreted.

Offset ?XFBP must be supplied with a byte pointer to a jobname for batch queues, or a byte pointer to a forms for print type requests. In either case, the name must be a legal filename or null. Jobnames need not be unique. If forms are being specified, the system operator must have defined the requested forms and must have allowed you to use them. Valid forms at your installation can be listed by using the CLI command ACL/V :UTIL:FORMS:+; you may use any form for which the Access Control List allows you read access.

Holding, Unholding, and Cancelling Queue Requests

You may use the EXEC to hold, unhold or cancel any queue entries which you have created. The parameter packet for this application is described in Table 9-9.

Table 9-9. ?EXEC Packet for Holding, Unholding, or Cancelling Queue Requests

Offset	Contents
?XRFNC	One of the following values: ?XFHOL Hold the specified entry. ?XFUNH Unhold the specified entry. ?XFCAN Cancel the specified entry.
?XFP1	-1 if ?XFP2 contains a byte pointer, anything else if ?XFP2 contains a sequence number.
?XFP2	Byte pointer to jobname (or sequence number).

Queue entries are held and unheld by their hold bits. Each queue request has two hold bits accessible only to the system operator. In addition, all queue entries have a sequence number hold bit, and batch entries have an extra jobname hold bit. If any hold bit is set, an entry will not be selected for processing.

?EXEC (continued)

If you supply a jobname in the buffer pointed to by offset ?XFP2, only the batch queue will be searched, and only the jobname hold bit will be changed. If you supply a sequence number in this buffer, all queue entries will be searched and the sequence number hold bit will be changed. In addition, for batch entries, the jobname bit will also be changed.

Jobnames and sequence numbers are not unique; you may possibly have several requests with the same jobname or sequence number. If so, all matching entries will be affected by this call.

If you cancel an entry, all possible hold bits, including the operator bits, will be cleared and a special "cancelled by user" bit will be set.

These functions will not affect any entry which is currently being processed.

Obtaining EXEC Status Information

The ?EXEC call may be used to obtain information on the status of the EXEC. The parameter packet for this application is shown in Table 9-10. The call will return information in offset ?XFP1, and if you supply a byte pointer to a 32-decimal byte area in offset ?XFP2, that area will also be filled in with information. A default of 0 inhibits this option.

Table 9-10. ?EXEC Packet for Obtaining EXEC Status Information

Offset	Contents
?XRFNC	?XFSTS
?XFP1	(Returned.) Bit zero: 1 if operator on duty, 0 if system unattended. Bit one: 1 if logged on in batch mode, 0 if logged on at a console or not at all. Right byte: 0 if not logged on at all, or the I.D. of the process subtree which is immediately subordinate to the EXEC.
?XFP2	Byte pointer to a 32-decimal byte area for return of the console name or the stream name if you are logged on under EXEC. The console name returned will not contain the at sign (@). If you supply a zero in this offset, no string will be returned.

Table 9-11 lists the EXEC functions and their parameters.

Table 9-11. EXEC Functions and Their Parameters

Type of Request	?XLMT	?XFGS	?XFBP	?XPBP	?XWO	?XW1	?XW2	?XW3
(QSUBMIT) ?XFSUB	No.	all	(BP) jobname	(BP) file to submit*	No./(BP) QOUTPUT	No./(BP) QLIST		No./(BP) destination name
(QPRINT) ?XFLPT	max No. of pages	all	(BP) forms+	(BP) file to print*	beginning page No.	end page No.	No. of copies	(BP) destination name
(QPLOT) ?XFPLT		all but ?BMBI ?BMTI ?BMFO	(BP) forms+	(BP) file to plot*	0	0	No. of copies	0
QPUNCH) ?XFPTP	Max No. of feet	all but ?BMBI ?BMIJ ?BMFO	(BP) forms+	(BP) file to punch*	0	0	No. of copies	(BP) destination name
XFHAM	0	all but ?BMBI ?BMTI ?BMFU	0	(BP) file to transfer*	stream No. (0-7)	type of transfer (0-4)	0	0
(QFTA) ?XFFTA	0	all but ?BMBI ?BMTI ?BMFO ?BMDA	0	(BP) source file name*	stream No. (0-7)	(BP) destination file name*	FTA flags	compression flag

& See *HASP Workstation Emulator User's Manual* for details.

\+ Must be a simple filename (no : in name).

* Must be complete pathname (start with : or @).

?FDAY

Convert a date to a scalar.

?FDAY
exception return
normal return

Input/Output

Input:

AC0	day of the month.
AC1	month of the year.
AC2	year minus 1900.

Output:

AC0	number of days since December 31, 1967.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERPRE	Invalid system call parameters.

Description

This call is the inverse of ?CDAY. It takes a date in day/month/year format (where 1901 is year one) that you load into accumulators 0, 1, and 2, respectively, and returns the number of days since December 31, 1967 in AC0. The permissible parameters for this call are legal dates from January 1, 1968 (1/1/68) to December 31, 2099 (31/12/199). If you need to calculate day of the week from the output of this call, divide by seven and note the remainder. January 1, 1968 (and every other day with a remainder of one) was a Monday.

?FTOD

Convert time of day to a scalar.

```
?FTOD
exception return
normal return
```

Input/Output

Input:

AC0	seconds.
AC1	minutes.
AC2	hours (0-23).

Output:

AC0	half the number of seconds since midnight.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERPRE	Invalid system call parameters.

Description

This call is the inverse of ?CTOD. It takes the time in seconds, minutes, and hours (from 0 to 23) that you load into accumulators 0, 1, and 2, respectively, and returns the number of biseconds since midnight (i.e., the number of seconds since midnight divided by two) in AC0. Permissible parameters for this call are any legal times from 0:00:00 to 23:59:59.

Note that ?FTOD rounds up before converting to biseconds (e.g., 3 seconds becomes 2 biseconds).

?GBIAS

Get the bias range.

?GBIAS
exception return
normal return

Input/Output

Input:

none.

Output:

AC0 left byte: maximum bias factor.

right byte: minimum bias factor.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

If there is no defined bias range, then 0 is returned in AC0. A maximum bias factor of 0 denotes there is no effective maximum. For more information about the use of bias factors, consult the ?SBIAS call and the *AOS System Manager's Guide.*

?GDAY

Get the current date.

?GDAY
exception return
normal return

Input/Output

Input:

none.

Output:

AC0 day of the month (1 to 31).

AC1 month of the year (1 to 12).

AC2 current year (less 1900).

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

The system call returns the values of the current day of the month, month of the year, and current year (less a 1900 offset).

?GHRZ

Get the real-time clock's frequency.

?GHRZ
exception return
normal return

Input/Output

Input:

none.

Output:

AC0 code indicating the frequency, as follows:

Code	Frequency (Hz)
0	60
1	10
2	100
3	1000
4	50

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This system call returns the real-time clock frequency which was set when the operating system was generated.

?GPRNM

Get a program name.

?GPRNM
exception return
normal return

Input/Output

Input:

AC0 PID or -1 (calling process).

AC2 byte pointer to a 256-byte buffer.

Output:

AC0 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERPRH Attempt to refer to a process not in the process tree.

DISK ERROR codes

Description

This call returns the complete pathname of the file on disk which was initially loaded into the address space of the process specified in AC0. If -1 is input in AC0, then the pathname of the caller's program is returned.

?GSID

Get the system identifier.

?GSID
exception return
normal return

Input/Output

AC2 byte pointer to 32-byte buffer into which the system identifier will be copied.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERMPR System call parameter address error.

Description

This system call returns the character string used as the current system identifier. The string is up to 32 characters long and is terminated with a null.

?GTMES

Get a CLI message.

?GTMES *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 address of message packet.

Output:

AC0, AC1 contents depend upon request type (see Table 9-8).

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

IPC codes

MISCELLANEOUS codes

Description

The CLI sends an initial IPC message to each new process. This message contains an edited version of the original CLI command and is truncated to its first 512 bytes. The ?GTMES call provides a convenient way to examine and retrieve portions of the CLI message. This call can also get an IPC message sent by a father process other than the CLI to its son.

The parameter packet format is shown in Figure 9-2 and described in Table 9-12.

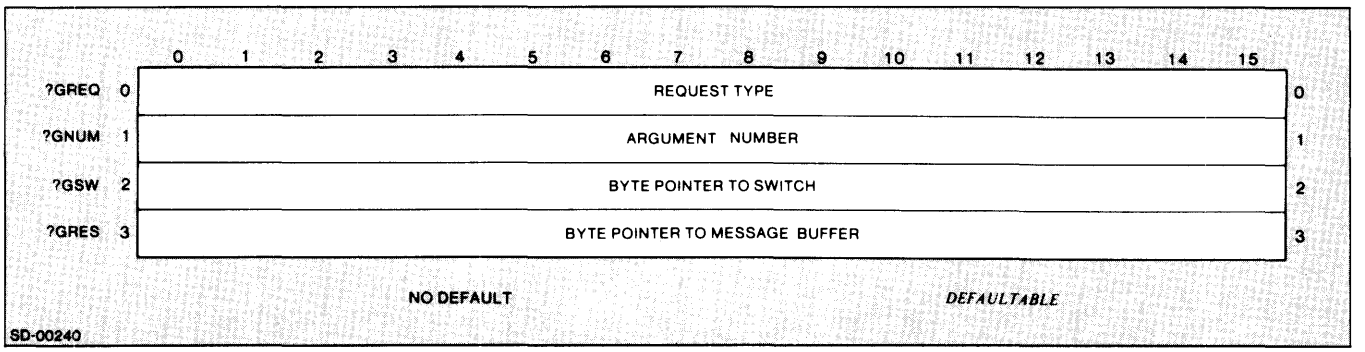


Figure 9-2. ?GTMES Parameter Packet

Table 9-12. ?GTMES Parameter Packet

Offset	Contents
?GREQ	Type of request. Types are shown in Table 9-13.
?GNUM	Argument number. Argument number zero indicates the program name, provided that CLI format is returned.
?GSW	Byte pointer to the name of a simple or keyword switch (used for ?GTSW type calls only). This switch can be no longer than 32-bytes (including trailing nulls) and must not contain any lowercase characters.
?GRES	Byte pointer to the area which will receive the result; -1 if no such area. The returned string is always terminated by a null.

Offset ?GREQ can contain one of a variety of types of requests. These requests and their names are given in Table 9-13.

Table 9-13. ?GTMES Request Types for Offset ?GREQ

Name	Meaning
?GMES	Copy the entire message to the area specified by ?GRES (unless ?GRES contains -1). The word length of the entire message is returned in AC1; message flags are returned in AC0. The message flag is: ?GFCF Message is in CLI format. This means there is an argument zero, which is the program name.
?GCMD	Read the edited CLI command line into the ?GRES buffer (if one has been specified). The byte length of the edited CLI command line is returned in AC1.
?GCNT	Get the argument count. The number of arguments, excluding the program name, is returned in AC0.
?GARG	Copy the argument minus switches into the area specified by ?GRES, if any. If the argument specified by ?GNUM consists entirely of decimal digits, its binary equivalent is returned in AC1 (otherwise -1 is returned) with a maximum allowable value of 65,535. The argument's byte length is returned in AC0.
?GTSW	Test a switch to see whether the argument indicated by ?GNUM has the switch pointed to by ?GSW. The test result is returned in AC0: -1 switch was not found. 0 simple switch was found. >0 byte length of keyword switch. If the switch is a keyword type and an area is specified by ?GRES, then the value of the switch (terminated by a null) is returned there. If the key word value consists entirely of decimal digits, its binary equivalent is returned in AC1 (otherwise -1), with a maximum allowable value of 65,535. (If the value is greater, an error is returned.) If the argument being checked has two occurrences of the same switch, ?GTMES only returns information pertaining to the first occurrence of the switch.
?GSWS	Set switch flags in AC0 and AC1. The argument specified by ?GNUM is examined for single character simple or keyword alphabetic switches. A bit is set in AC0 and AC1 for each switch that is found. Bit 0 of AC0 is set for A, bit 15 of AC0 is set for P, bit 0 of AC1 is set for Q, etc. After Z, AC1 bit 9, no bits are set.

If the request type provided in offset ?GREQ is any other than ?GMES (copy the entire message), the message from the CLI (or other father process) will have this format:

1. All arguments will be separated by single commas only.
2. The last character will be a null, and the message will be word aligned, i.e., filled out with a second null (if necessary to insure an even number of bytes).
3. These characters will not occur in the message: Space < > [ ] ( ) ; NEW LINE, carriage-return, form feed, or any imbedded nulls.
4. The high order bit of every byte will be zero, i.e., not used for parity.
5. Lowercase characters will be replaced with uppercase characters.

Argument numbers in offset ?GNUM range from 0 (the program name) to *n*, where *n* is the number of arguments in the command line. The string that the contents of offset ?GSW points to should be terminated with a null character not a slash; e.g., the name of the /L switch is L<0>. If no receiving area is specified in ?GRES, then the ?GTMES call will return information in AC0 and AC1 only (see Table 9-14).

A summary of values returned in AC0, AC1, and in the ?GRES area is given in Table 9-14.

Table 9-14. Parameters Returned by ?GTMES

Request Type	AC0	AC1	Contents of ?GRES area
?GMES	Message flags.	Total word length of message.	Entire message.
?GCMD	Unchanged.	Byte length of edited CLI command.	CLI command line.
?GCNT	Number of arguments (excluding program name).	Unchanged.	n/a
?GARG	Byte length of argument (excluding terminating null).	Binary equivalent if argument is decimal.* Otherwise, -1.	Actual argument string.
?GTSW	-1 switch not found. 0 simple switch. > 0 byte length of keyword switch.	Binary equivalent if keyword value is * decimal. Otherwise, -1.	Actual keyword switch value.
?GSWS	1B0 means /A 1B1 means /B . . . 1B15 means /P	1B0 means /Q 1B1 means /R . . . 1B9 means /Z bits 10-15 zeroed	n/a

* An error occurs if the argument or switch value is all decimal digits with a value greater than 65535. If you want to perform your own conversion, you may ignore the error.

AC0 contains the error code ERISV.

AC1 contains the length of the argument or switch value.

?GRES area contains the actual string.

?GTMES (continued)

Selecting ?GCMD reads the edited CLI command into the ?GRES buffer. If the command name was XEQ, EXECUTE, DEBUG, PROCESS, or CHAIN, then the command name is removed.

The information returned in the ?GRES buffer has a terminating null.

Use ?GTMES to get messages which are not in CLI format. By specifying no receiving area (-1 in ?GRES), a ?GMES call type returns flags in AC0 indicating whether or not the message is in CLI format.

?GTOD

Get the time of day.

?GTOD
exception return
normal return

Input/Output

Input:

none.

Output:

AC0	seconds (0-59).
AC1	minutes (0-59).
AC2	hours (0-23).

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This system call returns the current time of day.

?HNAME

Obtain Host ID/Name Information.

?HNAME
exception return
normal return

Input/Output

Input

AC0	byte pointer to buffer at least ?MXHN bytes long, or zero.
AC1	zero, or -1, or host ID.
AC2	ignored.

Output:

AC0	unchanged.
AC1	unchanged or Host ID returned.
AC2	unchanged or byte length of returned host name excluding the null terminator byte.

Exceptional Condition Codes in AC0

ERHNE	Host ones not exist.
ERIHN	Illegal host name.

FILE SYSTEM codes

Description

This system call is used to obtain host information on an AOS system running the Data General XODIAC networking product (see *XODIAC Network Management System User's Manual).* You can obtain local and remote host names and IDs with this call.

The value entered in AC1 determines the type of action this call will perform. The available actions are:

AC1 Contains	Action
0	Into AC1, return host ID for host name in buffer pointed to by byte pointer in AC0.
-1	Return local host information. Return the local host ID in AC1. Optionally, if AC0 is nonzero it must point to a buffer at least ?MXHN bytes long into which the local host name will be returned. In this case, upon return AC2 will contain the byte length (excluding the null terminator) of the host name.
Host ID	Return host name for host ID in AC1. AC0 must contain a byte pointer to a buffer at least ?MXHN bytes long into which the host name will be returned. Additionally, upon return AC2 will contain the byte length (excluding the null terminator) of the host name.

?IFPU

Initialize the floating point unit.

?IFPU
exception return
normal return

Input/Output

Input:

AC0 starting address of 18-word area for saving the FPU state.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This system call reserves an 18-word area for saving the state of the floating point unit (FPU). Each task using the FPU must first issue this call, or FPU results received by this task will be indeterminate. Even single task programs which use the FPU must issue this call, since the state of the FPU must be preserved for each process.

?INTWT

Define a console interrupt task.

?INTWT
exception return
normal return

Input/Output

Input:

none.

Output:

none.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

If console interrupts are disabled (by system call ?ODIS or by a previous CTRL-A sequence), then ?INTWT will enable them. The TCB address of the task issuing this call is placed in USTIT of the User Status Table. On a CTRL-A sequence the task issuing this call receives control at the normal return, and further interrupts are disabled until system call ?OEBL or another ?INTWT call is issued.

To use this call, one task must be dedicated to servicing console interrupts. You want to ensure that this task receives control as soon as possible, so it should be initiated at the highest priority of all tasks in the program.

?ITIME

Return a 48-bit timestamp.

?ITIME
exception return
normal return

Input/Output

Input:

none

Output:

AC0	low order portion of 48-bit timestamp.
AC1	middle order portion of 48-bit timestamp.
AC2	high order portion of 48-bit timestamp.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

You can use this system call to obtain 48-bit values suitable for timestamping events. This call uses the real-time clock to obtain its values so the granularity of the returned timestamp is equal to the real time clock frequency.

Note that this call is not guaranteed to return unique values each time it is made; it merely guarantees that each 48-bit value will not be less than any previously obtained 48-bit value. The guarantee is good only if the system is booted with the correct date and time every time it is booted.

?LOGEV

Log an event in the log file.

?LOGEV
exception return
normal return

Input/Output

Input:

AC0 event code.

AC1 length in bytes of message, or zero if no message.

AC2 byte pointer to message (ignored if AC1=0).

Output:

AC0 unchanged.

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERPRP Illegal parameter specified (event code in AC0 is not in required range).

ERPRV Caller is not privileged to make this call.

Description

This system call allows you to enter an arbitrary message into the system accounting file, SYSLOG. You are only permitted to make this call in SUPERUSER mode. This file is further described in the *AOS Operator's Guide* and the *Command Line Interpreter (CLI)(AOS and AOS/VS) User's Manual.* See also the ?SYLOG system call.

When you issue this call, the system writes data into SYSLOG in the format shown in Figure 9-3. Note that the record length, date, time, and event code are stored in an eight-word header, while the message that follows may be any size. The system will pad this message to make it a multiple of eight words long.

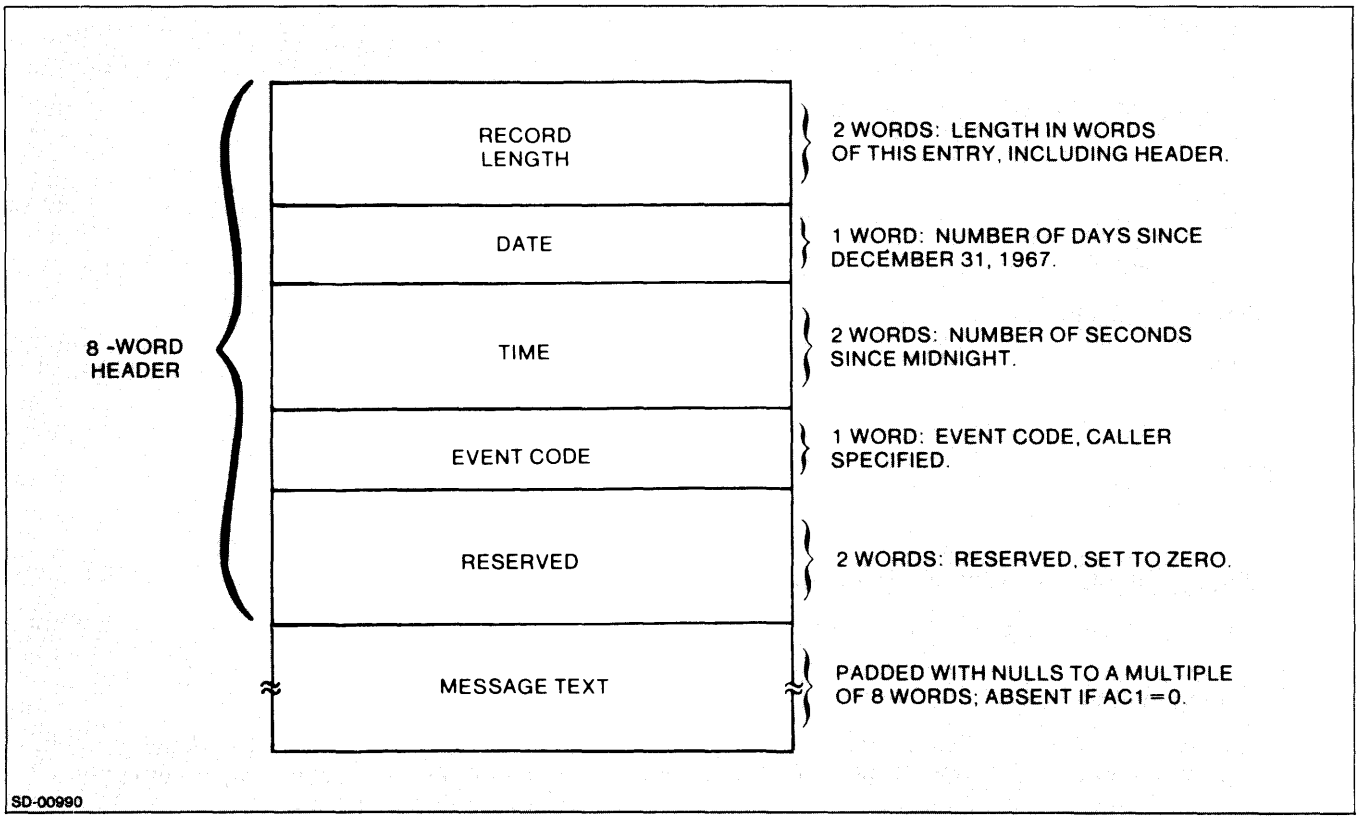


Figure 9-3. ?LOGEV Event Logging Format

The event code allows the caller to specify a unique format for each type of event logged into the file. When the operating system issues this call, it specifies an event code between ?LSMI and ?LDMA (currently assigned the values 0 and 2047 decimal, respectively), and adjusts the format of its messages in a manner unique to each code. When you make this call, you may use event codes ?LUMI through ?LMAX, inclusive.

These are currently defined to cover the range 2048 through 4096 decimal, but check a current PARU.LS listing or verify the value of ?LMAX minus ?LUMI in a program to insure range requirements. You can assign these codes any meaning you wish.

Note the following restrictions on this call: if AC1 (message length) is zero, any message will be ignored and only the eight-word header will be written into the file. If the message length exceeds 760 bytes, only the first 760 bytes will be entered. If logging is turned off, no message will be entered and the normal return will always be taken.

?ODIS

Disable console interrupts.

?ODIS
exception return
normal return

Input/Output

Input:

none.

Output:

none.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This system call prevents further CTRL-A sequences (CTRL-C followed by CTRL-A) from occurring until either system call ?OEBL or ?INTWT is issued, or a new program is started by system call ?CHAIN. When a new program is started, console interrupts are enabled by default.

?OEBL

Enable console interrupts.

?OEBL
exception return
normal return

Input/Output

Input:

none.

Output:

none.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

This system call enables CTRL-A sequences (CTRL-C followed by CTRL-A) to interrupt a program's normal operation. Unlike system call ?INTWT, this call does not supply an interrupt-processing capability.

Console interrupts are enabled when a program is first initiated.

?RNAME

Determine network location of a file.

?RNAME
exception return
normal return

Input/Output

Input:

AC0 byte pointer to file pathname.

Output:

AC0 zero if pathname points to a file on the local host, or host ID of the host on which the file resides.

Exceptional Condition Codes in AC0

File System codes.

Description

This call is most useful if your system is running the Data General XODIAC networking product. ?RNAME returns the host ID of the host on which the given file resides if the file exists. This call does not verify that the file exists. A returned host ID of zero means the file is a local file.

?SBIAS

Set the bias range.

?SBIAS
exception return
normal return

Input/Output

Input:

AC0 left byte: maximum bias factor.

right byte: maximum bias factor.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

ERPRV Caller is not the operator process.

ERBIF Illegal bias factor.

Description

This call sets the system bias factors. Only the operator CLI (PID=2) may issue this call. Both the minimum and maximum bias factors must be in the range 0-32. A maximum bias factor of 0 effectively means no maximum is enforced.

The minimum bias factor is the number of noninteractive swappable jobs that the system tries to keep in memory at all times. This partially overrides the normal system bias towards interactive jobs.

The maximum bias factor is the maximum number of noninteractive swappable jobs that the system will normally allow in memory at any one time. This is not an absolute value because even though the high bias factor is satisfied, a noninteractive process can be run if it can fit into memory without preempting any other process. This is to aid response time to interactive processes in a heavily loaded environment. It will help prevent the swapping in of large noninteractive jobs when a number of interactive jobs become blocked for an instant in time.

For more information about the use of bias factors, see the *AOS System Manager's Guide.*

?SDAY

Set the system calendar.

?SDAY
exception return
normal return

Input/Output

Input:

AC0 day of the month (1 to 31).

AC1 month of the year (1 to 12).

AC2 current year (less 1900).

Output:

AC0 unchanged.

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERTIM Attempt to set a day, month, or year which is outside the valid range. The range of acceptable years is 68-157 (in AC2).

ERPRV Attempt by a process other than the operator or one of its brothers to issue this call.

Description

This command sets the system calendar to a specific date. The date must be between 1 January 1968 and 31 December 2057, inclusive. The system will increment the date when the time of day passes 23 hours, 59 minutes, and 59 seconds. Only the operator process, or one of its brothers, may issue this call.

?SINFO

Get selected information on current AOS.

?SINFO
exception return
normal return

Input/Output

Input:

AC2 word pointer to eight-word parameter packet.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERMPR System call parameter address error.

Description

This system call returns selected information on the currently running AOS revision.

The specified parameter packet (see Figure 9-4) has offset ?SIRN returned with the current revision number; the left byte is the major number and the right byte is the minor number. For example, Rev. 02.03 would yield 001003 octal. Offset ?SIMM is returned with the maximum memory, by giving you the page number of the highest physical page. For example, a 512K-byte system would return 377 octal, (255 decimal) since page zero is the first page. You fill in offsets ?SILN and ?SIID with byte pointers to two 32-byte buffers, or default either value to zero. The buffer pointed to by ?SILN will be returned with the name of the master logical disk currently being used, while the buffer pointed to by ?SIID is returned with the system identifier of the currently running system.

The remaining four words of the parameter packet are currently reserved and must be set to zero.

If the word pointer in AC2 or either of the byte pointers in ?SILN and ?SIID point to locations outside the calling process's address space, an error is returned.

?SINFO (continued)

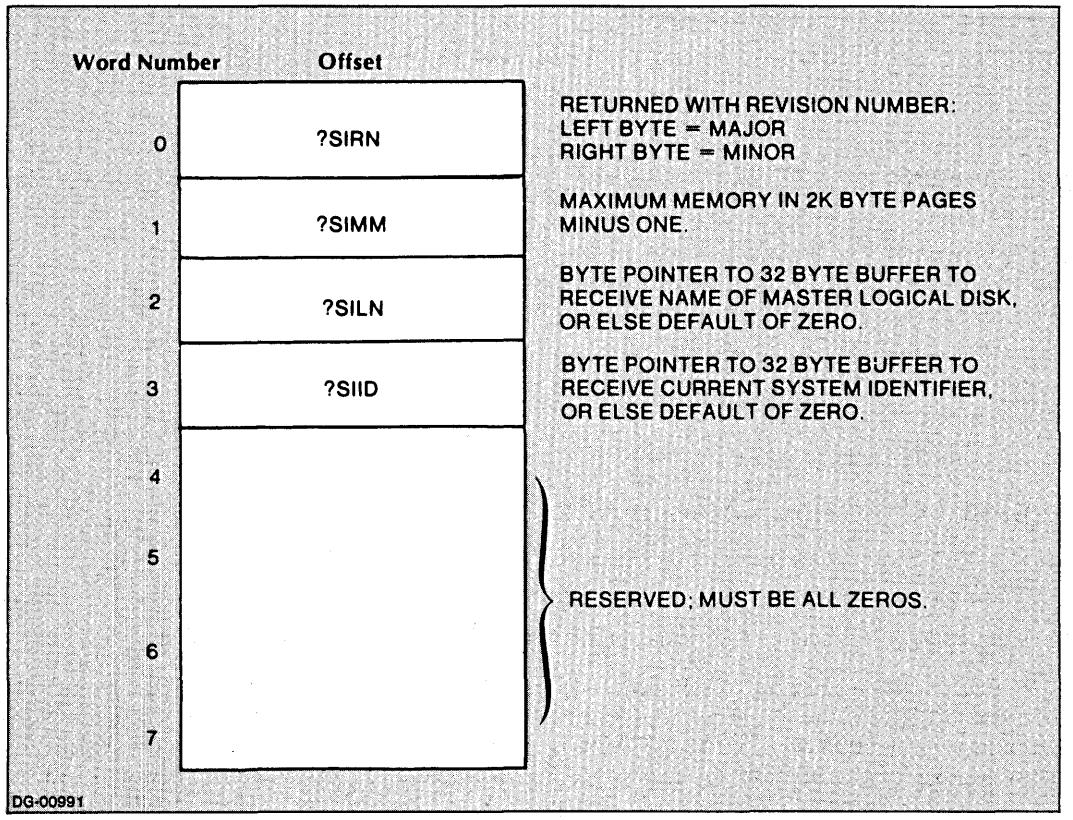


Figure 9-4. ?SINFO Parameter Packet

?SSID

Set the system identifier.

?SSID
exception return
normal return

Input/Output

Input:

AC2 byte pointer to the 32-byte buffer containing the ASCII string to be used as the system identifier.

Output:

AC2 unchanged.

Exceptional Condition Codes in AC0

ERPRV Caller not privileged to make this call.

ERMPR System call parameter address error.

ERPRE Invalid system call parameter (string is not 32 characters or less, terminating with a null).

Description

This call allows you to associate a unique system identifier with a running AOS system. The system identifier may be any string with up to 32 ASCII characters, including a terminating null. By using the ?GSID call, your program can access this character string to determine the system identifier. Only the operator's process (PID2) may issue the ?SSID system call.

?STOD

Set the system clock.

?STOD
exception return
normal return

Input/Output

Input:

AC0	seconds (0-59).
AC1	minutes (0-59).
AC2	hours (0-23).

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERTIM	Attempt to set a second, minute, or hour which is outside the valid range.
ERPRV	Attempt by a process other than the operator process or one of its brothers to issue this call.

Description

This call sets the system clock to a specific hour, minute and second. Only the operator process or one of its brothers may issue this call.

?SYLOG

Manipulate the system log file.

?SYLOG
exception return
normal return

Input/Output

Input:

AC0 log state and soft error state.

-1 = leave in current state.

+1 = start logging events in SYSLOG.

0 = stop logging events in SYSLOG.

+2 = suppress soft error messages to op console.

+3 = enable soft error messages to op console.

AC1 byte pointer to name to be given to current log file or 0 if current log file will not be renamed.

Output:

AC0 current log state. 0 if event logging is off; 1 if event logging is on.

AC1 unchanged.

Exceptional Conditional codes in AC0

ERPRE Illegal parameter specified (invalid value in AC0).

ERPRV Caller is not privileged to make this call.

Description

This system call allows the system operator to create, rename, and change the logging status of the system accounting file or log file. This file always has the pathname :SYSLOG. This system call may be issued only from the operator's process (PID2).

A ?SYLOG call that starts the logging of events will create the system log file if it does not already exist. Once logging is started, any messages sent to the log file, via ?LOGEV, as well as messages sent by the system itself and the EXEC (if it is running) will be written into :SYSLOG. See the ?LOGEV call for a description of the format of these messages. The Access Control List on :SYSLOG is set to OP, OWARE; the file type is ?FLOG. Turning logging off closes :SYSLOG and no further messages will be entered.

If you provide a byte pointer to a new filename in AC1, :SYSLOG (if open) will be closed and assigned the new name. When a new ?SYLOG call is issued to start logging events again, or if the present call specified that the file should remain open, a new :SYSLOG file will be created and henceforth all additional events logged will go into this new file. This means that all events are always logged into the file currently named :SYSLOG, and all past system logging files, once renamed, become historical. It is standard practice at Data General, and we recommend to all users, that you rename the system log file each morning with the previous day's date. You can also rename :SYSLOG with the ?RENAME system call, but we do not recommend it.

If an error occurs in processing the ?SYLOG call, the state and name of the system log file are left unchanged.

Examples

This section illustrates two system calls, ?DELAY and ?GTMES. ?DELAY suspends program control for a specified number of milliseconds, and ?GTMES reads a CLI command. (Note that we have illustrated the ?GTMES call throughout earlier examples).

Figure 9-5 illustrates the routine named DELAY.

This re-entrant routine fetches the value of AC1 input to the routine, and multiples this by 1000 decimal to produce an input argument to the ?DELAY call (this call delays control for a number of milliseconds). After the delay, control returns to the caller.

Figure 9-6 illustrates the program entitled TIMEOUT.

You invoke TIMEOUT by typing the command line

```
XEQ TIMEOUT seconds↵
```

where seconds is a number of seconds from 1 through 9. If you type something other than a digit (e.g., l or o), the CLM instruction sends control to OUT. This terminates the program and issues an appropriate error message.

?GTMES picks up the seconds argument. The ?GTMES packet specifies that the argument is to be copied into the area labeled SECS; the fourth word of the packet (?GRES) contains a byte pointer to this area. The second word in the packet specifies argument number 1 (0 would be the program name, TIMEOUT; we don't want that). And the third word contains 0, since we aren't interested in switch information. Note that we could have simply used the value returned in AC1, since in this case the argument should always be a decimal digit. We used the SECS area merely to illustrate this alternate technique.

After picking up the seconds argument, changing the ASCII code to octal, DELAY is called. Upon return from DELAY, we issue a good return and terminate the process. If you try this program yourself, notice that the time elapsing between typing the TIMEOUT command line and its termination will exceed the number of seconds you specify. This extra time is spent by the system as it creates the process in which to run TIMEOUT. This period of time depends on the amount of system activity which is currently happening, the number of concurrent users, etc.

```
 0001 DELAY AOS ASSEMBLER
                          .TITLE  DELAY
02                        .ENT    DELAY
03                        .EXTN   ERROR
04
05                ;       THIS ROUTINE DELAYS PROGRAM
06                ;       EXECUTION BY THE NUMBER OF
07                ;       SECONDS INPUT IN AC1.
08
09                        .NREL
10
11 00000'163710 DELAY:  SAVE    0
12        000000
13
14
15 00002'102400          SUB     0,0
16 00003'172070          ELEF    2,1000.
17        001750
18
19
20 00005'147710          MULS            ;DELAY FOR
21                       ?DELAY          ;(AC1) * 1000. MSEC
22 00010'002402          JMP     @.ERROR
23 00011'127710          RTN
24
25 00012'000000$.ERROR: ERROR
26
27                        .END
**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS  0002 DELAY
DELAY 000000' EN     1/02    1/11
ERROR 000000  XN     1/03    1/25
.ERRO 000012'        1/22    1/25
?DELA 000544  MC     1/21
?XCAL 000001         1/22
```

Figure 9-5. DELAY

```
 0001  TIMEO    AOS ASSEMBLER REV 01.09          17:19:46 12/05/78
01
                                .TITL   TIMEOUT
03                              .EXTN   DELAY,ERROR
04
05              ;THIS PROGRAM DELAYS TERMINATING ITSELF FROM 1 TO
06              ;9 SECONDS. TO INVOKE THE PROGRAM, TYPE "XEQ
07              ;TIMEOUT <SECS>", WHERE <SECS> IS A DECIMAL NUMBER
08              ;FROM 1 THROUGH 9.
09
10                              .NREL
11
12                      START:  ?GTMES  MSGPK   ;GET THE # OF SECS.TO WAIT
13 00004'002417                 JMP     @.ERROR
14 00005'020422                 LDA     0,MSGPK+?GRES   ;GET BYTE PNTR.
15 00006'106710                 LDB     0,1             ;GET TIMEOUT SECS.
16 00007'126370                 CLM     1,1
17 00010'000060                 "0
18 00011'000071                 "9
19 00012'000417                 JMP     OUT             ;ILLEGAL DELAY CHAR.
20 00013'147770                 ANDI    017,1   ;CHANGE ASCII TO OCTAL AND
21        000017
22
23 00015'006405                 JSR     @.DLA   ;CALL "DELAY"
24 00016'152620                 SUBZR   2,2     ;SET GOOD RTN FLAG
25                              ?RETURN
26 00021'002402                 JMP     @.ERROR
27
28 00022'000000$        .DLA:   DELAY
29 00023'000000$        .ERROR: ERROR
30                ;?GTMES PACKET
31 00024'000004         MSGPK:  .BLK    ?GTLN   ;LENGTH OF ?GTMES PKT
32        000024'               .LOC    MSGPK+?GREQ     ;MSG.REQUEST TYPE
33 00024'000003                 ?GARG
34        000025'               .LOC    MSGPK+?GNUM     ;ARGUMENT NUMBER
35 00025'000001                 1                       ;GET # OF SECS.
36        000026'               .LOC    MSGPK+?GSW      ;BYTE PNTR TO
37 00026'000000                 0                       ;KEYWORD SWITCH? NO.
38        000027'               .LOC    MSGPK+?GRES     ;BYTE PNTR. TO SECS
39 00027'000060"                SECS*2                  ;BUFFER
40        000030'               .LOC    MSGPK+?GTLN     ;END OF ?GTMES PKT.
41 00030'000001         SECS:   .BLK    1
42 00031'024406         OUT:    LDA     1,MSG
43 00032'030404                 LDA     2,FLAGS
44                              ?RETURN
45 00035'002766                 JMP     @.ERROR
46 00036'100040         FLAGS:  ?RFCF+32.       ;MSG.IN CLI.FORMAT,32.CHARS
47 00037'000100"        MSG:    .+1*2
48 00040'054517                 .TXT    "YOU SPECIFIED AN ILLEGAL DELAY."
49        052440
50        051520
51        042503
52        044506
53        044505
54        042040
55        040516
56        020111
57        046114
58        042507
59        040514
60        020104
```

Figure 9-6. TIMEOUT (continues)

```
 0002  TIMEO
01        042514
02        040531
03        027000
04
05
06
07
08
09
10
11
12
13
14                                .END    START

**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS



  0003 TIMEO

DELAY 000001  XN     1/03     1/28
ERROR 000002  XN     1/03     1/29
FLAGS 000036'        1/43     1/46
MSG   000037'        1/42     1/47
MSGPK 000024'        1/13     1/14     1/31     1/32     1/34     1/36     1/38
                     1/40
OUT   000031'        1/19     1/42
SECS  000030'        1/39     1/41
START 000000'        1/12     2/14
.DLA  000022'        1/23     1/28
.ERRO 000023'        1/13     1/26     1/29     1/45
?GTME 004545  MC     1/12
?RETU 004600  MC     1/25     1/44
?XCAL 000001         1/13     1/26     1/45
```

Figure 9-6. TIMEOUT (concluded)

End of Chapter

Chapter 10
Binary Synchronous Communications

AOS supports binary synchronous communications (BSC) over dedicated or switched communications lines. This chapter describes the system calls you need to implement BSC communications. We stress that this chapter is not a tutorial; we assume that you are familiar with BSC protocol, the rules governing binary synchronous communications, and that you will use the information here mainly as a reference to the BSC system calls. However, we will begin by defining some of the terms used throughout this chapter.

General Definitions

A *dedicated* communications line is one which continuously connects two or more stations, regardless of the amount of time the line is actually in use. The word "dedicated" is the key term; a line of this kind is dedicated to serving specific local and remote stations.

In contrast, a *switched* line is one on which you establish a connection between the local and remote stations by dialing procedures. *

AOS recognizes each binary synchronous line by the device name @SLNx, where x represents the line number. When you enable a BSC line (with the system call ?SEBL), you must supply the @SLN designator with the correct line number. However, you need not specify whether the enabled line is dedicated or switched.

The system assigns a channel number to each enabled line and returns this value to the ?SEBL parameter packet. Unlike disk files, you cannot open a BSC line on more than one channel.

The origin (sender) or destination (receiver) of data over a BSC line is a *station.* A station issues the ?SSND system call to send data and the ?SRCV call to receive data over an enabled BSC line. BSC protocol distinguishes between *send initial* and *send continue* calls, and *receive initial* and *receive continue* calls. A call is considered initial if it is the call that opens the communications session.

The ?SSND and ?SRCV system calls depend both upon timing and upon the interaction between the sending and receiving stations. For this reason, the system enters error recovery procedures when it encounters certain timing errors or inappropriate responses to the ?SSND and ?SRCV calls. Since you should view the error recovery procedures in the context of the send and receive calls, we document them in a separate section, entitled "BSC Error Recovery Procedures", at the end of this chapter.

Line Configurations

There are two types of BSC line configurations: point-to-point and multipoint. The difference between the two types is based on the relationship between the stations.

On a *point-to-point* line, each station must bid for the line; i.e., request its use. When both stations bid at the same time, the line is under *contention.* Strictly speaking, contention occurs when a point-to-point station sends a bid for the line, and receives a bid from the other station in response.

Under the AOS BSC implementation, only two stations can be on a point-to-point line. When you enable a point-to-point line, you must designate your computer as either the *primary* station or the *secondary* station. As these terms suggest, AOS favors the primary station over the secondary when contention occurs. The operating system resolves contention on a point-to-point line as follows:

- If your station is the primary, the system automatically follows your bid with another bid sequence. The secondary station should acknowledge this additional bid sequence.
- If your station is the secondary, the system gives you an exception return. You should then issue an ?SRCV receive initial call to receive the primary station's bid sequence.

On a *multipoint* line (also known as a *multidrop* line), there is no contention between stations. You must designate your computer as either the *control station* or a *tributary* of the control station.

The tributary stations on a multipoint line are completely subservient to the control station. A tributary can send data over the line only at the request of the control station and only to the control station. A tributary can receive data over the line only from the control station and only if the control station selects it. Tributary stations cannot communicate directly with one another.

A multipoint line usually connects one local station with more than one remote station. It can connect as few as two stations, but this is the exception rather than the rule. The important distinction between point-to-point and multipoint lines is that on a multipoint line, one station -- the control station -- has complete control over the activity on the line.

Note that when you enable a BSC line with ?SEBL, you must declare it to be either a point-to-point line or a multipoint line. You must also declare your station's status -- primary or secondary, control station or tributary -- with the ?SEBL call.

Figure 10-1 contrasts a point-to-point line configuration with a multipoint configuration.

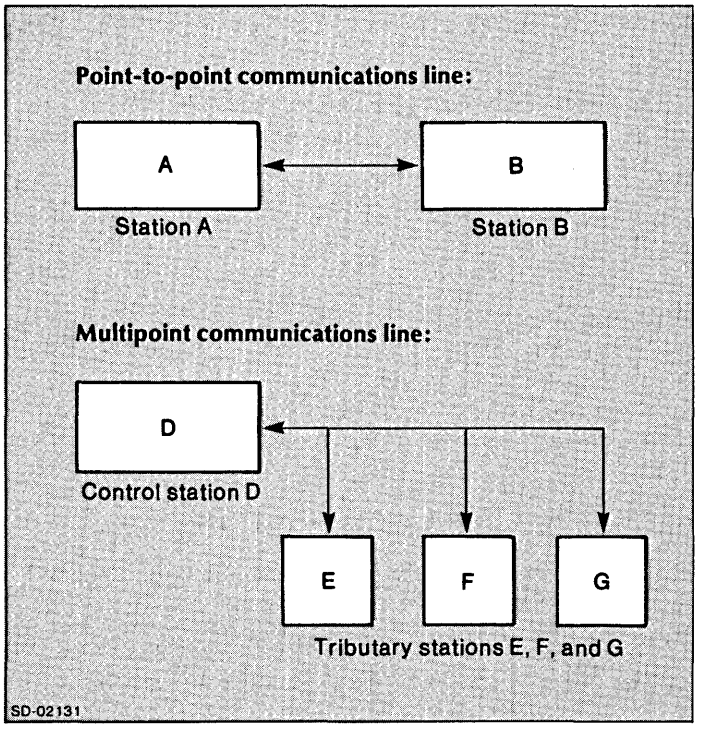


Figure 10-1. Point-to-point/Multipoint Line Configurations

Multipoint Lines: Polling and Selecting

The control station manages the activity on a multipoint line by two operations: polling and selecting.

The control station *polls* by contacting its tributaries to see if any of them has data to send to it. There are two kinds of polling operations. In a *general poll,* the control station contacts each of its tributaries, in round-robin fashion, and accepts the first positive response. In a *specific poll,* the control station contacts a single tributary to solicit data.

The control station *selects* by contacting a specific tributary to see if it is ready to receive data from the control station.

In order for polling and selecting to occur, each tributary on a multipoint line must have two unique identifiers: a *poll address,* and a *select address.* If your computer is a tributary, you must define its poll address and select address by issuing the ?SDPOL system call.

If your computer is a control station, you must issue ?SDPOL before polling or selecting in order to define a *polling list.* A polling list is a series of contiguous words that contains each tributary's poll address and *device address.* The device address points to the peripheral device from which the control station will request data when it polls that tributary.

To perform polling, a control station issues the system call ?SRCV ("receive data or control characters") specifying whether the call is a receive initial or a receive continue and whether the operation is general polling or specific polling.

In its first general poll, the control station starts with the poll and device address entry at the top of the polling list (the lowest relative terminal number), and sends this entry down the BSC line. Each tributary recognizes its own poll address; it responds to the poll only if the entry matches its poll address. If the poll address sent by the control station does not match a tributary's poll address, the tributary ignores it.

A general poll ends when a tributary responds to its poll address by sending data to the control station. If there is no response to a particular poll address entry, the control station continues to poll until it reaches the end of the polling list. At that point, AOS takes the exception return from the control station's ?SRCV call, and passes error code EREPL (end of polling list) to AC0.

As we mentioned, general polling is a round-robin operation. This means that when a general poll ends in a positive response, the next general poll begins with the next relative terminal on the polling list (i.e., the tributary immediately following the previous respondent). Specific polling, that is, polling one and only one tributary, is a way to break out of the round-robin method of general polling.

Enabling/Disabling a Relative Terminal

The operating system assigns a *relative terminal number* to each tributary station, based on its position on the polling list.

The first time you enable a multipoint line (?SEBL) and define its polling list (?SDPOL), the operating system enables all relative terminals on the list for polling. To disable a relative terminal without redefining the polling list, issue the ?SDRT call. To re-enable a relative terminal, issue ?SERT.

Disabling a relative terminal does not affect the corresponding tributary station; it simply means that the control station ignores that tributary when it performs general polling, until you subsequently re-enable the relative terminal or define a new polling list.

Binary Synchronous Protocol

The logic behind data transmissions over a BSC line is binary synchronous communications protocol. Briefly, binary synchronous protocol is a set of rules governing

- the initialization of communications over a BSC line
- the orderly exchange of data over a BSC line
- the termination of communications over a BSC line

These objectives are accomplished in part by the protocol's *data-link control characters,* synchronization characters mutually recognized by the sending and receiving stations. Data transmissions over a BSC line typically consist of text, header information (optional), and data-link control characters, which delimit various portions of the data block and control its transmission.

None of the BSC system calls require data-link control characters as input. The operating system provides the required control characters when you send text or header information over a BSC line, and removes them when you receive text or header information. However, since several of the system call descriptions refer to the data-link control characters, Table 10-1 defines the control characters mentioned in this chapter. Refer to the illustrations at the end of the chapter for illustrations of the system's BSC implementation.

Note that BSC protocol supports *transparent text mode,* under which most control characters are simply treated as bit patterns without control significance. The exceptions are DLE STX, which signals the beginning of transparent text mode, and DLE ETB or DLE ETX, which signal the end of transparent text mode. If you're sending data which may match the bit patterns of the control characters, you should send it transparently.

System Call Summary

The following list summarizes the binary synchronous communication system calls.

System Call	Function
?SDBL	Disable a BSC line
?SDPOL	Define a polling list or a poll address/select address pair
?SDRT	Disable a relative terminal
?SEBL	Enable a BSC line
?SERT	Re-enable a relative terminal
?SGES	Get BSC error statistics
?SRCV	Receive data or a control sequence over a BSC line
?SSND	Send data or a control sequence over a BSC line

Table 10-1. BSC Protocol Data-Link Control Characters

Data-Link Control Character	Description
ACK0	Affirmative Acknowledgement:
ACK1	Positive replies, sent in alternating sequence, to indicate that the receiver has accepted the previous block without error, and is ready to accept the next block of the transmission. ACK0 is also a positive response to a line bid (ENQ) for a point-to-point line and to a selection sequence on a multipoint line.
BCC	Block Check Character: A value generated by the transmitting station and sent with each data block to validate the block's contents. The receiving station generates its own BCC. If the two values match, the block is accepted as error-free. A BCC follows every ITB, ETB, and ETX character. There is a variety of schemes by which BCC may be calculated. Commonly, when the ASCII code set is used as the line code, the BCC accumulation is performed as a longitudinal redundancy check (LRC), and with the EBCDIC code set as the line code, as a cyclic redundancy check (CRC). (See the Input Status Word description for a more complete explanation of BCC types and their correct use.)
DLE	Data-link Escape: The first character in a two-character sequence used to signal the beginning or end of transparent text mode. The sequence DLE STX signals the beginning of transparent text mode. The sequence DLE ETB or DLE ETX signals the end of transparent text mode.
DLE EOT	Data-link Escape, End-of-Transmission: Signals a *line disconnect* for a switched line. The sending or receiving station usually transmit this sequence when all message exchanges are finished.
ENQ	Enquiry: Sent by a station on a point-to-point line to bid for the line (for transmission of data). Sent by the control station on a multipoint line to signal the end of a polling or selecting sequence. A transmitting station can also send ENQ to ask the receiver to repeat a response if the original response was garbled or not received when expected.
EOT	End-of-Transmission: Signals the end of a message transmission (consisting of one or more separately transmitted data blocks), and resets the receiving station. On a multipoint line, a polled station sends an EOT to indicate that it has nothing to send back to the control station. EOT can also serve as an abort signal to indicate a system or transmission malfunction.
ETB	End-of-Transmission Block
ETX	End-of-Text: Signal the end of a data block that began with an SOH or STX. Both the ETB and ETX characters reverse the direction of the transmission. When a station receives an ETB or ETX, it replies with a control character indicating its status (i.e., ACK0, ACK1, NAK, WACK, or RVI). An ETB terminates every text block except the last. An ETX implies an end-of-file condition; thus, it terminates the last block of text in a message.

(continues)

Table 10-1. BSC Protocol Data-Link Control Characters

Data-Link Control Character	Description
ITB	End-of-Intermediate-Transmission Block: Separates records within a block and/or delimits field boundaries for error checking. ITB *does not* reverse the direction of the transmission.
NAK	Negative Acknowledgement: Sent by the receiving station to indicate that it is not ready to receive, or to request the retransmission of an erroreous block.
RVI	Reverse Interrupt: A positive response used instead of ACK0 or ACK1; signals that the receiver must interrupt the transmission to send the transmitting station a high priority message. The transmitting station treats an RVI as a positive acknowledgement and, in response, transmits all the data that prevents it from becoming a receiving station. The transmitting station may perform more than one block transmission to empty all its buffers. On a multipoint line, a control station may send an RVI after receiving data from a tributary, to indicate that it wants to communicate with a different tributary.
SOH	Start-of-Header: Signals the start of header information (ancillary information within a block).
STX	Start-of-Text: Signals the beginning of the text (and terminates the header information).
SYN	Synchronization Character: Establishes and maintains character synchronization; also serves as filler in the absence of data or control characters. Each transmission must begin with at least two contiguous SYN characters.
TTD	Temporary Text Delay: A two-character sequence consisting of STX ENQ sent by the transmitting station to retain the line without immediately sending data. The receiving station responds with a NAK. The TTD, NAK sequence may repeat, if the transmitter needs additional delays.
WACK	Wait-Before-Transmit Positive Acknowledgement: A positive acknowledgement sent by the receiver; signals that the receiver is temporarily unable to receive. (A receiver can send WACK as a response to a line bid on a point-to-point line or a selection sequence on a multipoint line, or as a response to data.) A receiving station may send more than one WACK until it is ready to receive. The transmitting station may respond with ENQ, EOT, or DLE EOT.

(concluded)

?SDBL

Disable a BSC line.

?SDBL
exception return
normal return

Input/Output

Input:

AC1 contains the channel number assigned to the line.

Output:

AC1 unchanged.

Exceptional Condition Codes in AC0

ERDSL Device associated with the channel number is not a synchronous line.

Description

?SDBL disables the BSC line associated with the channel number specified in AC1. When you issue this call, the system breaks the association between the line and its channel, and the association between the line and the process that enabled it. *

?SDPOL

Define a polling list or a poll address/select address pair.

?SDPOL
exception return
normal return

Input/Output

Input:

AC bits 0-7 contain one of the following values:

- the number of relative terminals (tributaries) in the polling list, if the caller is a control station.
- 1, if the caller is a tributary.

bits 8-15 contain one of the following values:

- the length (in bytes) of each poll address entry (including device address characters), if the caller is a control station.
- the length of the poll address, if the caller is a tributary (the select address is the same length).

AC1 contains the channel number assigned to the line.

AC2 contains a byte pointer to the polling list.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERIMM	Not enough memory for poll/select list.
ERLNA	I/O request for disabled line.
ERLNM	Line not multipoint.
ERNSL	Attempt to enable non-sync line.

Description

The ?SDPOL call serves two purposes: If the caller is a control station on a multipoint line, ?SDPOL defines a polling list. If the caller is a tributary on a multipoint line, ?SDPOL defines the caller's poll address and select address. Note that only stations on a multipoint BSC line may issue this call.

The operating system uses the addresses defined by ?SDPOL during receive initial (?SRCV) general polling sequences. If your computer is a control station, you must issue ?SDPOL before you can perform polling or selecting. If your computer is a tributary, you must issue ?SDPOL to define your poll and select addresses.

Defining a Polling List

If you're using ?SDPOL from a control station, construct the polling list in contiguous locations in your logical address space before issuing the call. The entries in the list can be of any byte length, provided each entry is the same length. Figure 10-2 shows a sample polling list, where each entry is four bytes long.

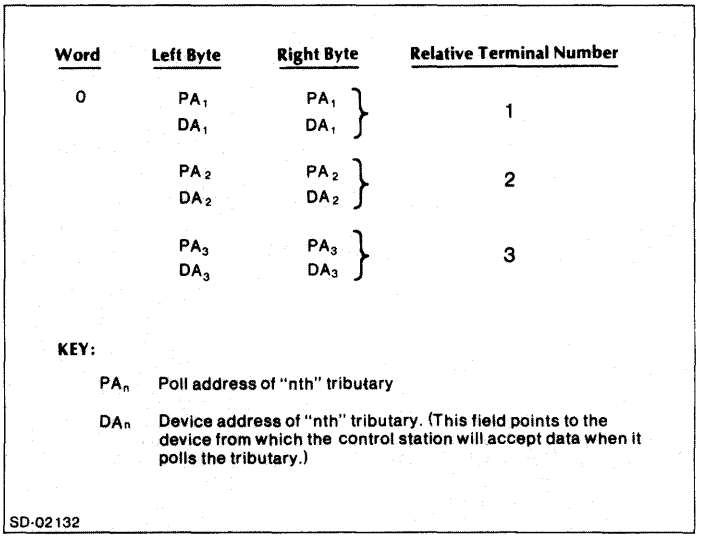


Figure 10-2. Polling List Defined by a Control Station

Before issuing the ?SDPOL call, load AC0 with the byte length of the entries (bits 16-23) and the number of entries (bits 24-31). The system uses these two values to determine the length of the polling list and to assign relative terminal numbers to the tributaries.

Defining a Poll Address and Select Address For a Tributary

The procedure for defining a single poll address and select address entry for a tributary is similar to the procedure for defining a polling list. First, define the poll and select addresses in contiguous locations in your logical address space. Before issuing the ?SDPOL call, load bits 8-15 of AC0 with the byte length of the poll or select address. (Each address must be the same byte length.) Load bits 0-7 of AC0 with 1. Figure 10-3 shows a sample poll address and select address entry, where each address is two bytes. *

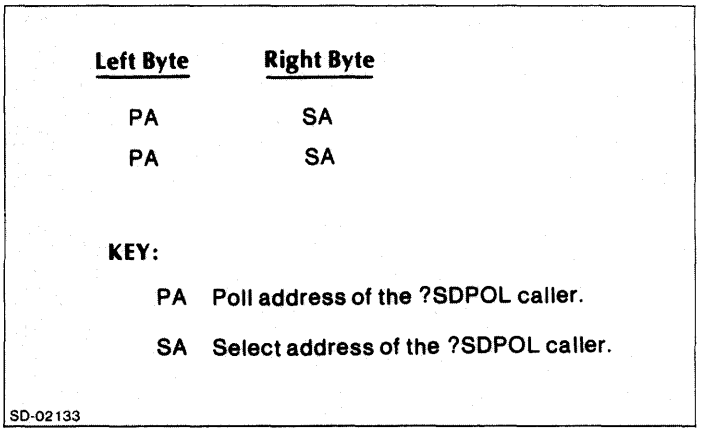


Figure 10-3. Poll and Select Addresses Defined by a Tributary

?SDRT/?SERT

Disable/re-enable a relative terminal.

?SDRT or ?SDRT
exception return
normal return

Input/Output

Input:

AC0	contains the relative terminal number to disable or re-enable.
AC1	contains the channel number assigned to the line.

Output:

AC0	unchanged.
AC1	unchanged.

Exceptional Condition Codes in AC0

ERIRT	Illegal relative terminal number.
ERLNA	I/O request for disabled line (?SERT).
ERLNM	Line not multipoint.
ERNPL	Polling list not defined.
ERNSL	Attempt to enable non-sync line (?SERT).

Description

The ?SDRT and ?SERT calls respectively disable and re-enable a relative terminal for subsequent polling or selecting. Only a control station on a multipoint line can issue these calls.

When a control station defines a polling list with ?SDPOL, the operating system automatically enables all the relative terminals on the list. In effect, ?SDRT "removes" a relative terminal from the list by directing the system to ignore that terminal during the next polling sequence. In general, you disable a tributuary if it repeatedly failed to respond to a poll or select sequence.

?SERT allows the control station to "undo" a previous ?SDRT call; that is, to re-enable a previously disabled relative terminal.

?SEBL

Enable a BSC line.

?SEBL
exception return
normal return

Input/Output

Input:

AC0 contains a byte pointer to the line's device name.

AC2 contains the address of the ?SEBL parameter packet.

Output:

AC0 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERDCU DCU inoperative (can't be initialized).

EREPE Line already enabled on ?SEBL call.

ERFCT Failure to connect.

ERINE Initialization parameter error. *

ERNSL Attempt to enable non-sync line.

Description

?SEBL enables a BSC line and associates it with the calling process. This call also directs the system to acquire resources for the line and to perform hardware-related initialization.

Before issuing this call, set up the ?SEBL parameter packet in your logical address space, load AC2 with the packet address, and load AC0 with a device name for the line. The device name for a BSC line is @SLNx, where @SLN is the BSC designator and x is the line number.

Figure 10-4 shows the structure of the ?SEBL packet, and Table 10-2 describes the contents of each offset.

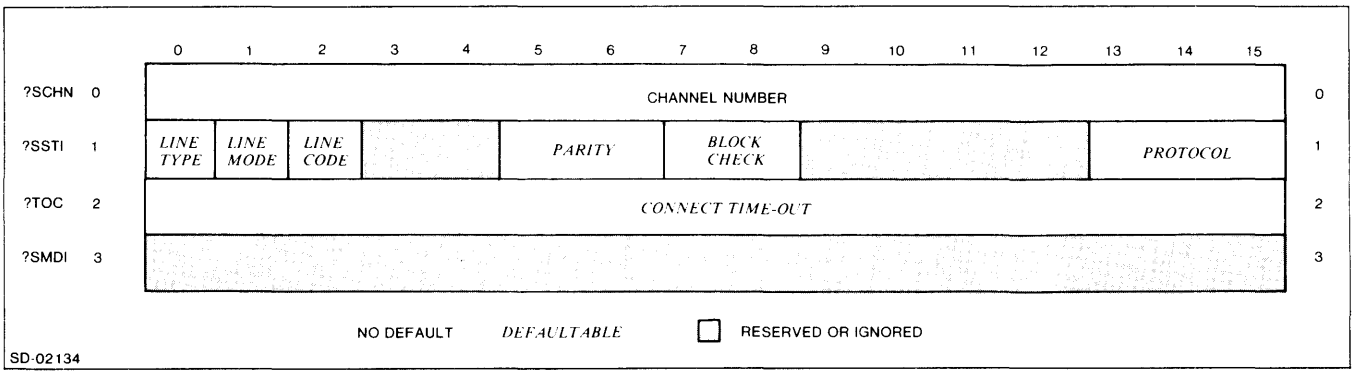


Figure 10-4. ?SEBL Parameter Packet: Structure

Table 10-2. ?SEBL Parameter Packet

Offset	Contents	Default
?SCHN	Channel number (returned by system)	N/A
?SSTI	Input status word: Line type: ?SDPP - point-to-point ?SDMD - multipoint Line mode: ?SDSC - secondary (point-to-point) tributary multipoint ?SDPR - primary (point-to-point) control station (multipoint) Line code: ?SEBC - EBCDIC ?SASC ASCII Block Check: ?SCRC - CRC16 ?SCIT - CCITT16 ?SLRC - LRC Parity: ?SNPR - no parity ?SOPR - odd parity ?SEPR - even parity Protocol: ?SBSC (normal BSC protocol)	?SDPP ?SDSC ?SEB ?SCRC ?SNPR ?SBSC
?STOC	Connect time-out: -1 30 seconds 0 no time-out 16383 maximum time-out	-1 (30 seconds)
?SMDI	Reserved (must be set to 0)	N/A

The Input Status Word

Offset ?SSTI contains specifications for the configuration, status, character set, parity, and protocol of the line you're enabling.

If you're enabling a point-to-point line (you selected ?SDPP as the line type), set the *line mode* parameter to ?SDPR if your computer is the primary station and to ?SDSC if your computer is the secondary station. If you're enabling a multipoint line, set the line mode to ?SDPR if your computer is the control station and to ?SDSC if it is a tributary.

The *line code* specification tells the operating system which character set, EBCDIC or ASCII, to use for the data-link control characters. The default character set is EBCDIC. Note that the system does not translate text or header information into the character set you specify here. You must translate the outgoing data, if necessary. AOS returns incoming data to you in the character set in which it was received.

The *block check* field lets you select the block check scheme you want to use for the transmission. *Block checking* is the system's method of validating the entire text block, according to the BSC protocol. Refer to Table 10-1 for a description of the block check data-link character. The sending and receiving stations on a BSC line must use the same block check method, otherwise, the system cannot verify the contents of the transmission blocks.

If your line code is EBCDIC (that is, if you selected either ?SEBC or the default line code specification), choose either ?SCRC or ?SCIT as the block check type. Both masks correspond to cyclic redundancy checking (see Table 10-1); ?SCRC corresponds to the CRC-16 method and ?SCIT to the CCITT method. It is invalid to specify the BCC type as ?SLRC (longitudinal redundancy checking) for EBCDIC transmissions.

If your line code is ASCII (that is, you selected ?SASC as the line code), you may choose ?SLRC, ?SCRC, or ?SCIT as the block check type. Furthermore, should you specify ?SLRC for an ASCII line, you may opt for parity checking as well. Parity checking is valid only for an ASCII line, and ought to be specified as odd parity (?SOPR), even parity (?SEPR), or no parity (?SNPR) if you do not wish parity checking to be performed at all.

Connect Time-out

The *connect time-out* specification, offset ?STOC, defines how many seconds the system will wait for the line to connect before taking the ?SEBL exception return. When you enable a BSC line, the system first examines the state of the data set flag, DSR, which the hardware raises as soon as the line is connected. If the DSR flag is not up, the system suspends the ?SEBL calling task until DSR is raised. If the flag is not raised within the time-out interval, the system takes the ?SEBL exception return and passes error code ERFCT to AC0.

?SGES

Get BSC error statistics.

?SGES
exception return
normal return

Input/Output

Input:

AC0 contains -1, for a cumulative error tally; otherwise, ignored.

AC1 contains the channel number assigned to the line.

AC2 contains the address of the ?SGES parameter packet.

Output:

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERLNA I/O request for disabled line.

ERNSL Attempt to enable non-sync line.

Description

?SGES returns the error statistics and the system records for each enabled BSC line. These statistics include the number of block check errors, the number of time-outs, and the total number of NAK (negative acknowledgement) characters received in response to send operations. If you set AC0 to -1 on input, the system returns a *cumulative error tally,* i.e., a record of all errors since the line was enabled, or since the last ?SGES in which you did not select the cumulative error option.

Before issuing ?SGES, set up a parameter packet of ?SGLN words, and load the packet address into AC2. Figure 10-5 shows the structure of the ?SGES packet. Table 10-3 describes each offset.

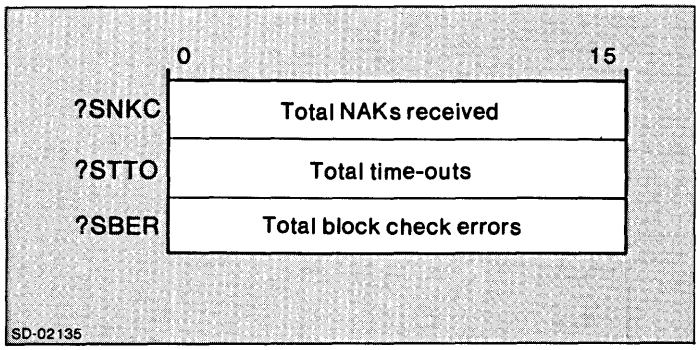


Figure 10-5. ?SGES Parameter Packet: Structure

Table 10-3. ?SGES Parameter Packet

Offset	Contents
?SNKC	Total number of NAKs received in response to a send operation (see ?SSND). A large number of NAKs may indicate that the data was not received or, if it was received, that the system found a block check error.
?SSTO	Total number of time-outs during receive operations. If there were no time-outs, the system returns 0 to this offset. (See the description of ?SRCV.)
?SBER	Total number of block check errors during receive operations. These are not errors that lead to exception returns from ?SRCV calls; rather, they are errors in the data. (When an ?SRCV error occurs, the system enters an error recovery procedure in which it tries again to receive the data, repeatedly, if necessary. The system may receive the data correctly on a retry. For more information about receive operations, see the description of ?SRCV. For information about the error recovery procedures, see "BSC Error Recovery Procedures" at the end of this chapter.)

?SGES can give you a general idea of the line's quality, by telling you the number of retries necessary for the previous transmission and the number of block check errors and NAK characters.

?SRCV

Receive data or a control sequence over a BSC line.

?SRCV
exception return
normal return

Input/Output

Input:

AC0 used by multipoint control stations only and contains one of the following values:

- 1, for a general poll.
- byte pointer to a poll address, for a specific poll.

AC1 contains the channel number assigned to the line.

AC2 contains the address of the ?SRCV parameter packet.

Output:

AC0 unchanged.

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERBPE User buffer byte pointer error.

ERCRC CRC check (Indicates a cyclical redundancy block check error).

ERDIS Disconnect occurred on a switched line.

ERENQ ENQ received after time-out (The system detected an ENQ from the sending station after reaching the limit for retries).

EREOT EOT character received.

EREPL End of polling list reached.

ERISE Input status error.

ERLIS Line in session (You tried to issue two receive initial calls in a row).

ERLNA Attempt to enable non-sync line.

ERNAK Transmission failure (the system received a NAK while waiting for a line bid, and the system has reached its limit for retries (point-to-point lines only)).

ERNPL Polling list not defined (this error occurs only on 1mod page 4 lines).

ERNSL Device associated with the channel number is not a synchronous line.

ERSCS You tried to issue a receive continue without a prior receive initial.

ERSIM There is an outstanding system call on this line.

ERTOF	Time-out value reached.
ERTRF	Transmitter failure.
ERUNI	Unknown or inappropriate response received.
ERVBP	Invalid byte pointer passed as a system call argument.

Description

?SRCV prepares the calling station to receive a block of data or data-link control characters over an enabled BSC line. Before issuing this call,

- reserve a receive buffer in your logical address space for the incoming data or control sequence
- set up the ?SRCV parameter packet in your logical address space
- load AC1 with the BSC line's channel number
- load AC2 with the packet address

Figure 10-6 shows the structure of the ?SRCV packet, and Table 10-4 describes each offset.

There are two types of ?SRCV calls: *receive initial* calls (calls that open the transmission) and *receive continue* calls. The system executes these calls differently, depending on whether the caller is a station on a point-to-point line, or the control station or a tributary on a multipoint line. We've divided the discussion of ?SRCV to reflect these differences.

Note that you must set certain parameters regardless of the kind of station you're calling from. The next section describes these parameters.

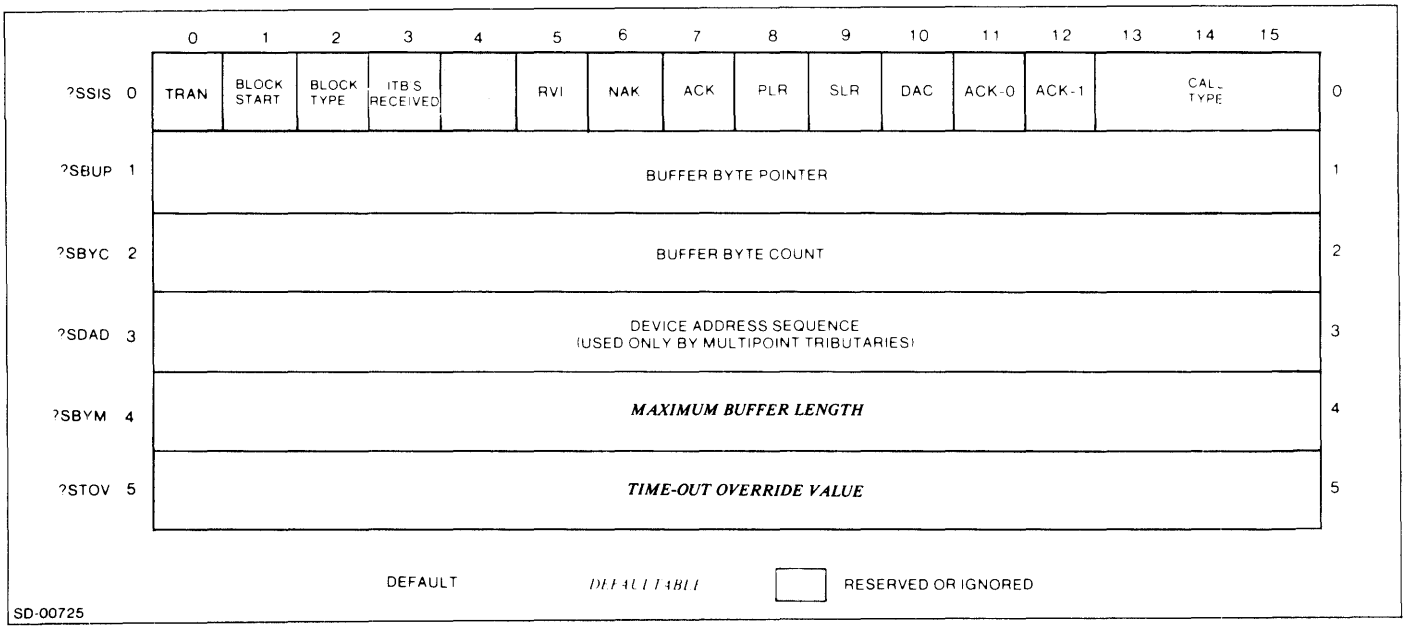


Figure 10-6. ?SRCV Parameter Packet: Structure

Table 10-4. ?SRCV Parameter Packet

Offset	Contents	Input Value	Output Value	Default
?SSIS	Input status word:			
	Text mode	None	?TRAN - If received transparent data	N/A
	Block start	None	?STXB - STX received ?SOHB - SOH received	N/A
	Intermediate text block indicator	None	?SITB	N/A
	Multipoint information	None	?SPLR - Your system was polled	N/A
			?SSLR - Your system was selected	N/A
	Respond type: (Receive continue only)			
	(Directs the system to respond to the sender with the specified control character.)	?SRVI - RVI ?SNAK - NAK ?SACK - ACK0 or ACK1 ?SAK0 - Override ACK0, ACK1 ?SAK1 - Alternation SEND specified ACK sequence	Unchanged	None
	Call type	?CINT - Receive initial ?CONT - Receive continue The following apply *only* to receive initials made by multipoint tributaries: ?SWAK - Send WACK, then wait for poll or select ?SPET - Send EOT, then wait for poll or select ?SPRV - Send RVI, then wait poll or select	Unchanged	None

(continues)

Table 10-4. ?SRCV Parameter Packet

Offset	Contents	Input Value	Output Value	Default
?SSIS (cont.)		?SPNK - Send NAK, then wait for poll or select		
	Component selection	None	?SDAC - Received device address characters	N/A
?SRES	Reserved	Must be 0	N/A	N/A
?SBUP	Buffer byte pointer	Byte pointer to a receive buffer for the data	Unchanged	N/A
?SBYC	Buffer byte count	None	Number of bytes received	N/A
?SDAD	Device address sequence	None	Left-justified device address, two characters maximum (You receive the device address immediately after your poll or select address)	N/A
			The system return this value to a multipoint tributary only after an ?SRCV initial call	N/A
?SBYM	Maximum buffer length	Byte length of receive buffer	Unchanged	N/A
?STOV	Time-out override value	Length of time the system will wait for the remote station to respond, before the system enters its BSC recovery procedure	Unchanged	(-1) 8 seconds

(concluded)

Required Input

Set offsets ?SBUP and ?SBUL to the address of the receive buffer you've set aside for the data, and set offset ?SBYM to the length of the receive buffer.

Specify the type of receive you're performing in offset ?SSIS, the input status word. The ?CINT mask defines the call as a receive initial, and the ?CONT mask defines it as a receive continue. The masks ?SWAK, ?SPET, ?SPRV, and ?SPNK define special types of calls that can be used only by multipoint tributary stations. They may be selected in place of a ?CINT call where appropriate. Be sure to set the correct mask. You cannot issue two receive initial calls in a row, nor can you issue a receive continue before you've issued a receive initial.

Receive Continue Calls

For receive continue calls, you must set one of the *response type* masks in offset ?SSIS. The response type field defines the control character the system will use to reply to the sending station after each transmission. The response type options are

- return the sender a negative acknowledgement (?SNAK)
- return the sender a positive acknowledgement (ACK0 or ACK1) in the correct alternating sequence (?SACK)

- return the sender an ACK0 positive acknowledgement (?SAK0)
- return the sender an ACK1 positive acknowledgement (?SAK1)
- return the sender a reverse interrupt (?SRVI)

Notice that there are three variations for a positive acknowledgement. Under standard BSC protocol, the system alternates the ACK0 and ACK1 characters; that is, if a station uses ACK0 as its first positive acknowledgement, it will use ACK1 as its second positive acknowledgement, and vice-versa. The ?SAK0 and ?SAK1 options let you specify a single positive acknowledgement character 1or the duration of the ?SRCV call.

Response type ?SRVI directs AOS to reply with an RVI (reverse interrupt) character. Use this response type if you want to interrupt the transmission to pass the sending station a high priority message.

You can continue to issue receive continue calls until you've obtained all the sending station's data. You will know the transmission has ended when the system takes the ?SRCV exception return and passes error code EREOT to AC0.

If you issue a receive continue call and the sending station fails to respond or responds inappropriately, the system enters its BSC recovery procedures.

?SRCV From a Point-to-Point Station

If you're issuing the ?SRCV as a *receive initial* call from a point-to-point station, the system waits for a bid sequence from the other station before executing the ?SRCV. By default, the system waits 8 seconds. You can set an alternate time-out value in offset ?STOV.

If the other station fails to send a bid sequence in time, the system enters the BSC error recovery procedure for receive initials. Refer to the end of this chapter for a list of the error recovery procedures. If the system receives the other station's bid in time, the ?SRCV call succeeds, and the system returns the first block to your receive buffer.

?SRCV From a Multipoint Control Station

When you issue the ?SRCV call from a control station on a multipoint line, the system performs either a general poll to solicit data from all your enabled tributaries or a specific poll to solicit data from one tributary. Your input to AC0, prior to the ?SRCV call, determines what kind of polling the system will perform.

On a *receive initial, general poll,* the system sends each entry in the polling list down the BSC line, starting with the entry for the first relative terminal (tributary). If a tributary has no data to send, it responds to its poll address with an EOT character, and the system continues polling.

A general polling operation ends when one of the following occurs:

- a tributary sends data to the control station (the system then places the data in the control station's receive buffer and loads AC1 with the sender's relative terminal number)
- the system reaches the end of the polling list without receiving data (the system takes the exception return from the ?SRCV call and passes error code EREPL to AC0)
- a tributary fails to respond within the time value specified in packet offset ?STOV (the system takes the exception return, with error code ERTOF), or a tributary responds inappropriately (the system takes the exception return, with error code ERUNI)

Receive continue calls from multipoint stations work the same as receive continues from point-to-point stations. See the "Receive Continue Calls" section for details.

?SRCV From a Multipoint Tributary

Issuing an ?SRCV receive initial call from a tributary signals the system that the tributary is prepared to accept data from the control station. The system responds by monitoring the line for your tributary's poll or select address. If the control station fails to send the poll or select address within the interval you set in ?STOV, the call fails and the system returns error code ERTOF to AC0.

If your tributary was polled in time, the system sets bit ?SPLR in offset ?SSIS and takes the normal return from the ?SRCV call. You should respond by issuing an ?SSND send continue call if you have data to send or an ?SSND EOT (end-of-transmission) if you do not.

If there is no data to send, but you wish to continue monitoring the line for another occurrence of a poll or select sequence, issue a single ?SRCV call with the ?SPET mask in ?SSIS of the call packet. This will cause the tributary's system to send an EOT in response to the control station's poll, and then immediately resume monitoring the line for either a poll or a select address. AOS will not return from the ?SRCV call until some condition occurs that would cause a return to an ordinary ?SRCV initial call for a multipoint tributary.

Similarly, issuing a single ?SRCV call with the mask ?SPWK, ?SPRV, or ?SPNK set will send a WACK, RVI, or NAK (respectively) as a response to a control station, and then will immediately resume monitoring the line for the appropriate poll or select address.

Exercise caution when using one of these ?SRCV call types. Determine what action the control station is designed to take should it receive a WACK, RVI, or NAK in response to a poll or select it has issued.

If your tributary was selected in time, the system sets bit ?SSLR in offset ?SSIS and takes the normal return. If you're ready to receive data, respond to the select sequence with an ?SRCV receive continue which has one of the positive response bits set (?SACK, ?SAK0, or ?SAK1). ?SAK0 and ?SAK1 override the system's usual ACK0 and ACK1 alternation, and select a specific positive response (ACK0 or ACK1) for this receive call.

These parameters also reset the system's alternation pattern. For example, if you select ?SAK0, the next positive response will be ACK1, unless you override the alternation pattern again.

If you are not ready to receive data, issue an ?SSND continue with the call type set to ?SWAK (wait-before-transmission, positive acknowledgement.)

?SRCV Output Values

On a normal return from an ?SRCV call, the system sets certain masks in packet offset ?SSIS to indicate the text mode of the transmission and the presence of several data-link control characters or poll, select, or device addresses. The previous section described two of these flags, ?SPLR and ?SSLR. Table 10-5 lists all the masks and describes their meanings.

The system sets the ?SITB flag when one or more ITB (intermediate text block) characters delimit certain portions of the text block. When a block contains these characters, the system precedes each intermediate text block portion with a value indicating its length. Figure 10-7 pictures a receive buffer consisting of three intermediate text block portions with lengths of 4, 5, and 6 characters, respectively.

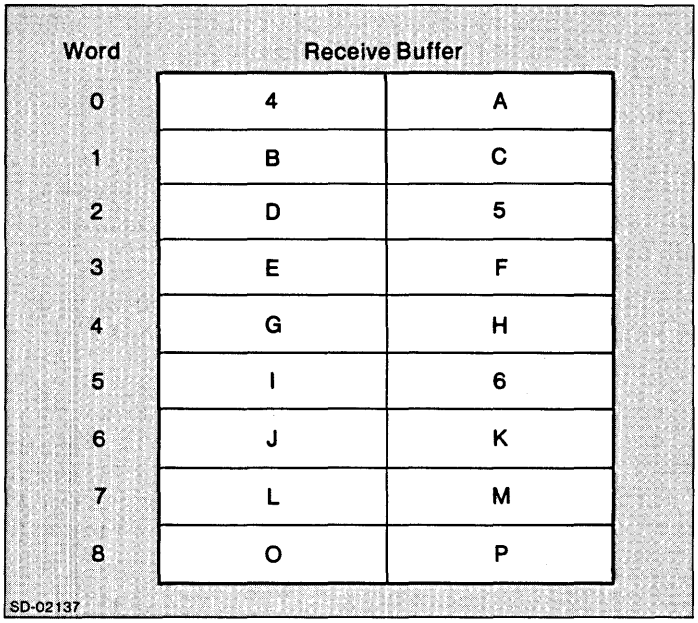


Figure 10-7. ITB Receive Buffer Format

Table 10-5. Masks Returned on ?SRCV Calls

Mask	Information Returned	Meaning
?TRAN	Text mode	If set, transparent text received
?SOHB	Start of block	If ?SOHB set, start of header (SOH) character received
?STXB		If ?STXB set, start of text (STX) character received
?ETBB	End of block	If ?ETBB set, end of transmission block (ETB) character received
?ETXB		If ?ETXB set, end of text (ETX) character received
?SITB	ITB received	If set, end of intermediate transmission block (ITB) character received
?SPLR	Poll/select indicator	If ?SPLR set, poll address received
?SSLR		If ?SSLR set, select address received
?SDAC	Device/component selection	If set, device address received

?SSND

Send data or a control sequence over a BSC line.

?SSND
exception return
normal return

Input/Output

Input:

AC0	contains a byte pointer to a tributary's select address and device address (used by multipoint control stations, only).
AC1	contains the channel number assigned to the line.
AC2	contains the address of the ?SSND parameter packet.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERBCT	Byte count of the buffer to be sent exceeds available synchronous system buffer space.
ERBNK	Bid error (too many NAKs). (The?SSND caller received a NAK in response to its line bid (point-to-point), or its select sequence (multipoint control station).)
ERBPE	User buffer byte pointer error.
ERCNV	Conversational reply received.
ERCRC	CRC error on a data block returned in response to a data block which has been sent.
ERCTN	Contention situation while bidding. (Both stations on a point-to-point line tried to bid at the same time.)
ERDIS	Disconnect received on a switched line. (This error also occurs when a ?SSND caller receives an DLE EOT in response to its TTD sequence.)
EREOT	EOT character received. (The ?SSND caller received an end of transmission character in response to its data block or WACK character.)
ERLIS	Line in session on ?SSND call. (You tried to issue two ?SSND initial calls in a row.)
ERLNA	I/O request for disabled line.
ERNAK	Transmission failure (NAK count). (The system received a NAK in response to a data block, or a NAK WACK sequence from the receiver; this also occurs when the sending station receives a NAK in response to an ENQ (repeat previous response), and the retry limit is exceeded.)
ERNSL	Attempt to enable non-sync line.
ERRVI	RVI response received. (An RVI character received in response to the sender's bid (point-to-point only) or select sequence (multipoint, only), or following the receipt of a WACK.

?SSND (continued)

ERSCS — ?SSND continue without line in session. (You tried to issue an ?SSND continue without a previous ?SSND initial.)

ERSIM — There is an outstanding call on this line.

ERTOF — Transmission failure. (The time-out value has elapsed.)

ERTRF — Transmitter failure.

ERUNI — Uninterpretable response received.

ERVBP — Invalid byte pointer passed as a system call argument.

ERWAK — A WACK character received in reponse to a line bid (point-to-point only) or a select sequence (multipoint only).

Description

?SSND lets the calling station send a block of data or data-link control characters over an enabled BSC line. A process can issue this call to bid for the BSC line (send initial), to bid for the line and then send data (send continue), or to send data (send continue).

Before issuing this call, set up the ?SSND parameter packet in your logical address space, load AC2 with its address, and load AC1 with the line's channel number. Figure 10-8 shows the structure of the ?SSND packet, and Table 10-6 describes each offset.

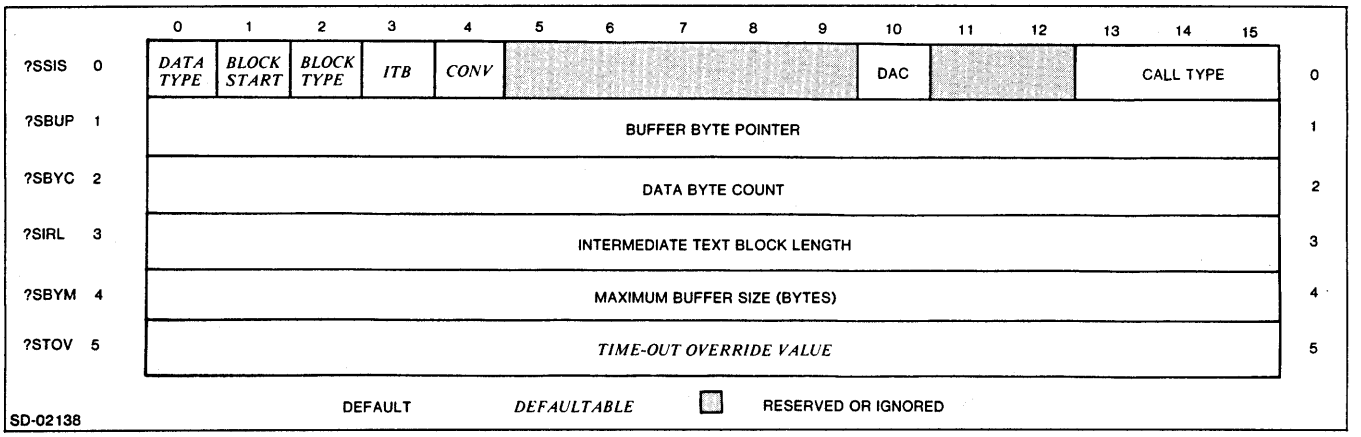


Figure 10-8. ?SSND Parameter Packet: Structure

Table 10-6. ?SSND Parameter Packet

Offset	Contents	Input Value		Output Value	Default
?SSIS	Input Status word:				
	Data type:	?NTRN	-Nontransparent data mode	Unchanged	?NRN
		?TRAN	- Transparent mode		
	Block start:	?STXB	- Start of text.	Unchanged	?STXB
		?SOHB	- Start of header		
	Block type:	?ETBB	- End of block	Unchanged	?ETBB
		?ETXB	- End of text		
	Send intermediate text blocks:	?SITB		Unchanged	No ITBs
	Accept conversational replies:	?SCON		Unchanged	Do not accept conversational replies
	Send device address characters (for select sequences only)	?SDAC		Unchanged	Do not send device address
	Call type:	?CINT	- Send initial (with data)	Unchanged	None
		?CONT	- Send continue (with data)		
		?SWAK	- Send WACK sequence		
		?SEOT	- Send EOT sequence		
		?SDET	- Send DLE ETB or DLE ETX sequence (to signal) end of transparent text mode)		
		?STTD	- Send TTD sequence (to signal temporary text delay)		
?SRES	Reserved	Must be 0		N/A	N/A
?SBUP	Buffer byte pointer	Byte pointer to data		you're sending	Unchanged *
?SBYC	Buffer byte count	Number of bytes to send		Unchanged	N/A
?SIRL	ITB record length	Length (in bytes) of intermediate text blocks (records delimited by an ITB character)		Unchanged	(0) No ITBs
?SBYM	Maximum buffer length	Byte length of data buffer		Unchanged	N/A
?STOV	Time-out override value	Length of time the system will wait for a reply before entering BSC recovery procedures		Unchanged	(-1) 8 seconds

?SSND (continued)

Input Status Word

Packet offset ?SSIS, the input status word, defines certain characteristics of the block you will send.

The *data type* field specifies whether the data is in transparent text mode (?TRAN), or nontransparent mode (?NTRN). Under transparent text mode, most data-link control characters are treated only as bit patterns; that is, they have no control significance. The exceptions are DLE STX, which signals the beginning of transparent text mode, and DLE ETB or DLE ETX, which signal the end of transparent text mode.

Although the system inserts the necessary data-link characters, including the BCC character, when you issue an ?SSND call, three fields in offset ?SSIS let you specify the data-link characters you want to use.

The *block start* field specifies whether the operating system should preface the block with an STX (start of text) character or with an SOH (start of header) character. Set mask ?SOHB if you're sending header information in this transmission and ?STXB if you're sending text. The ?STXB mask is the default. The *block type* field directs the system to append either an ETB (end of text block) or an ETX (end of text) character to the data block. The ETX character terminates the last text block in a message. Thus, you should set mask ?ETXB if this is the last block you're sending. The default for this field is ?ETBB. This mask corresponds to the ETB character, which must terminate every text block except the last.

The *send intermediate text block* field directs the system to delimit portions of the data block with ITB (intermediate transmission) characters. If you select this option (by setting mask ?SITB), you must set offset ?SIRL to the length of the intermediate text blocks. If you set ?SIRL to 4, for example, the system places an ITB character after every fourth byte in the text block. The system ignores the contents of ?SIRL if bit ?SITB is not set.

Select the ?SCON mask if you want to accept *conversational replies* from the receiving station. Conversational mode allows the receiver to respond to the block you send with header information or data, instead of a positive ACK0 or ACK1 sequence. Note that the block you send must be a complete data block (not a data-link sequence) for conversational mode to work. In general, you should not give a conversational reply to a header block or a block that ends with an ETB character.

The *call type* field defines the ?SSND call as one of the following types:

(?CINT)	a send initial with data
(?CONT)	a send continue with data
(?SWAK)	a wait-before-transmission (positive acknowledgement)
(?SEOT)	an end-of-transmission
(?SDET)	a line disconnect call
(?STTD)	a temporary text delay

Table 10-7 describes each type of call.

Table 10-7. ?SSND Call Types

Station type	Call type	System action
Point-to-point	Send inital (with data)	Send ENQ (line bid) to the receiving station, and await an ACK0 response. a. If no ACK0 within time-out interval (?STOV), then enter BSC recovery procedure. If ACK0 within the time-out interval, then ?SSND data block, wait for ACK1 response. b. If no ACK1 within time-out interval, then enter BSC recovery procedure. If ACK1 within timeout interval, then take ?SSND normal return.
	Send continue (with data) (?CONT)	Send data to receiving station and await correct alternating response (ACK0 or ACK1). a. If no response within time-out interval then enter BSC recovery procedure. b. If correct response, then take ?SSND normal return.
All types (point-to-point and multipoint)	WACK sequence	Send WACK to receiver, and await an ENQ reply. a. If no ENQ within time-out interval, then enter BSC recovery procedures. b. If ENQ within time-out interval, then take ?SSND normal return.
	EOT sequence or DLE EOT (a switched line disconnect)	Send control sequence and take ?SSND normal return. Calling station may now issue ?SSND initial, ?SRCV initial, or ?SDBL, (disable line) call.
All types (point-to-point and multipoint)	Send continue with temporary text delay (?STTD)	Send TTD sequence, and await NAK reply. a. If receiver responds with DLE EOT, then take ?SSND exception return (error code ERDIS) and enter BSC error recovery procedures. b. If receiver responds with other than NAK or DLE EOT, take ?SSND exception return on error ERUNI and enter BSC error recovery procedures. c. If NAK within time-out interval, take ?SSND normal return.

(continues)

Table 10-7. ?SSND Call Types (continued)

Station type	Call type	System action
Multipoint control station	Send initial (with data) (?CINT)	Send select sequence (pointed to by AC0) to receiver, await ACK0 response. a. If no ACK0 within time-out interval, then enter BSC recovery procedure. b. If ACK0 (acknowledging select sequence), then send data block and await ACK1. c. If no ACK1 within time-out interval, then enter BSC recovery procedure. d. If ACK1, then take ?SSND normal return.
	Send continue (with data) (?CONT)	Functionally identical to point-to-point send continue calls.
Multipoint tributary	Send initial (with data)	Not applicable. (You will never issue an ?SSND initial if you are a multipoint tributary; instead, you issue an ?SRCV initial to monitor the line for your poll address. If you are polled and have data to send, issue an ?SSND send continue and send the first block of data.)
	Send continue (with data)(?CONT)	Functionally identical to point-to-point send continue calls.

(concluded)

Other ?SSND Offsets

Offset ?SBUP points to the buffer you've reserved in your logical address space for the data you will send. Note that the buffer byte count you supply in offset ?SBYC must be for the entire contents of the data buffer. If the data consists of intermediate text blocks (defined by ?SITB and ?SIRL), the data buffer should contain an exact multiple of the intermediate block length you specified in offset ?SIRL. Otherwise, the last intermediate text block will be shorter than the others, since the system does not pad the last intermediate text block.

If you set both ?SBUP and ?SBYC to 0 on a send initial, the system interprets your send call as a bid for the line. If the bid succeeds, the system simply takes the ?SSND normal return, without sending data. If you set either of these parameters, *but not both,* to 0 the system takes the ?SSND exception return and passes one of the following error codes to AC0:

Error Code	Input
ERBPE	?SBUP =0 ?SBYC nonzero
ERBCT	?SBUP nonzero ?SBYC =0

When you select the conversational reply option (?SCON), the system stores the response data from the receiver in the data buffer defined by ?SBUP, and records the length of the response data in offset ?SBYC.

BSC Error Recovery Procedures

When AOS receives an inappropriate response to an ?SSND or ?SRCV call or does not receive a reponse within the time-out interval specified in offset ?STOV, it enters its BSC recovery procedures. The error recovery procedures differ, depending on which operation was underway when the error occurred. In addition, the system's action within each recovery procedure depends on the cause of the error.

In most cases, the system responds to a BSC error by trying the particular procedure again, repeatedly, if necessary, until its retry count is exhausted. The retry count is a system-maintained variable; it is outside the user's control.

Table 10-8 describes the system's error recovery procedures for the various types of send and receive calls.

Table 10-8. BSC Error Recovery Procedures

Call Type	System Action
Send Initial	
Time-out NAK, or inappropriate response.	Resend ENQ, unless retry count exceeded. If retry count exceeded, take ?SSND exception return. (Possible errors in AC0, ERTOF, ERNAK, ERUNI.)
ENQ	If calling station is the primary, resend ENQ. If calling station is the secondary, take ?SSND exception return. (Error in AC0: ERCTN.)
Send Continue	
Time-out or inappropriate response	Send ENQ, unless retry count exceeded. if retry count exceeded, then take ?SSND exception return. (Possible errors in AC0: ERTOF, ERUNI.)
NAK	Resend data, unless retry count exceeded. If retry count exceeded, then take ?SSND exception return. (Error in AC0: ERNAK.)
Receive Initial (point-to-point and multipoint tributary)	
Time-out, or inappropriate response	Retry receive initial, unless retry count exceeded. If retry count exceeded, then take ?SRCV exception return. (Possible errors in AC0:ERTOF,ERUNI.)
Receive Continue	
Time-out or inappropriate response	If retry count exceeded, take ?SRCV exception return (possible errors in AC0:ERTOF,ERUNI); otherwise, await ENQ from sender (assuming that the sender will issue a time-out and send an ENQ).
ENQ	Resend last response and attempt receive continue; unless retry count exceeded. If retry count exceeded, take ?SRCV exception return. (Error in AC0:ERENQ.)
CRC (block check) error	Send NAK and attempt receive continue, unless retry count exceeded. If retry count exceeded, take ?SRCV exception return. (Error in AC0: ERCRC.)

(continues)

Table 10-8. BSC Error Recovery Procedures

Call Type	System Action
Receive Initial (multipoint control station)	
Time-out or inappropriate response	Retry receive initial, unless retry count for particular relative terminal is exceeded. If retry count exceeded, take ?SRCV exception return. (Possible errors in ACO: ERTOF, ERUNI.)
EOT	If a polled terminal responds with EOT, step to the next relative terminal on the polling list and continue polling. If end of the polling list is reached, take ?SRCV exception return. (Error in AC0: EREPL.)

(concluded)

BSC Implementation

The figures in this section illustrate how AOS implements BSC protocol using the BSC system calls. Before you read this section, refer to the system call descriptions and to the definitions of the BSC data-link control characters. The figures describe the following system calls:

?SEBL	Point-to-point	Figure 10-9
?SSND Initial	Point-to-point	Figure 10-10
?SSND Continue	Point-to-point	Figure 10-11
?SRCV Initial and Continue	Point-to-point	Figure 10-12
?SSND	Multipoint control station	Figure 10-13
?SRCV	Multipoint control station	Figure 10-14
?SRCV	Multipoint tributary	Figure 10-15

Recall that you cannot issue a send initial call from a multipoint tributary, and that receive continue calls from multipoint stations and from point-to-point stations are identical.

You will notice that each figure has three columns. The first column represents the calls you issue, with their normal and exception returns. The second column illustrates the actions the operating system takes, and the third column shows the actions the remote station takes.

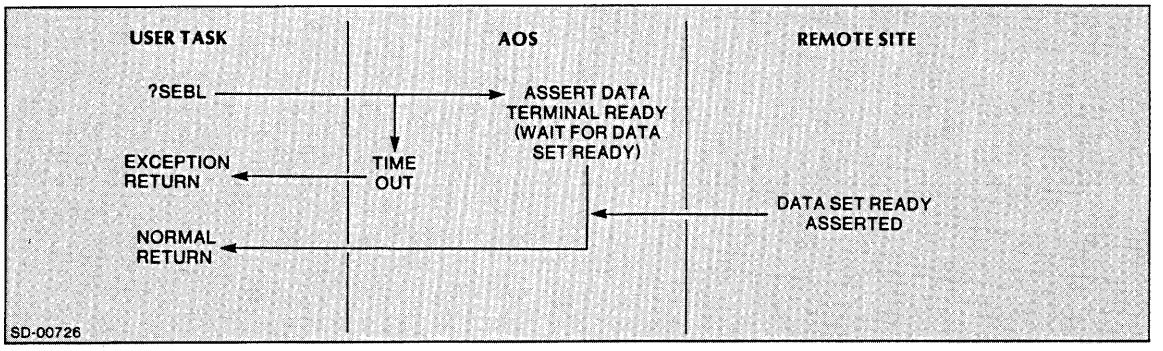


Figure 10-9. ?SEBL Call, Point-to-Point

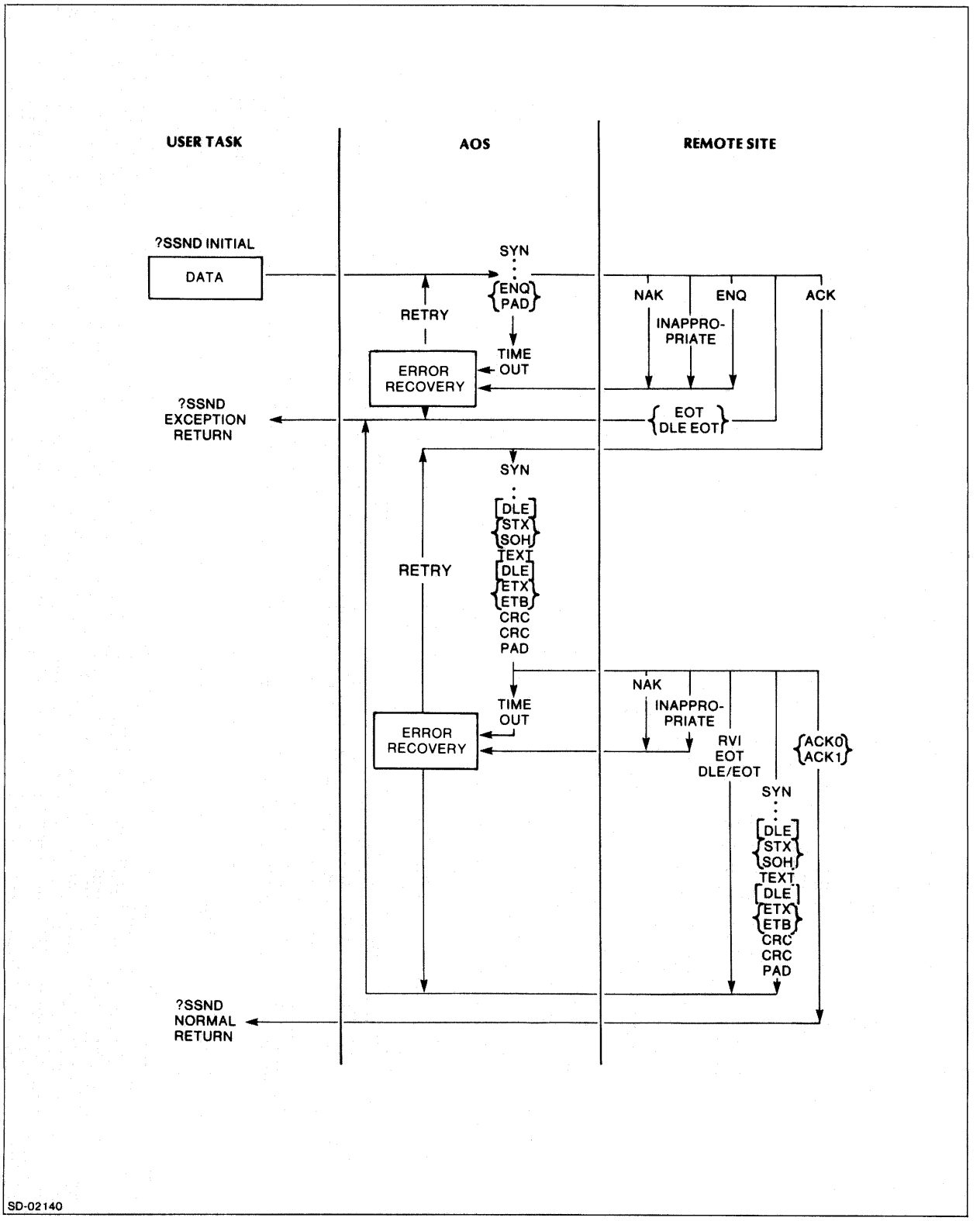


Figure 10-10. ?SSND Initial, Point-to-Point

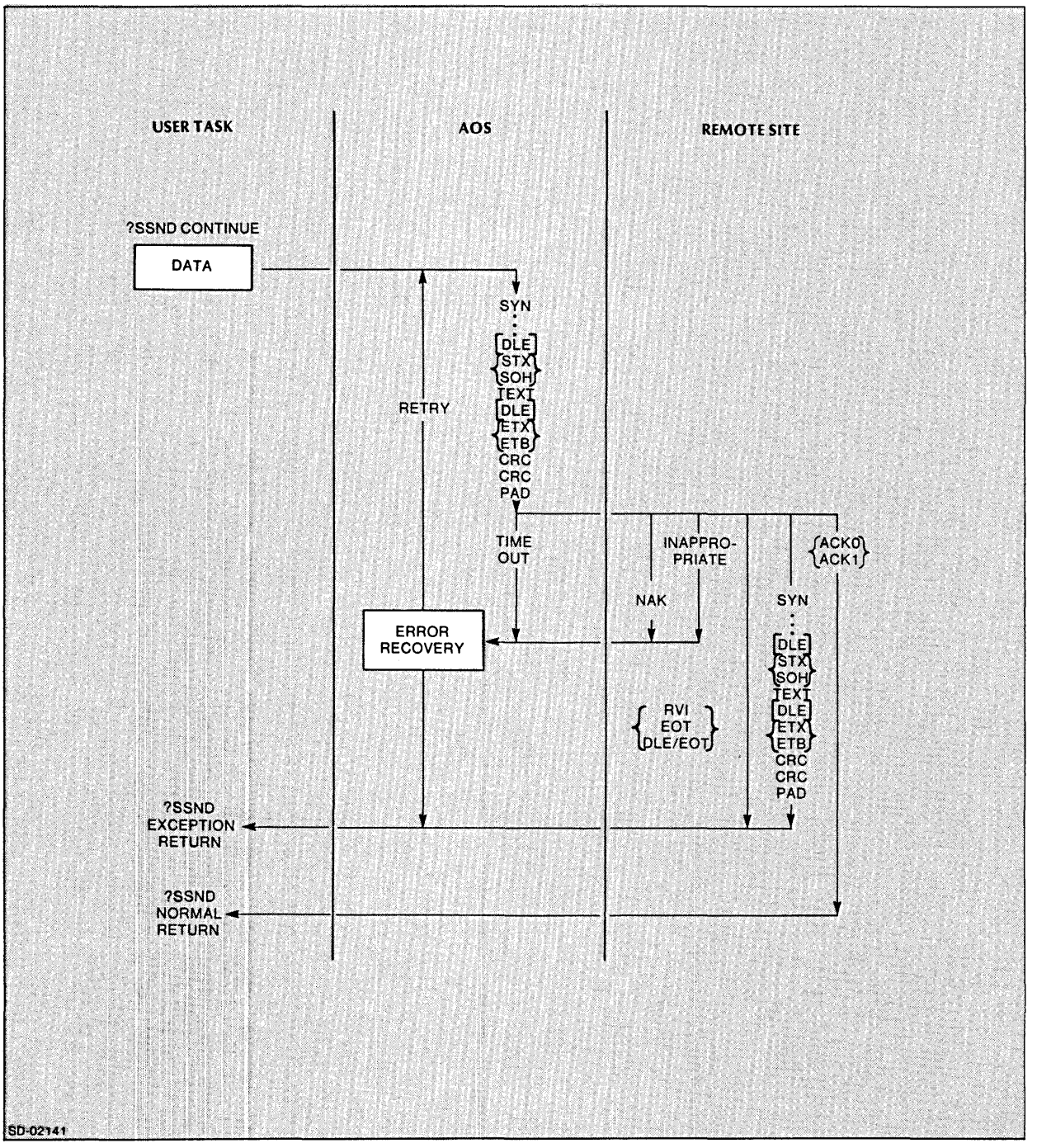


Figure 10-11. ?SSND Continue, Point-to-Point

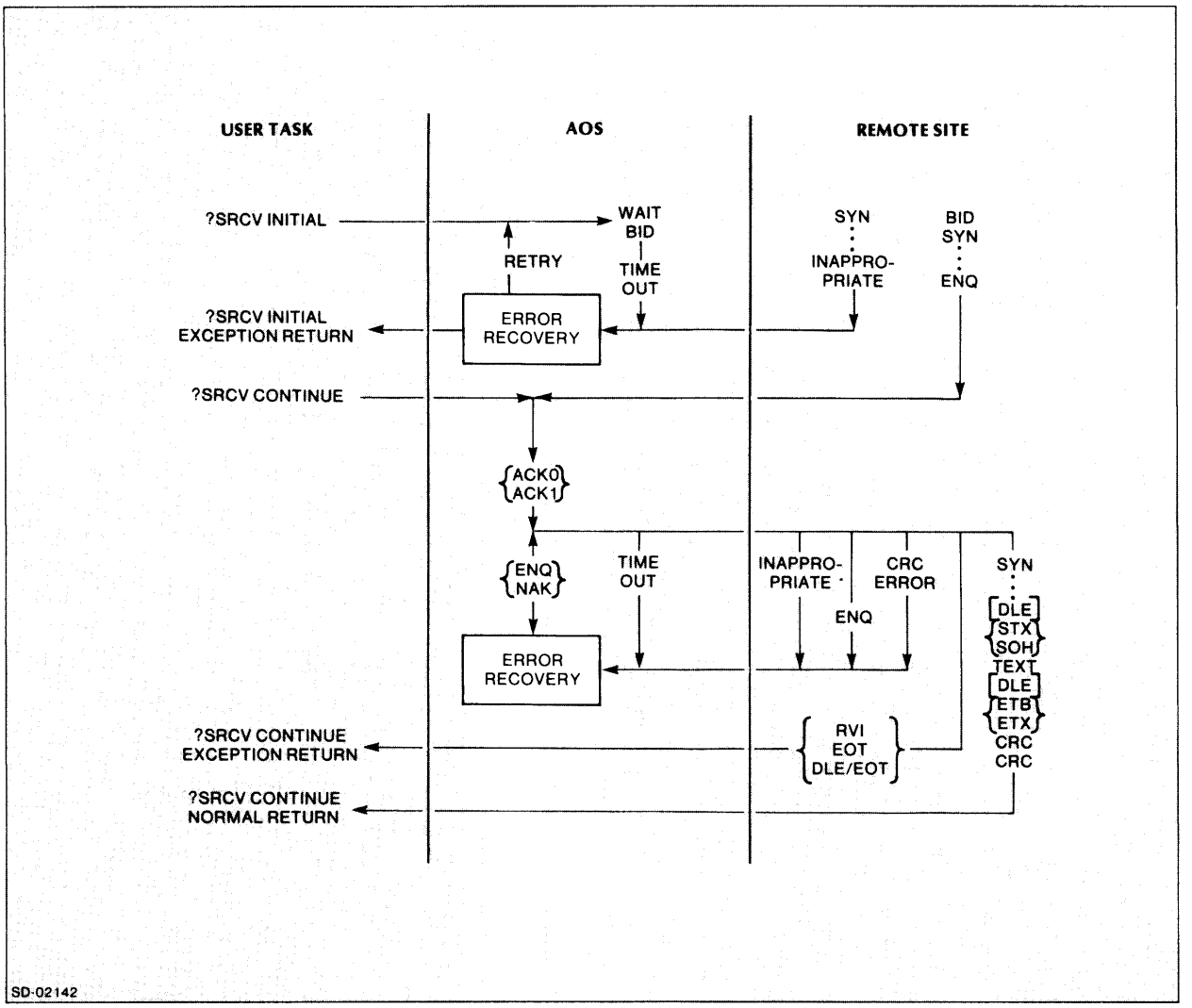


Figure 10-12. ?SRCV Initial and Continue, Point-to-Point

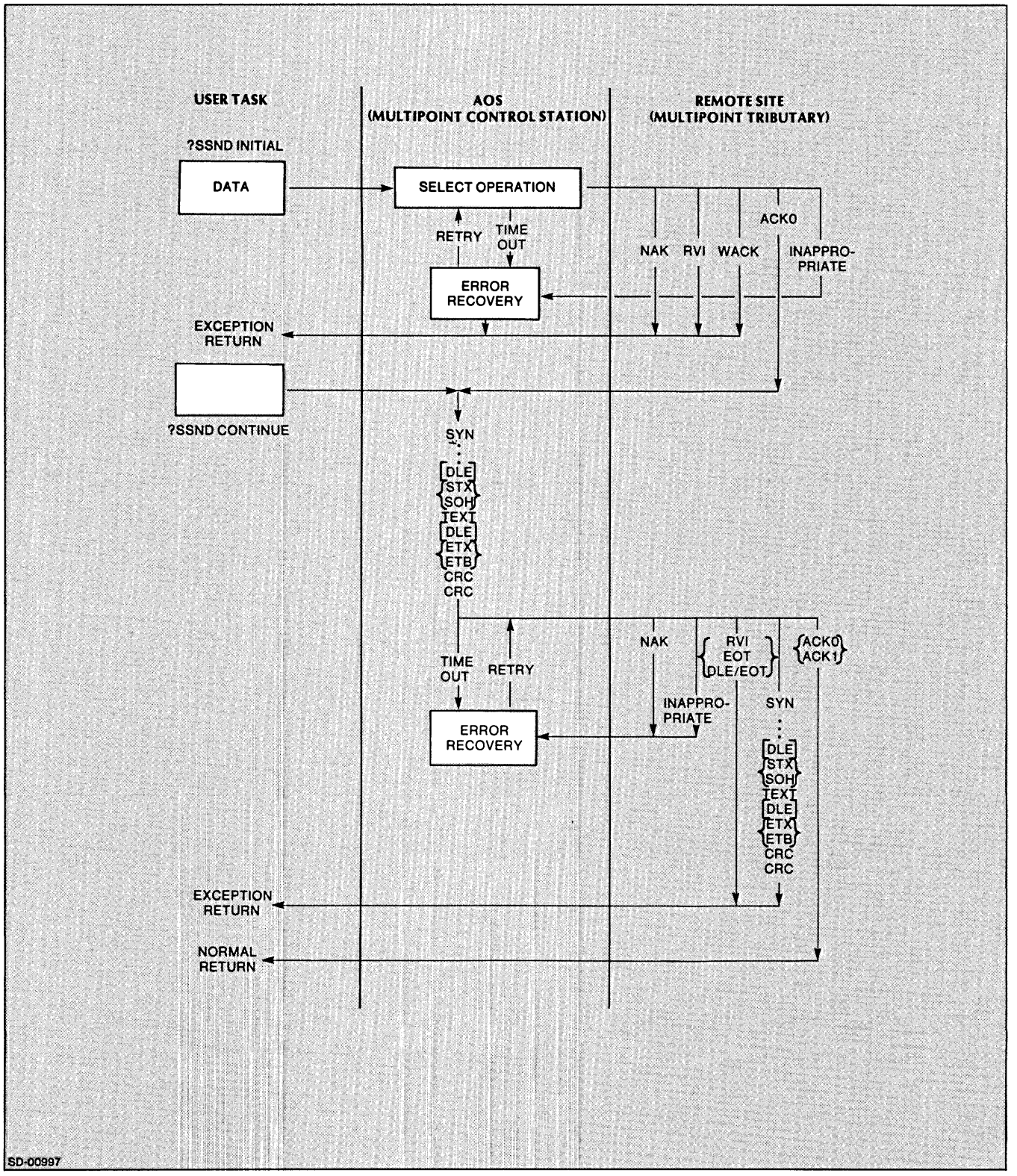


Figure 10-13. ?SSND, Multipoint Control Station

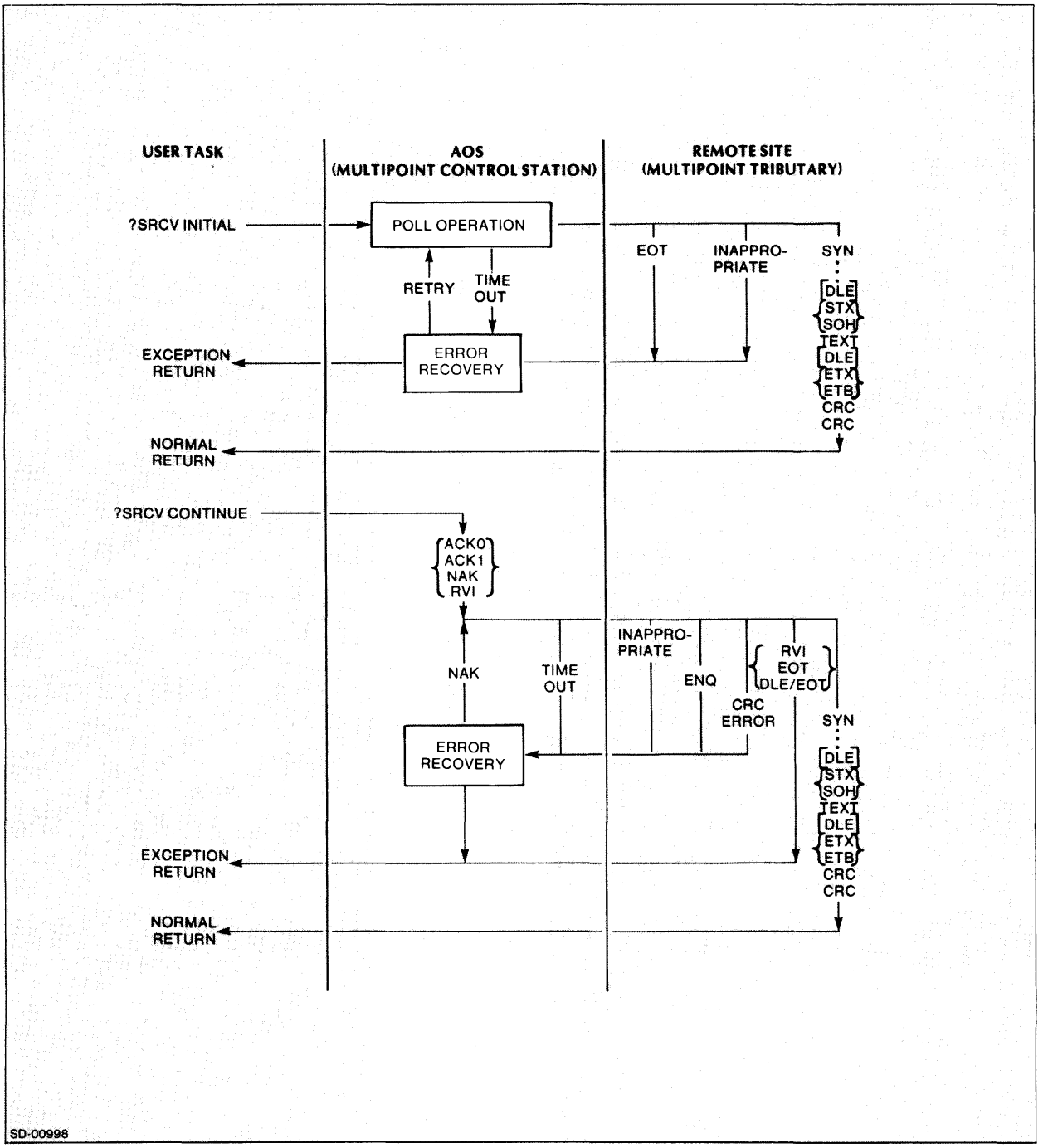


Figure 10-14. ?SRCV, Multipoint Control Station

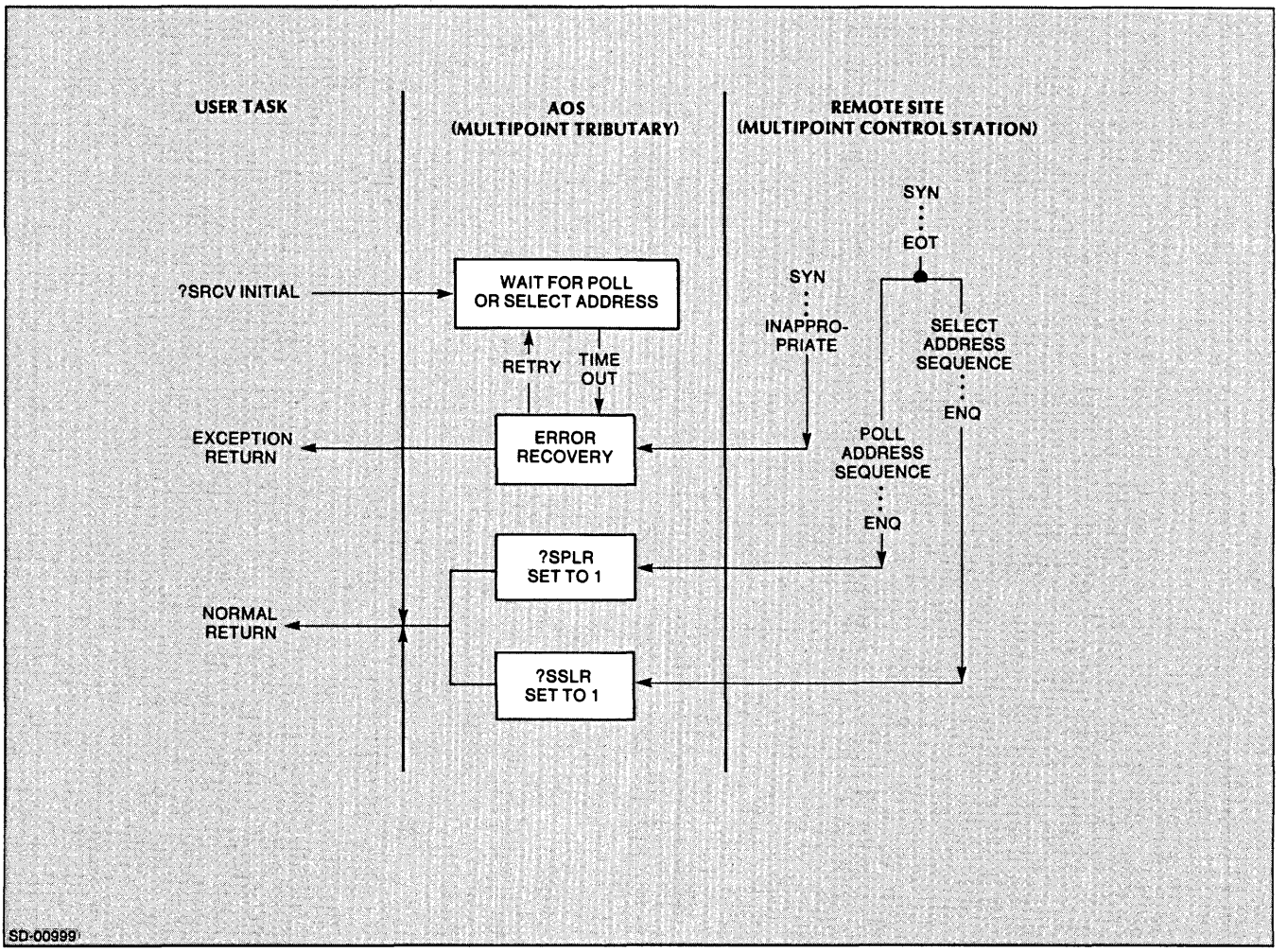


Figure 10-15. ?SRCV, Multipoint Tributary

End of Chapter

Chapter 11
Connection Management

The AOS connection management facility lets you establish a customer-server relationship between processes, and use the server process to perform certain functions on behalf of its customers. Typically, a server process performs generalized routines accessible to customer processes. For instance, you might create a server process to build files, perform I/O, or establish networking protocol for customer processes.

Connection management provides two distinct advantages:

- Customers do not need the IPC privilege to issue ?IS.R calls against the server (to send an IPC message to the server and receive an IPC message in reply)
- Servers can move bytes to and from their customers' buffers

Making a Connection

You establish a connection between two processes by defining one process as the server and the other as the customer. The system call ?SERVE defines the calling process as a server. The ?CON system call defines a customer and establishes the logical connection between the customer and an existing server. The model configuration in Figure 11-1 shows a server process with connections to three customer processes.

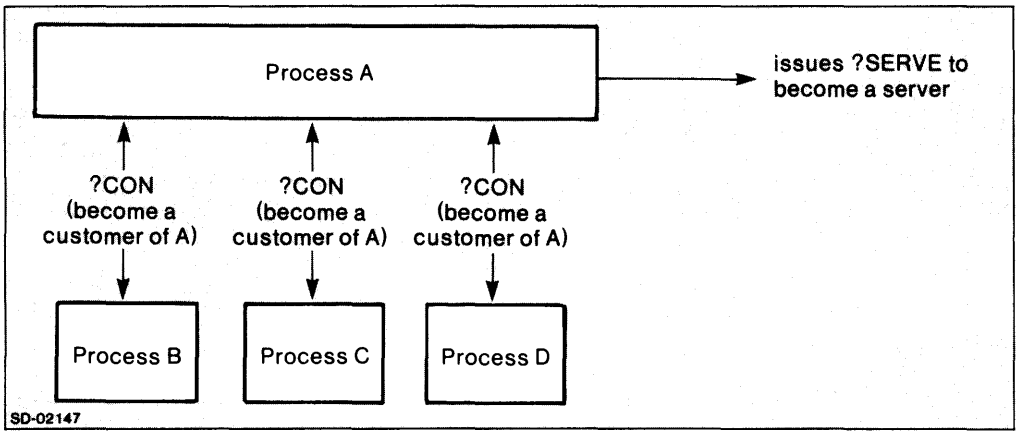


Figure 11-1. Model Customer/Server Configuration

AOS maintains a *connection table* to manage exchanges between customers and servers. When a customer establishes a connection (?CON) with a declared server, the system writes an entry in the connection table specifying the PIDs of the server and customer. Each ?CON call generates one connection table entry. *

A process can act as a server for other processes and as a customer of other servers as long as it issues the appropriate number of ?SERVE and ?CON calls. Figure 11-2 shows a multilevel connection, where process A is the server of processes B, C, and D, and a customer of process X. Multilevel connections let you set up intermediate servers for some functions, and one or more superior servers for other functions.

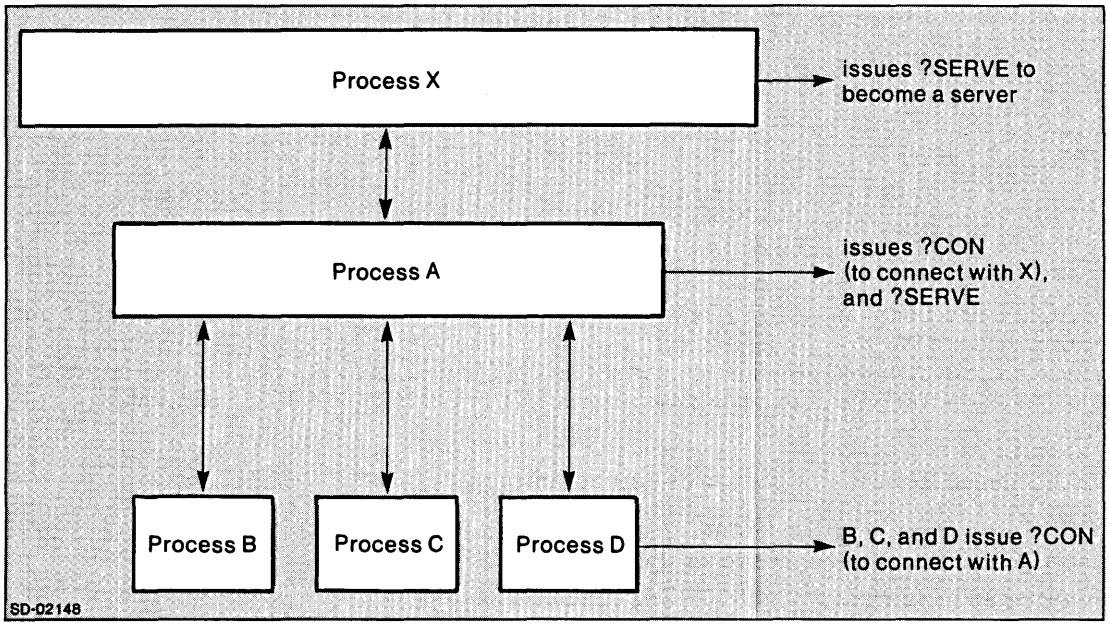


Figure 11-2. Multilevel Customer/Server Configuration

You can also establish a double connection between two processes. A *double connection* is one in which each process can act as either the customer or the server of the other, depending on the action to be performed. As Figure 11-3 illustrates, a double connection requires two ?SERVE calls and two ?CON calls. The system creates two connection table entries, one for each ?CON.

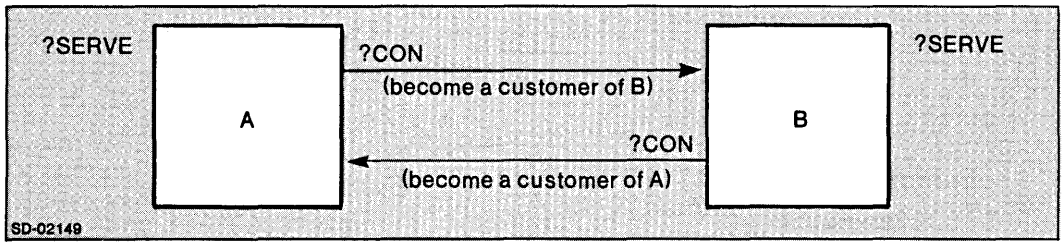


Figure 11-3. Double Connection

Server Process

Once a process has server status (established with ?SERVE), it can issue the following calls:

?CTERM	Terminate a customer
?MBFC	Move bytes from a customer's buffer
?MBTC	Move bytes to a customer's buffer
?PCNX	Pass a connection to another server
?RESIGN	Resign as a server
?VCUST	Verify a customer

* ?CTERM terminates a customer process.

The ?MBTC and ?MBFC calls allow the server to move bytes to and from a customer's logical address space. Before executing either of these calls, the system checks the connection table to verify that a valid connection exists between the two processes.

The ?PCNX call passes a customer-server connection from one server to another and directs the system to revise the connection table entry accordingly. This call is useful for passing a valid customer from a dispatching server to a specialized server process.

?RESIGN signals the system that the caller has resigned as a server.

?VCUST determines whether a target process is a customer of the ?VCUST caller. If the process is not a customer, the system takes the exception return and passes error code ERCDE to AC0. If the connection between the two has been broken, the ?VCUST call fails on error code ERCBK.

Typically, server processes communicate with their customers via the IPC calls ?ISEND, ?IREC, and ?IS.R. Note that customer processes do not need the IPC privilege to issue ?IS.R calls to their servers, and server processes do not need the IPC privilege to issue ?ISEND or ?IS.R calls to their customers.

Chaining

Using the ?CHAIN call, a customer process can execute a new program without affecting its connection with the server. When a customer process issues ?CHAIN, the operating system notifies the server via an IPC message. The server receives the message by issuing the ?IREC call against the global origin port ?SPTM. (See Receiving Process Termination Messages, Chapter 4.)

Note that if the customer process issues ?CHAIN, its address space changes, and any services that the server is currently performing that directly access the customer's address space could result in an error.

Breaking a Connection

A connection is a two-sided relationship including a customer process and a server process. Either side may break a connection at any time. If one side breaks a connection, the system will record the fact in the connection table. However, the system will not clear the connection from the table until it has been broken from both sides. A process can break a connection from its own side in two ways: by terminating, or by issuing a connection-breaking system call.

When a process terminates, the system automatically breaks all of the process's connections from the process's side. A process may terminate by its own actions, either intentionally or unintentionally, or by another process's actions. For example, a customer process may terminate because

- it intentionally decides to ?RETURN
- it unintentionally traps
- its server terminates it with a ?CTERM call
- a third process terminates it with a ?TERM call

A process can terminate when it or another process issues one of the following process termination system calls:

?BRKFL	Create a break file (for the calling process)
?CTERM	Terminate a customer (a server call only)
?RETURN	Terminate the calling process and return control to its father
?TERM	Terminate a process

Except for ?CTERM, these calls are discussed in Chapter 2.

A process can break a connection without terminating by issuing one of the following connection-breaking system calls:

?DCON	Break a connection
?RESIGN	Resign as a server (a server call only).

Notice that the ?CTERM and ?RESIGN calls are *server only* calls. The other termination and connection-breaking system calls are available to both servers and customers.

When the system detects a broken connection, it sets a flag bit in the appropriate connection table entry. For the system to actually clear the entry, however, it must receive disconnects from *both* the customer and the server. For example, a customer could issue ?DCON to break its connection with the server, but the customer-server entry will remain in the connection table until the server also breaks the connection. In general, both processes in a connection should break the connection as soon as it has served its purpose. This keeps the connection table entries within the maximum range and allows the system to reassign the PIDs. (The maximum number of connections allowed under AOS is revision dependent.)

Obituary Message

When a customer or server breaks a connection, AOS sends the other process an *obituary message,* a zero-length IPC message similar to the IPC message sent to a father process when its son terminates. A customer can suppress the obituary message by setting bit ?MCOBIT in AC1 when it issues the ?CON call. (See the description of ?CON for details.) A server cannot suppress the obituary notice.

To receive an obituary message, a process must issue an ?IREC call; it must specify 0 and ?SPTM in ?IREC offsets ?IOPH and ?IOPL, respectively (origin port), and 0 in offset ?IDPN (destination port). The system returns the obituary message as termination code ?TBCX in offset ?IUFL of the ?IREC header.

If a process breaks a connection by terminating, the system sends an obituary message to the other party in the connection, and a standard termination message to the father. Thus, if a server is also the father of its customer, it must issue two ?IREC calls in order to receive both messages when the customer terminates.

The system sends a similar IPC message to the server when one of its customers ?CHAINs to a new program. Again, the server must issue an ?IREC against port ?SPTM to receive the chain message. The system returns the chain message as termination code ?TCCX in offset ?IUFL of the ?IREC header.

A process may break many connections with one action. For example, if a process terminates, the system will break all its connections. If a server resigns, the system will break all the connections for which it was a server. And if a process involved in a double connection breaks the connection, the system breaks both connections. So, when a process breaks a connection, the system may send many obituary messages. For a double connection, AOS will send two obituary messages to the same process.

System Call Summary

The following list summarizes the connection management system calls:

Call	Function
?CON	Become a customer of a specified server
?CTERM	Terminate a customer process
?DCON	Break a connection (disconnect)
?MBFC	Move bytes from a customer's buffer
?MBTC	Move bytes to a customer's buffer
?PCNX	Pass a connection from one server to another
?RESIGN	Resign as a server
?SERVE	Become a server
?VCUST	Verify a customer

?CON

Become a customer of a specified server.

?CON
exception return
normal return

Input/Output

Input:

AC0 contains the PID of the server or a byte pointer to the server's process name.

AC1 contains the following optional bit masks:

?MCPID	if AC0 contains a byte pointer.
?MCOBIT	to suppress the obituary message when the server ?DCONs, ?RESIGNs or is terminated.

Output:

AC0 contains the PID of the server.

Exceptional Condition Codes in AC0

ERCCS	Cannot connect to self (AC0 contains the caller's PID, or a byte pointer to caller's process name).
ERCTF	Connection table full.
ERNAS	Process is not a server (i.e., the target process).
ERPNM	Illegal process name.
ERPRH	Attempt to access process not in hierarchy.

Description

?CON performs a dual function: it defines the calling process as a customer of an existing server process (specified in AC0), and it directs the system to create a corresponding connection table entry. If the target process is not a declared server (it has not already issued ?SERVE), the call fails on error code ERNAS. If the caller tries to connect with itself, the call fails on error code ERCCS.

By specifying ?MCOBIT in AC1, you can suppress the default obituary message sent to the customer when the server breaks the connection.

A customer can issue a ?CON call to change the state of an established connection. Such a call could change the state of obituary message suppression. Specifically, if a customer process originally requested obituary message suppression, it could issue a ?CON call to change the connection so that it would receive obituary messages. To do this, the process should first set AC1 so that it does not include ?MCOBIT, and then issue the ?CON call.

Note that although a customer can request obituary messages via this call, the process cannot request message suppression. To request message suppression, the connection must be broken and then re-established via another ?CON call.

?CON and ?SERVE are the fundamental system calls for connection management. Potential customers must issue ?CON to establish connections with existing server processes. Similarly, potential servers must issue ?SERVE to declare themselves as servers.

?CTERM

Terminate a customer process.

?CTERM
exception return
normal return

Input/Output

Input:

AC0 contains the PID of the customer to terminate or a byte pointer to that customer's process name.

AC1 contains -1 if AC0 contains a byte pointer.

Output:

none.

Exceptional Condition Codes in AC0

ERCBK Connection broken.

ERCDE Connection does not exist.

ERPRH Attempt to access process not in the hierarchy.

ERPNM Illegal process name.

ERPRV Caller not privileged for this action (The caller is not a server of the designated customer).

Description

?CTERM terminates the customer process specified in AC0, and breaks all its connections with servers. The calling process must be a server of the specified customer.

When a server issues ?CTERM, the system returns an obituary message from the global IPC port ?SPTM. The obituary message notifies the server of the customer's demise.

To receive the obituary message, the server must issue a global ?IREC against port ?SPTM. The system passes the obituary message (termination code ?TBCX) to offset ?IUFL in the server's ?IREC header. (If the server is the customer's father, it must issue two ?IREC calls: one for the obituary message, and one for the termination message.)

?CTERM allows a server process to terminate its customer, even if the customer is not in the server's hierarchy. ?CTERM differs from ?TERM (described in Chapter 2) in that ?TERM allows a server process to terminate a customer only in its own hierarchy.

Note that ?CTERM breaks the customer-server connection from the customer's side. After issuing a ?CTERM call, a server process should issue a ?DCON to clear the customer-server entry from the connection table.

?DCON

Break a connection (disconnect).

DCON
exception return
normal return

Input/Output

Input:

AC1 contains the PID of the other process.

Output

AC1 unchanged.

Exceptional Condition Codes in AC0

ERCDE Connection does not exist (three likely errors: the target process does not exist, it is not connected with the caller, or the caller issued two ?DCON calls against the same connection).

Description

?DCON breaks the connection between the calling process and the process (customer or server) specified in AC1. Both servers and customers can issue this call.

?DCON breaks the connection from either the caller's or the server's end. Note that a single ?DCON does not clear the customer-server connection from the connection table. For this to occur, both processes must break the connection.

?MBFC

Move bytes from a customer's buffer.

?MBFC *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 contains the address of the ?MBFC packet, unless specified as an argument to the call.

Output:

AC1 if the call takes an exception return, AC1 contains the actual number of bytes moved.

AC2 contains the address of the ?MBFC packet.

Exceptional Condition Codes in AC0

ERCBK Connection broken.

ERCDE Connection does not exist.

ERMPR System call parameter address error.

ERPRV Caller not privileged for this action (caller is not a server of the specified customer). *

Description

?MBFC moves a specified number of bytes from a buffer in a customer's logical address space to a buffer in the server's logical address space. The calling process must be a server of the designated customer.

Figure 11-4 shows the structure of the move bytes packet, which is used for both the ?MBFC and ?MBTC calls. The symbol ?MBLTH represents the length of the packet. Either load the packet address into AC2 before issuing the call, or cite its address as an argument to the call.

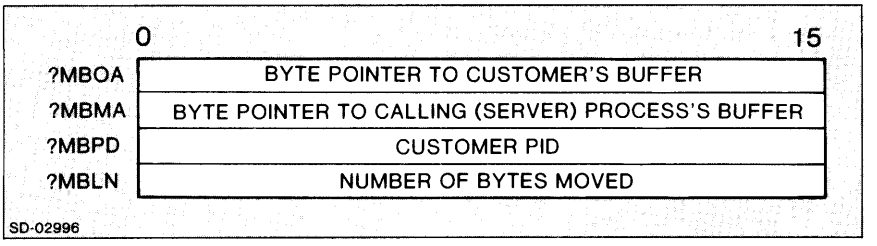


Figure 11-4. ?MBFC/?MBTC Parameter Packet

?MBFC (continued)

You can move a maximum of 2048 bytes with this call. If you specify a byte length larger than 2048, the system takes the exception return on error code ERMPR, and returns 0 to AC1 (i.e., 0 bytes moved). If the server's buffer is too small to accommodate all of the bytes, the system moves as many as it can and returns the number of bytes moved to AC1 and error code ERMPR to AC0.

We advise you not to use this call against a customer that has ?CHAINed to a new program, unless you can verify the validity of the customer's buffer. Similarly, customer processes should not issue ?CHAIN calls while they have oustanding requests to servers.

The ?MBFC call and a companion call, ?MBTC, give a server process access to a customer's logical address space. These calls are useful for any application in which the server must move bytes to fulfill a customer's request.

?MBTC

Move bytes to a customer's buffer.

?MBTC *[packet address]*
exception return
normal return

Input/Output

Input:

AC2 contains the address of the ?MBTC packet, unless specified as an argument to the call.

Output:

AC1 if the call takes an exception return, AC1 contains the actual number of bytes moved.

AC2 contains the address of the ?MBTC packet.

Exceptional Condition Codes in AC0

ERCBK Connection broken.

ERCDE Connection does not exist.

ERMPR System call parameter address error.

ERPRV Caller not privileged for this action (the caller is not a server of the specified customer).

Description

?MBTC moves bytes from a buffer in the server's logical address space to a buffer in the designated customer's logical address space. The calling process must be a server of the target customer.

?MBTC takes the same parameter packet as ?MBFC. Refer to Figure 11-4 for the structure of this packet. You can cite the packet address as an argument to the call, or you can load its address into AC2 before issuing the call.

You can move a maximum of 2048 bytes with ?MBTC. If you specify a larger byte count, the ?MBTC call fails on error code ERMPR, and the system returns 0 to AC1 (0 bytes moved). If the customer's buffer is too small to accommodate the bytes, the system moves as many as it can; it returns the number of bytes moved to AC1 and error code ERMPR to AC0.

Note that the same restriction applies to this call as to ?MBFC; that is, avoid using ?MBTC against a customer that has ?CHAINed to a new program, unless you can guarantee the validity of the customer's buffer.

?PCNX

Pass a connection from one server to another.

?PCNX
exception return
normal return

Input/Output

Input:

AC0 contains the customer's PID.

AC1 contains the PID of the new server.

Output:

AC0 unchanged.

AC1 unchanged.

Exceptional Condition Codes in AC0

ERCBK Connection broken.

ERCDE Connection does not exist.

ERCCS Attempt to connect a process to itself.

ERNAS Process is not a server.

ERPRH Attempt to access process not in hierarchy.

Description

?PCNX passes the connection established between the caller and one of its customers to another server process. Before issuing this call, load AC0 with the PID of the customer process, and load AC1 with the PID of the new server. Note that loading AC1 with the PID of the calling process passes the connection to the current server, so the net effect is nil.

The calling process must be a server of the customer specified in AC0, and there must be an unbroken connection between the two. The new server process must have already issued ?SERVE to declare itself a server.

When a server issues ?PCNX, the system updates the connection table entry so that it contains the PID of the new server.

The ?PCNX call allows a server to break its connection with one of its customers and to re-establish the connection with a new server process. ?PCNX is useful for passing a customer-server connection from an intermediate server to a specialized server.

?RESIGN

Resign as a server.

?RESIGN
exception return
normal return

Input/Output

Input:

None.

Output:

None.

Exceptional Condition Codes in AC0

ERNAS Process is not a server.

Description

?RESIGN breaks the connection between the calling process and its customers, and sends obituary messages to the customers (unless they set the ?MCOBIT option in their ?CON calls.) The calling process, which must be a server, issues this call to revoke the ?SERVE call it issued previously. If the caller is not a server, the ?RESIGN call fails, and the system returns error code ERNAS to AC0.

Note that a ?RESIGN call alone (as similarly a ?DCON call alone) does not clear the customer-server entry from the connection table. The system clears the entry only when both the customer and the server disconnect.

?SERVE

Become a server.

?SERVE
exception return
normal return

Input/Output

Input:

none.

Output:

ACs unchanged.

Exceptional Condition Codes in AC0

No exceptional condition codes are currently defined.

Description

?SERVE designates the calling process as a server, capable of performing functions on behalf of customer processes. (Potential customers connect with the server by issuing ?CON calls).

?SERVE and ?CON are the fundamental system calls for connection management. Potential servers must issue ?SERVE to declare themselves as servers. Similarly, potential customers must issue ?CON to establish connections with existing server processes.

?VCUST

Verify a customer.

?VCUST
exception return
normal return

Input/Output

Input:

AC0 PID of the process to verify.

Output:

AC0 unchanged.

Exceptional Condition Codes in AC0

ERCBK Connection broken.

ERCDE Connection does not exist.

ERNAS Process is not a server.

Description

?VCUST, a server only call, tests to see whether the process specified in AC0 is a customer of the calling process.

If the target process is the caller's customer and there is an unbroken connection between the two, control takes the normal return. If the target process is not the caller's customer, the call fails on error code ERCDE. If the target process is a customer but the connection has been broken (e.g., the customer issued a ?DCON), the call fails on error code ERCBK.

?VCUST gives you access to the system's connection table, in order to find out whether a process is a customer of the calling process. ?VCUST is also a useful way to find out if a customer process has broken the connection from its end.

End of Chapter

Chapter 12
Array Processing

Some computing applications are characterized by array operations; i.e., operations that must be performed on each element of a large data array. General purpose computers solve this problem with program loops, which have the disadvantage of requiring that the instructions in the loop body be repeatedly fetched and decoded. The array processor (AP) enables a single instruction to perform a series of operations; e.g., adding two arrays of numbers and storing the results in the third. The use of the AP leads to a substantial improvement in execution speed for array operations.

This chapter explains how to use Data General's AP system calls under AOS. This software combination allows you to write powerful AP programs in either FORTRAN 5 or DG/L, and execute those programs on AOS. *The Array Processor Software (APS) User's Manual* explains how to write APS programs.

To maintain program integrity, only one process should use the AP at a time. Since AOS cannot prevent a process from issuing AP instructions (and thereby changing the state of the AP), the system provides a voluntary protocol for scheduling AP usage. This protocol allows you to find out if anyone is currently using the AP, thereby maintaining AP resource use by one program only. To follow the protocol you should initialize the AP for your program's exclusive use by issuing the ?APINIT system call before performing any AP operations.

If another process has initialized the AP when you issue an ?APINIT call, AOS will not allow your program to initialize until the other process releases AP resources. In this case, you will receive error condition ERAPB (Array Processor Already Busy).

Since AOS does not pend ?APINIT, you must re-issue the call until the AP is free and AOS allows your process to initialize and "own" AP resources. Keep in mind that even after you initialize the AP, AOS cannot prevent other programs from issuing AP instructions. It can, however, restrict the mapping of AP memory (?APMAP) to the AP "owner".

On the APS/AOS system you cannot map non-AP (main) memory into your logical address space using APS routines. When defining your mapping window, be sure that the area of your logical address space lies in the shared portion of memory. If not, the ?APMAP call will generate errors. After you map AP memory into your logical address space, you lose the original contents of the mapping window.

When you issue an ?APINIT call, your process "owns" AP resources until you issue an ?APREL call. Thus, you should issue an ?APREL call as soon as your program is finished with the AP so that other processes may perform AP operations. You can initialize and release AP resources any number of times in your program.

If your process terminates or chains before issuing an ?APREL call, AOS automatically frees AP resources for use by other processes.

After you release the AP, the area of your mapping window that contained AP memory is undefined. That is, after an ?APREL call, the portion of your logical address space that held AP memory is invalid, and you cannot address it.

System Call Summary

The following list summarizes the system calls used in array processing:

?APINIT	Initialize the AP for use.
?APMAP	Map AP pages into user's shared area.
?APOK	Set or retrieve status of array processor WCS (microcode).
?APREL	Release the AP.

?APINIT

Initialize the array processor for use.

?APINIT
exception return
normal return

Input/Output

Input:

none.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERAPN	AP does not exist.
ERAPS	AP already busy.

Description

This call ensures that the calling process is made the sole "owner" of the AP. The process will remain the owner until it issues the ?APREL call, or terminates, or chains.

Note that the ?APINIT protocol is voluntary. Because of the nature of the AP/CPU interface, AOS cannot prevent other programs from issuing AP instructions. If your program issues the ?APINIT call and "owns" the AP, other programs will receive an error return when they issue an ?APINIT call.

?APMAP

Map array processor pages into the user's shared area.

?APMAP
exception return
normal return

Input/Output

Input:

AC0	number of AP pages to map (0 to 4).
AC1	starting logical address of destination to receive first AP page. This address must be page aligned and in the shared area.
AC2	number of AP page to be mapped into address specified in AC1 (0 to 3).

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERAPN	AP does not exist.
ERAPL	AP WCS is not loaded.
ERAPB	AP already busy.
ERPRE	Invalid system call parameter.
ERSAL	Shared I/O request not Map slot aligned.
ERNSA	Shared I/O request not to shared area.
ERAPR	Attempt to release non-AP page.

Description

This call maps contiguous pages of AP memory into contiguous slots of the user's shared partition. It starts by mapping the first AP page (specified in AC2) to the starting logical address in the user's shared area (specified in AC1). The calling process must already have a shared area, and should have issued the ?APINIT call.

AC0 specifies how many AP pages you want to map with this call. Numbers 1 through 4 are accepted inputs (specifying zero will result in no pages being mapped).

AC1 gives the starting logical address (within the user's shared area) that will receive the first AP page to be mapped. This address must be page-aligned, and must not fall outside the shared area. Since you can map more than one page at a time, care must be taken to ensure that the number of pages mapped does not increment the address outside the shared area.

AC2 specifies which AP page will be mapped into the starting address (given in AC1). The AP pages will be numbered logically 0 to 3. An error will result if a value outside this range is given.

?APOK

Set or retrieve status of array processor WCS (microcode).

?APOK
exception return
normal return

Input/Output

Input:

AC0 one of the following:

- zero if informing system that AP WCS is loaded.
- -1 if status of AP WCS is wanted.
- any other value to inform system that AP WCS is not loaded.

Output:

AC1 if status was requested the PID of the process using the AP is returned in AC1. If the AP is not busy, a zero is returned.

AC2 if status was requested, it is returned in AC2. A zero means AP WCS is loaded, a -1 means it is not.

Exceptional Condition Codes in AC0

ERAPN AP does not exist.

ERAPL AP WCS is not loaded.

ERPRV Caller not privileged for this action.

Description

The utility that loads the WCS for the AP issues this call. Until this call has been made, a program issuing the ?APMAP call will receive error condition ERAPL. A process making this call must have I/O privileges to use the "set" option. If at any time the privileged user wants to inform the system that the AP is not ready, a value other than 0 or -1 should be passed in AC0. If the caller only wants AP status, a -1 should be passed in AC0. The system returns status in AC1; the caller needs no special privileges to obtain status.

?APREL

Release the array processor.

?APREL
exception return
normal return

Input/Output

Input:

none.

Output:

AC0	unchanged.
AC1	unchanged.
AC2	unchanged.

Exceptional Condition Codes in AC0

ERAPN	AP does not exist.
ERAPB	AP already busy.

Description

After issuing this call, the calling process will no longer "own" the AP. All AP pages that are currently mapped into the process's address space will be released. This will leave those slots invalid. On return, the AP will be available to this or any other process.

End of Chapter

Chapter 13
Block I/O

At the device level, all disk, tape and MCA I/O is performed by the system in full physical blocks. Calls to ?READ or ?WRITE records are translated to calls performing appropriate block reads (?RDB) or block writes (?WRB), and the remainder of the block is buffered in the system. In the same fashion, ?OPEN and ?CLOSE calls are translated to block open (?GOPEN) and close (?GCLOSE) calls. The I/O cycle followed by the system is similar to the I/O cycle described earlier in this manual:

Operation	Record I/O	Block I/O
Create the file.	?CREATE	?CREATE
Open the file.	?OPEN	?GOPEN
Read or write.	?READ/ ?WRITE	?RDB/?WRB
Close the file.	?CLOSE	?GCLOSE

If you wish to eliminate some system overhead by doing your own buffering, or wish to access magnetic tape or MCA links in a more general way than is allowed by record I/O, you can issue the block I/O calls directly.

The disk's physical block length is 256 words (512 bytes); tape physical block length is whatever was specified at ?OPEN time, offset ?IMRS. MCA block size is specified for each read/write.

System Call Summary

The following system calls are used for performing block I/O:

?GCLOSE	Close a file previously opened for block I/O.
?GOPEN	Open a file for block I/O.
?GTRUNC	Truncate a disk file.
?PRDB/?PWRB	Perform physical block I/O.
?RDB	Read a block.
?WRB	Write a block.

?GCLOSE

Close a file.

?GCLOSE
exception return
normal return

Input/Output

Input:

AC1 bit zero: 1 if modified status desired on output, otherwise 0.

bit 1-15: channel number.

Output:

AC1 if bit zero was set to one on input, bit zero on output has the following meaning:

0 = file not modified.

1 = file modified.

otherwise, unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This call closes a file that was previously opened by a call to ?GOPEN. If bit zero of AC1 is set, the modified status of the file (0 = not modified, 1 = modified) will be returned in this location.

?GOPEN

Open a file for block I/O.

?GOPEN *[packet address]*
exception return
normal return

Input/Output

Input:

AC0	byte pointer to filename.
AC1	channel number (-1 causes assignment by system).
AC2	address of five-word packet for returned data.

Output:

AC0	bits 0-9: system reserved bits.	
	bits 10-15: universal access codes.	
	?FAOB	owner access.
	?FAWB	write access.
	?FAAB	append access.
	?FARB	read access.
	?FAEB	execute access.
AC1	unchanged.	
AC2	unchanged.	

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

This call opens a file for block I/O and associates a channel number with this file. Any disk file or disk unit, any tape unit or file, or any MCA unit or link can be opened for block I/O. If no channel number (-1) is specified, then the system assigns a free channel and returns the number in the packet of open information.

The standard and IPC packet definitions are shown in Figure 13-1. Note that a five-word area must be reserved to receive the packet information. (A four-word area for IPC packets.) Record format, file type, and file control parameters are the same as those that were specified at ?CREATE time.

For information on how to set up the ?OPCH word for input to ?GOPEN, see Figure 13-2. The ?OPCH word allows different ?GOPEN options.

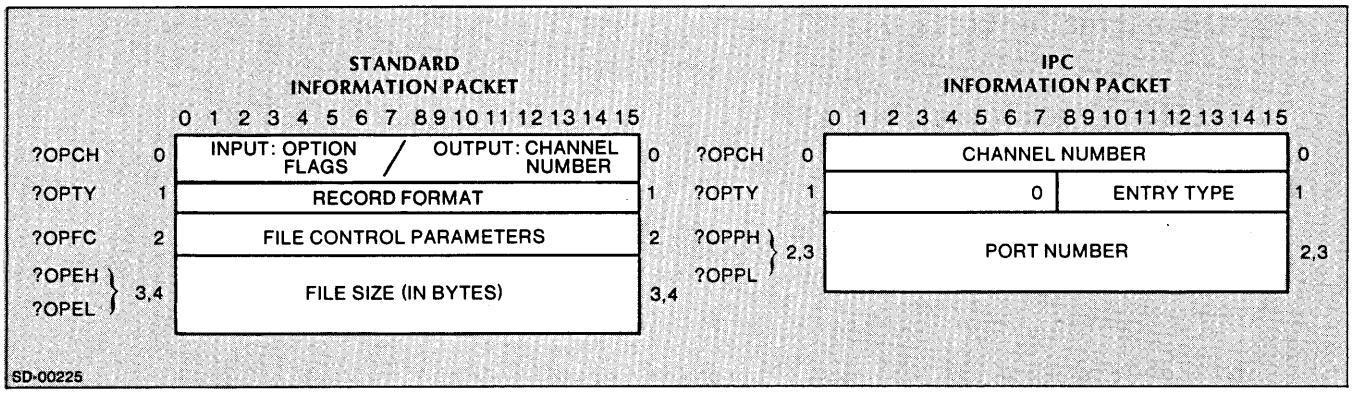


Figure 13-1. Information Packets Returned By ?GOPEN

?RDB/?WRB ignores record formatting and operates in terms of blocks only.

The ?OPFL word of the ?GOPEN packet specifies option flags for the ?GOPEN system call. Figure 13-2 illustrates the format of the ?OPFL word. The exclusive and private open options are explained below. The density and parity options are explained in the section on Special Magnetic Tape Considerations, and the form feed option is explained in the section on Data Channel Line Printers.

	0	1	2-4	5	6	6-15
OPFLI	EX	FF	DEN	PAR	PRI	RESERVED

EX	?OPME	For exclusively opening a file
FF	?OPMD	For inhibiting the initial form feed to a line printer
DEN		Used only for MTB tape drives:
	?OPDL	Low density
	?OPDM	High density
	?OPAM	Automatic density matching
PAR	?OPEP	For even parity selection on 7-track drives
PRI	?OPMP	For private open option on exclusive open

SD-02997

Figure 13-2. ?OPFL Word on ?GOPEN Packet

When you open a file exclusively, it cannot be opened by anyone again until you close it. If you attempt to exclusively open a file that has already been opened, the system will return error ERE01 (File Is Open, Can't Exclusively Open). If you exclusively open a file, and then you or another user tries to open the file, the system will return error ERE02 (File Is Exclusively Opened, Can't Open).

The exclusive open has a private open option which is less restrictive; it specifies that the file cannot be opened by a different process. When a process exclusively opens a file using the private open option, no other process can open the file. The process that initially opened the file, however, may privately open the file as many times as it wants on different channels.

So, a process can privately open a file that it has already privately opened. However, if it tries to privately open a file that it has opened nonprivately, or that another process has opened, the system will return error ERE01. And, if the process tries to nonprivately open a file that has been privately opened by any process (including itself), the system will return error ERE02.

?GTRUNC

Truncate a file.

?GTRUNC *[packet address]*
exception return
normal return

Input/Output

Input:

AC1 channel number.

AC2 address of parameter packet.

Output:

AC1 unchanged.

AC2 unchanged.

Exceptional Condition Codes in AC0

ERMUS Multiple user of file; cannot truncate.

ERSHR Shared file; cannot truncate.

FILE SYSTEM codes

CHANNEL-RELATED codes

Description

This call truncates disk files. It shrinks the size of a disk file by deleting data from the end.

The parameter packet, shown in Figure 13-3, is ?TLTH words long and specifies a number N as the new size of the file (in bytes). The ?GTRUNC call leaves the first N bytes of the file unchanged, and deletes the rest of the file.

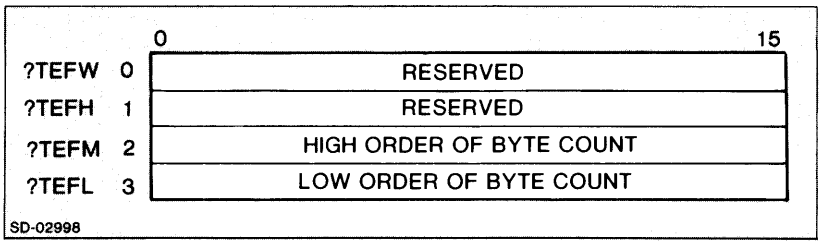


Figure 13-3. ?GTRUNC Parameter Packet

This system call releases any data elements and index blocks that start after the new end of the file. If the last data element contains any disk blocks that start after the new end of the file, the system call zeros them. If the new end of the file falls within a disk block, ?GTRUNC leaves this last disk block unchanged.

Since ?GTRUNC takes a channel number, you must open the file before issuing the call. The file must be opened only by the caller, and that caller can open it only once. The file cannot be shared opened. The caller can open the file exclusively, but this is not required.

?PRDB/?PWRB

Perform physical block I/O.

?PRDB/?PWRB *[packet address]*
exception return
normal return

Input/Output

Input

AC0 unused.

AC1 channel number.

AC2 address of physical block I/O parameter packet.

Output:

AC0 unchanged.

AC1 byte count read or written.

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

?PRDB and ?PWRB read and write blocks of data on disk and tape. Disk blocks are specified by block number.

The block I/O parameter packet is shown in Figure 13-4 and defined in Table 13-1. Table 13-2 illustrates physical block I/O controller status words.

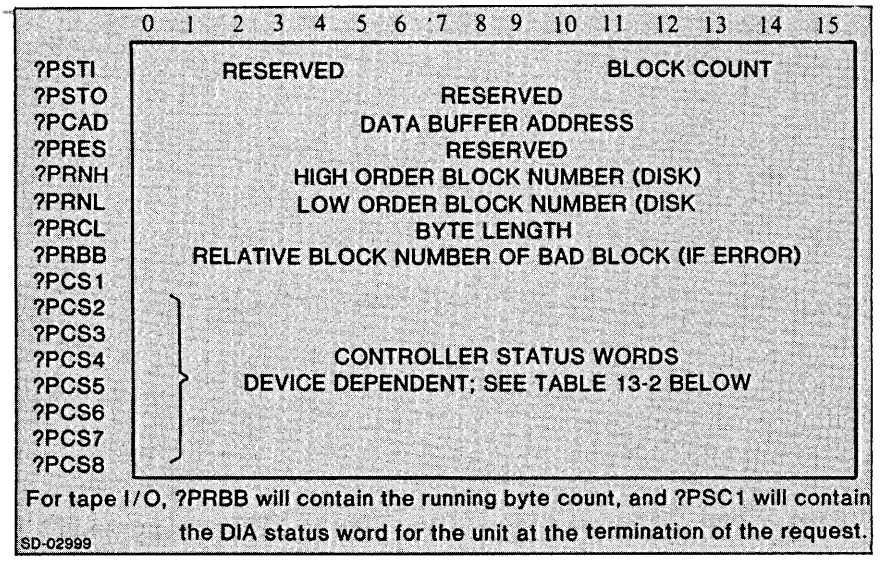


Figure 13-4. Physical Block I/O Parameter Packet

Table 13-1. Physical Block I/O Parameter Packet

Offset	Contents
?PSTI	Right byte: block count. Bits 0-7 = reserved.
?PSTO	Reserved.
?PCAD	Address of data buffer in memory.
?RPES	Reserved.
?PRNH	High-order portion of disk file block number.
?PRNL	Low-order portion of disk file block number.
?PRCL	Number of bytes in last block transferred.
?PRBB	Relative block number of bad block (if error). Transfer byte count if tape.
?PCS1 - ?PCS8	Controller status words (device dependent - see table below).

Table 13-2. Physical Block I/O Controller Status Words

Controller Disk Type	Status Word	Use
6070 (20MB)	?PCS1	DIA (transfer) status
6045/6050/6051 (10MB) and 6030 (diskette)	?PCS1	DIA (transfer) status
4231A (92MB)	?PCS1	DIA (transfer) status
6060/6061/6067/6122 (50MB, 96MB, 190MB, 277MB)	?PCS1 ?PCS2 ?PCS3 ?PCS4	DIA (transfer) status DIB (drive) status ECC word one ECC word two
6063/6065/ (fixed head)	?PCS1 ?PCS2 ?PCS3 ?PCS4	DIC (transfer) status <unused> ECC word one ECC word two
6097/6100/6103 (12.5MB, 25MB)	?PCS1 ?PCS1	DIA (transfer) status DIA status

RDB/?WRB

Perform block I/O.

?RDB/?WRB *[packet address]*
exception return
normal return

Input/Output

Input:

AC1 channel number.

AC2 address of block I/O parameter packet.

Output:

AC1 byte count read or written, unless ?PRNL in parameter packet is -1 (see the section "Special Magnetic Tape Considerations," following).

AC2 unchanged.

Exceptional Condition Codes in AC0

FILE SYSTEM codes

Description

?RDB and ?WRB read and write blocks of data on magnetic tape, disk, or MCA links. Disk blocks are specified by block number, MCA data is specified by link number. Any block within a tape volume can be accessed. Note that you cannot use ?RDB to read blocks into the write-protected area of your address space. You can, however, use ?WRB to write from the write-protected area.

The block I/O parameter packet is shown in Figure 13-5 and defined as outlined in Table 13-3.

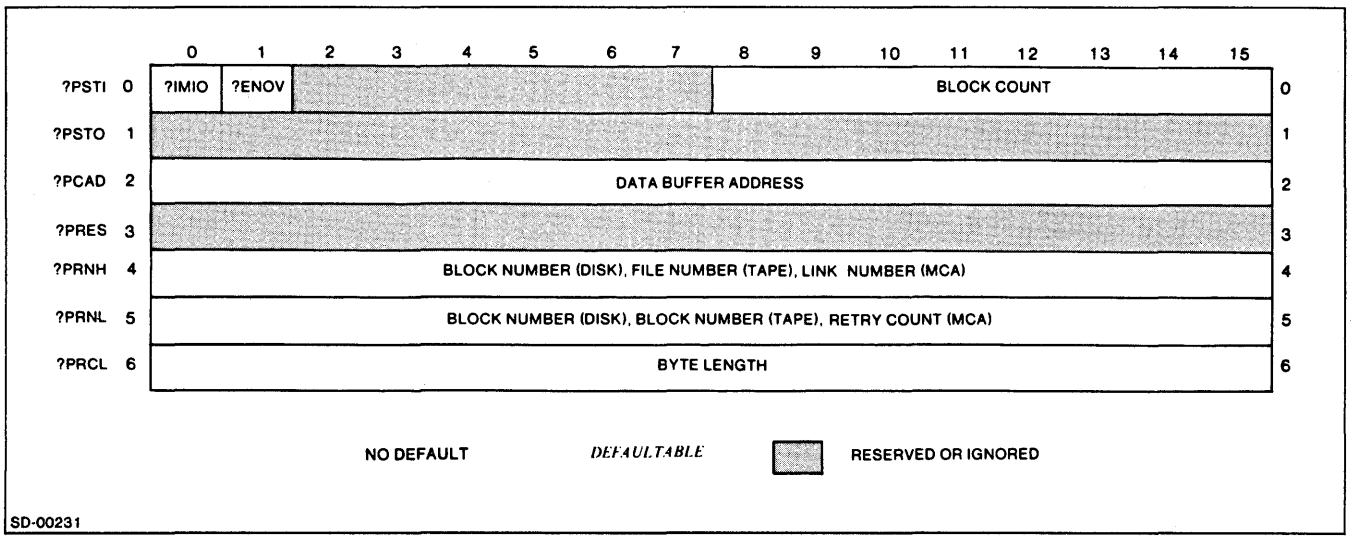


Figure 13-5. Block I/O Parameter Packet

Table 13-3. Block I/O Parameter Packet

Offset	Contents
?PSTI	Right byte: block count bit 0 = transmission mode (MCAs; setting this bit to zero indicates protocol I/O; setting it to one indicates direct I/O). bit 1 = end-of-tape override bits 2-7 = reserved
?PSTO	reserved
?PCAD	Address of data buffer in memory.
?PRES	reserved
?PRNH	High-order portion of disk file block number; file number for tape; link number for MCA. For disk files only: the right byte contains the block number; the left byte is reserved and must be zero.
?PRNL	Low-order portion of disk file block number; block number for tape or -1; retry count for MCA.
?PRCL	Number of bytes in last disk or tape block transferred; number of bytes in last MCA transmission.

Special Magnetic Tape Considerations

Input for ?GOPEN

When you are performing block I/O on magnetic tapes, you have the option of specifying the file number at ?GOPEN time (e.g., @MTB0:3) or in offset ?PRNH of the ?RDB/?WRB packet. If you choose to specify the file number at ?GOPEN time, the system ignores the contents of ?PRNH (see ?RDB/?WRB).

There are two additonal options: you can select even parity on seven-track drives, and you can select density mode on MTB drives. For seven-track units, you can select even parity by using the ?OPEP mask in the ?OPFL word. AOS does odd parity I/O by default.

MTB tape units have four software-selectable density modes: DEFAULT, ?OPDL, ?OPDM, and ?OPAM. Select one by putting the appropriate mode into the ?OPFL/?OPCH word of the ?GOPEN packet. The system manager sets the default as either NO CHANGE, ?OPDL, ?OPDM, or ?OPAM. The mode meanings are as follows:

NO CHANGE	I/O will be done at current density.
?OPDL	I/O will be done at 800 BPI.
?OPDM	I/O will be done 1600 BPI.
?OPAM	I/O will be done at the density of the tape unless the first I/O is a write to file 0, block 0. In this case, I/O will be done at current density.

All tape files are exclusively opened. Figure 13-6 shows an example of the ?GOPEN call for magnetic tape.

```
Input: AC0      Byte Pointer to @MTA0.          ;File number not specified.
       AC1      -1                              ;File system assigns channel.
       AC2      Address of ?GOPEN packet.
       GOPKT:   0                               ;?OPFL word. Default all options.
                0
                0
                0
                0

Output: (If no error conditions)
        AC0   Bits 10-15 are universal access codes.
        Other ACs are unchanged.

        GOPKT:  73                              ;?OPCH - Typical channel number.
                ?FMTU                           ;?OPTY - File type is tape unit. No format.
                11000                           ;?OPEC - Device code 22 and unit 0 (22_0).
                -1                              ;?OPER - Logical end of tape file.
                -1                              ;?OPEL - Logical end of tape block.
                                                ;LEOT is from the last close of the unit.
SD-03000
```

Figure 13-6. A ?GOPEN Example for Magnetic Tape

Input for ?RDB/?WRB

An I/O transfer requires a block count, a buffer address, a tape file position, and the tape block size. Two I/O transfer options are as follows:

- overriding LEOT (logical end of tape)--done by using the ?ENOV mask in the ?PSTI word
- appending an LEOT mark--done by using the ?SAFM mask in the ?PSTI word

If the block count field is nonzero, the system will transfer the indicated number of blocks beginning at the specified tape position. If the block count field is zero, the tape will only be positioned; no I/O will take place. If the block count field is zero and the ?SAFM bit is on, the system will write a LEOT.

Tape file position is represented by a file number and a block number. The system ignores the file number (?PRNH in the block I/O packet) if you provided it at ?GOPEN call time.

The block number has two possible cases. If the block number is -1 AOS will look at the block count field, and act according to the block count field protocol described previously. Or, if the block number is zero, the system will return in AC1 the number of blocks in the file. If the number of blocks is not zero, then I/O will occur at the end of the file. For a ?WRB call, this effects an appending operation. If the block number is not -1, then it is a real block number.

AOS imposes some limits on the block size of a tape transfer. First, AOS rounds up an odd byte block size to the nearest even size. Second, it establishes the following values as illegal:

- transfers of 0-3 bytes
- transfers larger than 8K bytes on MTA units
- transfers larger than the maximum transfer size on MTB and and MTC units. For these units, the system manager specifies maximum transfer size

For a ?WRB call, ?PRCL contains the block size to write with. For a ?RDB call, ?PRCL contains the block size, reserved in the buffer (?PCAD), for each block to be transferred.

LEOT, a tape file position beyond which there is no good information, is used to determine a legal file position. You can override LEOT by setting bit ?ENOV in the ?PSTI word. AOS implements LEOT as two consecutive EOF marks or, equivalently, as a zero length file. On tapes where AOS may encounter or write to zero length files, you should set the override LEOT bit (?ENOV) in the ?PSTI word.

You can use two methods to write a LEOT. The default method involves closing the file after writing to it. This causes the system to write a LEOT after the last block that was written. To use the other method set the ?SAFE bit in the ?PSTI word. In this case, the system writes a LEOT after the ?WRB completes. The first method is faster. The second provides more tape security since the tape is always terminated by a LEOT.

If AOS encounters an EOF while reading, the byte count it returns indicates the amount of data received. The system places the data in the buffer ?PCAD, and takes the exception return with EREOF. The value returned in AC1 is always a true byte count; no rounding occurs. On a ?RDB, the system always truncates blocks exceeding the maximum length to the value specified in ?PRCL. During a multiblock read, if a block is read whose length is less that specified in ?PRCL, the system will skip forward to the maximum length in the buffer before starting to write the next block (see Figure 13-7).

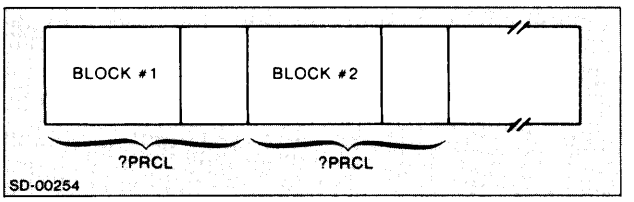


Figure 13-7. Buffer Alignment of Read Blocks

Output for ?RDB/?WRB

The system returns the number of bytes transferred in AC1 for both the exception and normal returns. If the request block (?PRNL) is -1 and the block count (?PSTI, right byte) is zero, then the system will return the number of blocks in the file instead.

Figure 13-8 shows an example of the ?WRB call for magnetic tape.

```
Input: AC1      Channel number from ?OPCH word of ?GOPEN packet.

       AC2      Address of block I/O packet.

       WRPKT:   4          ;?PSTI   Default options, block count = 4.
                0          ;?PSTO   Reserved.
                BUFR       ;?PCAD   Address of buffer.
                0          ;?PRES   Reserved.
                2          ;?PRNH   File number since filename did not specify one.

                0          ;?PRNL   Block number.
                2048.      ;?PRCL   Write blocks of 2048. bytes each.

        BUFR:   .BLK    2048. * 4       ;Buffer can hold 4 2048. byte blocks.

Output:  AC1     8192      ;Number of bytes transferred.

         Other ACs unchanged.

         Packet is unaltered.
SD-03001
```

Figure 13-8. A ?WRB Example for Magnetic Tape

Special MCA Considerations

* Unlike the I/O cycle described for MCAs in Chapter 6, you can use the ?GOPEN call to open the MCA either as a unit or for a specific link. To open the device as a unit, use the ?GOPEN call with one of the following unit filenames: @MCAT, @MCAR, @MCAT1, or @MCAR1. If you open the MCA as a unit, then you must specify a link in offset ?PRNH during the read or write block operation.

To open an MCA for a specific link, use ?GOPEN and one of the following pathnames: @MCAT:n, @MCAR:n, @MCAT1:n, or @MCAR1:n, where n refers to a link in the range 0-15. Note that MCAT refers to an MCA used for writing (i.e., transmitting). MCAR refers to an MCA used for reading (i.e., receiving) purposes. If you open MCA on a specific link (e.g., @MCAT:3), the system will not change the link number, and will ignore offset ?PRNH.

The link number can be changed dynamically on each ?RDB/?WRB call from that which was specified on ?GOPEN. A link of 0 indicates a request to read from any link unit. Link number 0 applies to MCARs only. Read requests from link 0 are satisfied after all requests to read from specific links. Zero is not a valid link number for write block (MCAT) operations.

If the MCA transmitter is opened specifically for a unique link (i.e., the link is specified on the open call, not in a read block or write block call), an EOF with a word count of zero is transmitted to the MCA receiver, with a two millisecond time-out wait, when the file is closed.

NOTE: You should avoid this format if the receiving process is not prepared for this protocol.

The retry count (?PRNL) is in the range 0 through 255 inclusive; it applies only to ?WRB operations from an MCAT. Each count represent twenty attempts to establish a communications path to a target MCAR. A retry count of 0 specifies the maximum retry period. *

You can select direct MCA I/O by selecting ?IMIO in offset ?PSTI. Direct I/O circumvents the protocol, resulting in fewer interrupts. If you select this mode, however, you must perform your own data and line validation. Direct I/O on MCAs should be used only for device boots and in very controlled circumstances.

Also, no direct transmission requests will be honored unless MCAT's output queue is empty. If you attempt to transmit when the queue is not empty, you will receive error ERPRO. When your write request has been queued, subsequent direct or protocol I/O will not be queued to that MCAT link until the current request is complete. If you attempt to perform reads or writes on an MCA link while direct I/O is in progress, you will receive error ERDIO.

Data Channel Line Printers

Data Channel line printer processes in your system must be resident. However, if you use output request spooling on your system (with or without the EXEC capability), processes that send output to a data channel line printer need not be resident. If your system has a data channel line printer controller with the DMA VFU option, you have a number of unique features not available with programmed I/O line printers.

A memory within the printer eliminates mechanical paper tape printer control. For a complete description of these extensive horizontal and vertical formatting capabilities, see the Peripherals manual for the ECLIPSE® computers.

To change the DMA VFU option memory, you must specify ?ENOV in offset ?PSTI of the ?WRB parameter packet. If you do not select ?ENOV, the line printer will ignore all VFU definition commands.

End of Chapter

Appendix A
Exceptional Condition Codes

This appendix categorizes and describes the exceptional condition codes (from ERICM through ERVSY) you receive in AC0 when any system call takes an exception return. All system exception codes are in group zero; error code structures and groups are described in Chapter 9.

Certain exception codes (such as ERMPR, system call parameter address error) are listed in several categories. Codes are listed alphabetically in each category. The categories themselves are found in the following order.

- Channel-Related
- File System
- Initialization and Release
- IPC
- Magnetic Tape
- Memory
- Miscellaneous
- Process
- System Call
- Task
- User Device
- Array Processor
- Connection Management
- Synchronous Line

Exceptional conditions are defined parametrically in the user parameter file, PARU.SR. The system provides an error message file named ERMES which contains a textual description of each error code; you can read the description associated with an error code by issuing system call ?ERMSG or CLI command MES.

Channel-Related Codes

Mnemonic	Description
ERACU	Attempt to close an unopened channel.
ERCIU	Channel in use. Attempt to close a channel with shared pages in use. You must first release the shared pages; only then can you close the channel. In ?GNFN, another system call is outstanding on this channel. You can receive this error from ?GNFN only in a multitask program.
ERFNO	The channel you used in this call is not currently open.
ERICN	Attempt to use a channel outside the legal range, 0 through 100 octal. (Note: The system assigns 100 octal to a file if you designate the file as @NULL.)
ERNDR	The channel specified to ?GNFN is not opened on a directory.
ERNMC	No free channels. A process cannot have more than 64 channels open at one time.

File System Codes

Mnemonic	Description
ERACL	You specified an illegal ACL in an ?SACL call.
ERACN	File entry has no Access Control List (returned by ?GACL).
ERAWD	You attempted to ?CREATE a file without having write-access to the directory which was to contain the file.
ERBOF	You tried to set the file pointer before the beginning of the file (?IRNH/?IRNL of I/O packet set to too large a negative value).
ERBSZ	Attempted read or write of a block with an odd number of characters.
ERCDN	Controller does not support this density.
ERCPD	You issued a call which requested more disk space than is available in the control point directory.
ERDAD	Directory access is denied to you.
ERDCT	Modem was disconnected before the completion of a ?READ or ?WRITE.
ERDDE	Directory does not exist.
ERDTO	Device timed out.
EREAD	Execute access is denied to you.
EREOF	End of file. In ?GNFN, this means that there are no more directory entries.
EREO1	You attempted to open a file exclusively (?IEXO in the ?OPEN packet), yet the file was already open.
EREO2	You attempted to open a file which was already opened exclusively (?IEXO in the ?OPEN packet).

(continues)

File System Codes

Mnemonic	Description
ERFAD	File access is denied to you.
ERFDE	Filename does not exist. A filename in a pathname was not found, or the user has no access to that entry. Alternatively, a process's console port number was requested, and the process has no console.
ERFIL	File read error.
ERFTL	You used a filename which was too long; 31 characters is the maximum length. In ?GNFN, your template's length exceeded the 63-character maximum.
ERFTM	File/tape density mismatch.
ERICB	Not enough contiguous blocks to allocate a disk file element.
ERIFC	Illegal filename character. Legal filename characters are limited to the following: A through Z, a through z, 0 through 9, period (.), dollar sign ($), question mark (?), and underscore (SHIFT O). Illegal template characters in ?GNFN.
ERIFT	Illegal file type. You tried to create a file with an unknown system file type.
ERILB	Illegal library number. Perhaps you deleted or altered the symbol table file (.ST) associated with your program, or you did not make available one or more shared libraries which were required by your program. Ensure that required shared libraries are either present in your working directory or are listed in your process's search list.
ERILN	Illegal MCA link number. Link number is outside the range 1-15 for a transmitter, or 0-15 for a receiver.
ERILO	You attempted to ?SOPEN a file whose element size is not a multiple of four.
ERIOT	Illegal type of I/O, e.g., ?RDB/?WRB to a character device.
ERIRB	You supplied a user buffer as a call parameter, and the buffer was too small.
ERIRL	Illegal record length in variable record header. A nondigit was found in the 4-byte length field of a variable record header.
ERIRV	You specified an illegal retry value in ?PRNL to a ?WRB call for MCA I/O.
ERITC	I/O has been terminated by a ?CLOSE call.
ERITD	Indecipherable tape density.
ERITF	Incompatible tab format. The data channel line printer has received an unexpected tab character. The I/O request is aborted.
ERLNK	Attempt to create a link whose length exceeds 256 bytes.
ERLTL	Line too long. On a data sensitive read or write, the maximum line length was exceeded before a terminator was detected.
ERLVL	You attempted to create a directory at a tree depth which exceeds the system maximum.
ERMIL	Attempt to exceed the maximum index level, or file exceeds its maximum permissible size.
ERMPR	System call parameter address error. (Buffer designated to receive pathname information may be too small; see ?CGNAM, Chapter 6.)

(continued)

File System Codes

Mnemonic	Description
ERNAD	Used a nondirectory argument in a pathname. All filenames in a pathname, except the last filename, must be filenames of directories. For magnetic tape or MCA units, the last filename may have the form *unit:n*, where *unit* is a magnetic tape or MCA unit, and *n* is a decimal number. For labeled mag tape, the last two entries must be *:volid:filid*.
ERNAE	Filename already exists. You attempted to create a directory entry with a name that is already in use.
ERNRD	Insufficient room in directory. Directories can be a maximum of 2^{11} blocks long.
ERNRR	MCA transmitter time-out because no ready receiver was found.
ERNSA	You requested shared I/O into a nonshared area. Alternatively, memory addresses and/or I/O size are not entirely within the current shared area.
EROPR	You attempted to open a channel that is already open.
ERPAR	Parity error.
ERPET	You tried to read beyond a double tape mark, the logical end of tape.
ERPRM	Permanent file delete error. The file is permanent and cannot be deleted.
ERPUF	Physical unit failure.
ERPWL	Physical write lock. Write enable ring is missing from a magnetic tape reel and writing was attempted.
ERRAD	Read access is denied to you.
ERRFM	Illegal or unspecified record format.
ERSAL	You issued a shared I/O request, and either the memory address to be used in the transfer does not begin on a 2K-byte boundary or the number of blocks in the transfer is not a multiple of four.
ERSIM	Simultaneous requests have been made on the same channel; another task has an active request on this channel.
ERSPC	Magnetic tape space is exhausted (an end-of-tape mark was detected on a write to magnetic tape).
ERSRE	Search list resolution error. Alternatively, some other file system error was received when the system attempted to resolve a directory name in the pathname.
ERSRR	MCA transmission was not completely received because the receiver requested less than the full transmission.
ERSTR	MCA transmission was shorter than requested by the receiver.
ERTIN	MCA transmitter failure detected upon an attempted read.
ERTMR	Too many tape retries.
ERUOL	Physical unit is off-line.
ERVIU	LD is in use; the attempted release cannot be performed.
ERWAD	Write access is denied to you.
ER9TP	Cannot select even parity on nine-track drives.

(concluded)

Initialization and Release Codes

Mnemonic	Description
ERARC	Attempt to release the console device.
ERARG	Internal system I/O error.
ERARU	Attempt to desassign an unassigned device.
ERDAI	Device is already in use. You attempted to assign or open a device that was assigned to another process.
ERDVC	Illegal device type. AOSGEN information is inconsistent. Either device information was entered incorrectly, or it was modified since SYSGEN.
ERFIX	LD needs to be fixed (use the FIXUP utility).
ERIBS	The device you attempted to initialize is already initialized.
ERIDD	The system found inconsistent data in a system database called a Disk Information Block (DIB); your LD cannot be initialized.
ERIDT	You attempted to initialize a spooled device (e.g., @LPT).
ERIDU	The set of disks you tried to initialize do not form the complete LD which was specified to the DFMTR utility.
ERILD	The set of disks you tried to initialize belong to two or more different LDs.
ERMIS	The disk revision number and file system revision number do not match.
ERIPD	Initialization privilege denied. You tried to initialize an LD, but you do not have owner access to its root directory and you were not in superuser mode.
ERVIU	You tried to release an LD that is in use.

IPC Codes

Mnemonic	Description
ERFDE	Filename specified in the ?ILKUP IPC code does not exist or is not an IPC type entry.
ERIDP	Illegal destination port number.
ERIOP	Illegal origin port number.
ERIVP	Invalid port number. Either the number is outside the legal range, or it is not assigned.
ERMPR	System call address error (see also System Call Codes).
ERNEF	IPC message was longer than the buffer that was to receive it.
ERNMS	No matching send request in IPC spool file, and the receiver did not specify ?IFBNK, i.e., that it should be suspended if no message was ready.
ERNOR	No outstanding receive request, and the sender does not want the message to be spooled.

Magnetic Tape Error Handling

Mnenomic	Description
ERBSZ	Illegal block size for device. This occurs with a request size that is too large or a 0-3 byte block size with a nonzero block count.
ERCON	Tape controller does not support this density mode. This is caused by a ?GOPEN call to a non-MTB unit with a nondefault density.
ERDTO	Device timeout. The request has made no progress in a reasonable time. This error suggests a hardware problem.
EREO1	File is open. It cannot be exclusively opened. Someone is already using the tape unit. (All tape units are exclusively opened.)
EREOF	End of file. A block requested is beyond the current file's length.
ERFTM	File/tape density mismatch. This error occurs when the specified density mode in the ?GOPEN packet does not match the density mode of the tape. This is a partial guard against doing I/O when the wrong tape has been mounted. This error cannot occur with automatic density matching mode.
ERITD	Indecipherable tape density. AOS has not been able to discover the tape's density. This is either a tape media or a tape unit problem.
ERPET	Magnetic tape request is past logical end of tape. A tape position requested at some time during the I/O request is in that part of the tape which has no information. If LEOT is overridden this error cannot occur.
ERPUF	Physical unit failure. (Possible error conditions are data late, illegal parity, EOT, EOF, BOT, bad tape, and odd character.)
ERPWL	Physical write lock. There is no write ring in the tape, and a ?WRB has been issued.
ERSCA	System call abort. The I/O request was terminated by process termination or a power failure.
ERSPC	File space exhausted. The physical end of tape was encountered. This terminates a request after the block that contained the EOT mark.
ERTMR	Too many retries. AOS has not been able to correct a parity or bad tape error after several attempts.
ERUOL	Unit off line. The tape unit is off line; the controller cannot access it.

Memory Codes

Mnemonic	Description
ERADR	Illegal starting address. The program file's starting address does not lie within its address space.
ERMEM	Insufficient amount of memory available. Possibly you have attempted to exceed the maximum amount of core memory which was allotted to your process when it was created.
ERMRL	Memory release error.
ERNSW	Out of swap file space. Contiguous disk space for this image cannot be allocated.
EROVN	Illegal overlay number; the overlay area is not currently occupied by the specified overlay; or, the overlay number could not be found in the overlay directory.
ERRDL	Resource deadlock (see the general procedure call ?KCALL, described in Chapter 3).
ERROO	Attempt to release an overlay (by ?OVREL, ?OVEX, or ?OVKIL) that was not in use.
ERSEN	The external reference you specified in a shared routine call did not exist in the system tables found in user space (you may inadvertently have overwritten the area below ?USTART).
ERSHP	Error upon setting a shared partition. The pages you specified are already in an unshared area, or are otherwise illogical.
ERSRL	Attempt to release a shared page via ?RPAGE when that page is not in use.

Miscellaneous Codes

Mnemonic	Description
ERASS	You tried to assign a device which was already assigned.
ERBSZ	Illegal block size for device.
ERCGF	Indeterminate internal system error upon either an attempted ?PROC or ?OPEN of a generic filename. If upon a ?PROC, the new process will not be created. If upon an ?OPEN, a fatal system error is indicated. Perform a memory dump for system analysis.
ERDID	You tried to delete a directory containing entries of one or more inferior directories.
ERDIF	Illegal dump format, a fatal system error, was detected during the initial system load. Try cleaning the tape heads and repeat the load procedure. If this fails to help, you probably have a bad system tape.
ERDIO	Attempt to issue MCA direct I/O request while device queue contains an entry.
ERDIU	You tried to delete the working directory (=).
ERESO	You attempted to issue ?EXEC from a process which was not created by EXEC.
ERFUR	Fatal user runtime error. This error indicates an internal system error detected by a routine from URT.LB. Contact your local Data General representative.
ERGES	Internal system error; process is terminated.
ERHIS	Illogical histogram packet, or attempt to start a second histogram when a first already exists.
ERIBM	Inconsistent data in block allocation map; a fatal system error. This error can occur when disk blocks are allocated or deallocated.
ERICL	Message targeted by ?GTMES has an illegal format.

(continues)

Miscellaneous Codes

Mnemonic	Description
ERIGM	You used an illegal parameter in a ?GTMES call.
ERISV	You attempted to pass a switch whose value exceeds 2^{16} -1.
ERLRF	Resource load or release failure.
ERMPR	System call parameter address error.
ERMRD	Attempt to issue ?SEND to a console which has ?CNRM set in its characteristics.
ERNAC	Attempt to issue ?SEND to a nonconsole device.
ERNSD	In response to "Specify Master Logical Disk" upon a program load, you specified a logical disk which did not contain the system disk. Specify the proper logical disk.
ERPDF	System detected an error in one or more words in the User Status Table (see Appendix C).
ERPRO	Attempt to issue an MCA I/O request while direct MCA I/O is in progress.
ERPRV	You are not privileged to perform this action or issue this call.
ERRBO	The operator refused your ?EXEC request.
ERRMP	Internal system error; process is terminated.
ERRST	This error code is used only by the system and you should never receive it. If you do, contact your local Data General representative.
ERSOC	Internal system error; process is terminated.
ERSTO	System stack overflow (an internal system error).
ERTIM	Attempt to set the system clock to an illegal time, or the system calendar to an illegal date.
ERTXT	The actual error message length exceeds the one which was requested. This code is returned only by ?ERMSG.
ERVSY	There are several operations which you cannot perform when using the universal system (supplied on the system tape or diskette). These are described in *How to Load and Generate Your AOS System*. You attempted to perform one of these illegal operations.
ERWMT	You requested the dismounting of an already dismounted tape reel or disk.
ERXMT	Signal to address already in use. You attempted to transmit a message to a nonzero mailbox.
ERXMZ	Attempt to issue ?XMIT with an invalid message. Message must be nonzero.
ERXNA	EXEC module is not present in the system, yet you issued ?EXEC.
ERXUF	You requested an unknown function in the ?EXEC parameter packet, offset ?XRFNC.

(concluded)

Process Codes

Mnemonic	Description
ERBLR	Attempt to block a resident process (?BLKPR).
ERBMX	Attempt to create a process with an illegal maximum size. The size of the created process cannot exceed the size of the caller's process.
ERCON	Console device specification error. Either the named device is not a console device, or it is a device which is currently in use by another process.
ERDFN	Upon a ?PROC, the generic DATA file you specified does not exist.
EREXC	A resident process attempted to issue ?PROC and block its son.
ERGFE	You specified one or more generic files circularly. For example, you specified OUTPUT to be equal to LIST, and LIST equal to INPUT. Or, you specified a generic file to be set to itself (e.g., DATA to DATA).
ERIFN	Upon a ?PROC, the generic INPUT file you specified does not exist.
ERIFT	The file to be ?PROC/?CHAINed is not of type ?FPRG.
ERIPR	Illegal ?PROC parameter. Packet defaulted the IN, OUT, DATA, or LIST generic filenames, but specified that the father was not to block its son.
EROFN	Upon a ?PROC, the generic OUTPUT file you specified does not exist.
ERLFN	Upon a ?PROC, the generic LIST file you specified does not exist.
ERMPR	System call parameter address error.
ERPDF	The system detected an error in a program's User Status Table (see Appendix C for a description of this table).
ERPNM	Illegal process name (e.g., too long or uses illegal characters).
ERPNU	A process name specified in a ?PROC call is in use by another process.
ERPRH	Attempt to access a process which is not in the tree.
ERPRN	Attempt to create a process when the maximum, 64, already exist.
ERPRP	Illegal process priority. You attempted to specify a process priority greater than your own, and you were not privileged to do so.
ERPTY	Illegal process type. You tried to change a target process's type to one which is different from your own (via ?CTYPE) or you tried to create a process (?PROC) of a type different from your own, when you lack privilege ?PVTY. The error can also mean that you tried to designate a process as both pre-emptible (?PFRP) and resident (?PFRS), or you attempted an operation for which residency is required, and you are not resident. For example, you must be resident to issue I/O to MCAR, MCAT, LPB, or LPD, or to issue the ?IHIST and ?IXHIST system calls.
ERSNM	You tried to create more processes than you are entitled to create (see ?PPCR in the ?PROC parameter packet).
ERUNM	Attempt to assign a username, other than that of the calling process, and caller lacks privilege ?PVUI.

System Call Codes

Mnemonic	Description
ERGSG	You issued an unknown (illegal) system call.
ERICM	Illegal system command; attempt to execute a system call which is not currently defined. This code is never returned by a system call defined in SYSID.SR, the parameter file defining the names of all system calls.
ERMPR	System call parameter address error. An address used as a call parameter is not within user address space, or it is the address of a write-protected area which would have been written into.
ERPRE	Invalid or illogical system call parameter.
ERRST	This error code is used only by the system, and you should never receive it. If you do, contact your local Data General representative.

Task Codes

Mnemonic	Description
ERABT	Task not found for abort. You issued an ?IDKIL system call for a nonexistent task.
ERLTK	Last task killed. The last (or only) task in a process was killed. The process is aborted.
ERNMW	No message waiting.
ERNOT	Insufficient number of TCBs are available for initializing all tasks specified in the ?TASK packet.
ERQTS	Either you tried to queue a task (?TASK) without having first issued ?IQTSK, or you issued more than one ?IQTSK.
ERSFT	User stack fault. You supplied no stack fault handler, so the default stack fault handler returned this code.
ERSTS	Stack too small. You attempted to initialize a task with a stack less than the minimum length (30 words).
ERTID	Task ID error. You issued a task call with an invalid task ID.
ERTMT	Too many tasks. You attempted to initialize too many tasks. Maximum number of concurrent tasks is 32 decimal.
ERXMT	You attempted to transmit to a mailbox currently in use.
ERXMZ	You attempted to transmit a zero.

User Device Codes

Mnemonic	Description
ERDCH	There is insufficient room in the data channel map.
ERDNM	You passed a device code outside the range 1-76 octal. Device code 77 octal is valid, but is reserved.
ERIBS	This interrupt device code is already used.
ERMIM	An ?IMSG call has already been issued to this DCT, and only one ?IMSG per DCT is allowed.
ERPRV	You lack the privilege required to issue this call (?PVDV).

Array Processor Codes

Mnemonic	Description
ERAPB	AP already busy.
ERAPL	AP WCS is not loaded.
ERAPN	AP does not exist.
ERAPR	Attempt to release non-AP page.
ERNSA	Shared I/O request not to shared area.
ERPRE	Invalid system call parameter.
ERPRV	Caller not privileged for this action.
ERSAL	Shared I/O request not map slot aligned.

Connection Management Codes

Mnemonic	Description
ERCCS	Caller tried to connect to itself.
ERCDE	Connection does not exist.
ERCTF	Connection table is full; i.e., 127 connections have already been established.
ERMPR	System call parameter address error.
ERNAS	Process (caller or recipient of call) is not a server.
ERPNM	Illegal process name; e.g., too long, or uses illegal characters.
ERPRH	Attempt to access a process which is not in the process tree.
ERPRV	Caller lacks the privilege to issue this call (?PVDV).

Synchronous Line Codes

Mnemonic	Description
ERBCT	Send byte count exceeded system buffer capacity.
ERBNK	Bid error (too many NAKs).
ERBOV	Buffer overflow. Incoming data block length exceeded buffer byte count you specified with ?SRCV call.
ERBPE	User buffer byte pointer invalid.
ERBRT	Retry count exceeded.
ERCNV	Conversational reply received as response to send operation.
ERCRC	CRC error occurred.
ERCTN	Contention situation during bidding (secondary station only).
ERDCU	DCU inoperative and cannot be initialized.
ERDIS	Disconnect of switched line.
ERDSL	Line already disabled on ?SDBL call.
ERENQ	ENQ sequence received after time-out.
EREOT	EOT sequence received.
EREPE	Line already enabled on ?SEBL call.
EREPL	End of polling list (multidrop line).
ERETX	ETX code received.
ERFCT	Failure to connect a switched line.
ERIMM	Insufficient memory for poll/select list (multidrop line).
ERINE	Invalid packet address encountered.
ERIRT	Illegal relative terminal number (multidrop line).
ERISE	Input status word error occurred (format).
ERLIN	Illegal line number specified on ?SEBL call.
ERLIS	You attempted to issue a send-initial call while line was in session, i.e., following a previous send-initial call.
ERLNA	I/O request for disabled line.
ERNAK	NAK count exceeded (send failure).
ERNSL	Attempt to issue ?SEBL call to nonsynchronous line.
EROTH	Possible loss of data on HASP line.
ERPLS	Insufficient space for poll lists (multidrop line).
ERRVI	Reverse interrupt sequence received.
ERSCS	Attempt to send-continue without prior send-initial call.
ERSEQ	Out-of-sequence AOSGEN entry encountered during system initialization.
ERTOF	Time-out value exceeded (send a failure).
ERTRF	Transmitter failure error occurred.
ERUNI	Uninterpretable response received.
ERWAB	Wait-a-bit call received (pertains to HASP line only).

End of Appendix

Appendix B
Real-Time Programming Example

This appendix contains an example program running under AOS. This program uses the initial task to initiate two other tasks. Each of these tasks reads its file (File1 or File2, respectively) and writes it to the line printer (see Figure B-1).

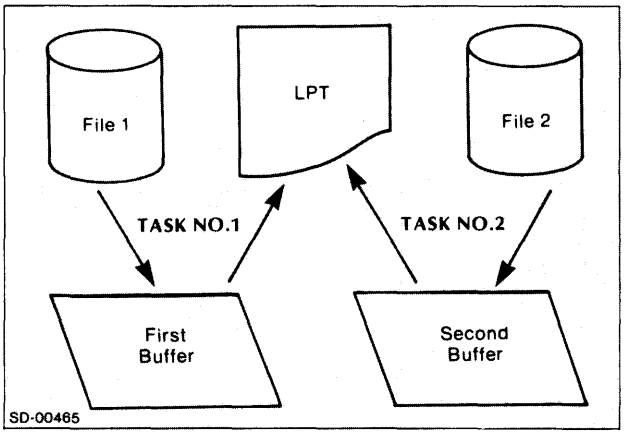


Figure B-1. Main Task Activity

Although it clearly would be more efficient to use one task to perform both operation sequences, we use two tasks here for illustration purposes. The system creates a spool file for each task, and therefore there is no possibility that it will intersperse the data on the printer. For illustration purposes, however, this program treats the write sequence as a critical region. The critical region is locked before writing and is unlocked after writing.

Programming Details

The illustration program, entitled EXAMP, is dissected and shown in the following series of figures.

The first section of the program, the initial task (page 1, lines 1-19), establishes the other two tasks, initializes the critical region, and kills itself. The beginning of the program, START, is .ENTered so that its starting location can be seen in the listing output by Link. Unshared code relocation is specified, since the only part of the program that might be usefully shared (Figure B-2) is very small; sharing this part would require the use of an entire 2048-byte page.

The critical region is managed by ?XMT/?REC calls. The lock (LOK) is initialized so that the first task will find the region unlocked and usable; see page 1, lines 12-14. The two workhorse tasks are established, and the initial task kills itself (line 18). Details of the task specification will be seen in the task parameter packet (page 4, lines 41-60, and page 5, lines 1-6).

The task code proper begins at MAIN and ends with the stack (STACK); see Figure B-3.

```
 0001 EXAMP AOS ASSEMBLER REV 00.00              10:18:56 02/02/77
01
                        .TITLE EXAMP
03                      .ENT START
04
05        000003        .TSK 3
06
07                      .NREL           ;UNSHARED CODE RELOCATION
08        000001        .TXTM 1         ;PACK TEXT STRING BYTES LEFT/RIGHT
09
10              ;PRELIMINARY SET-UP.
11
12 00000'060454 START:  LEF     0,LCK   ;INITIALIZE THE CRITICAL REGION.
13 00001'126000         ADC     1,1
14                      ?XMT
15 00004'000545         JMP     ERTN
16                      ?TASK   TASKPK  ;INITIATE THE TWO WORKHORSE TASKS.
17 00011'000540         JMP     ERTN
18                      ?KILL           ;KILL THE DEFAULT TASK.
19
```

Figure B-2. EXAMP Program Initialization (page 1, lines 01-19)

```
20              ; TASK CODE PROPER
21
22 00014'163710 MAIN:   SAVE    0       ;MAIN TASK ACTIVITY.
23        000000
24 00016'031000         LDA     2,0,2
25                      ?OPEN           ;OPEN FILE1 OR FILE2
26 00021'000530         JMP     ERTN    ;ERROR RETURN
27 00022'031776         LDA     2,?CAC2,3
28 00023'031001         LDA     2,1,2
29                      ?OPEN           ;OPEN LINE PRINTER
30 00026'000523         JMP     ERTN
31 00027'031776 CONT:   LDA     2,?CAC2,3
32 00030'031000         LDA     2,0,2   ;GET READ PKT ADDRESS
33                      ?READ           ;READ RECORD FROM PROPER FILE
34 00033'000420         JMP     EOF     ;KILL TASK AT END OF FILE
35 00034'060420         LEF     0,LCK   ;WAIT UNTIL NO WRITE IS IN PROGRESS
36                      ?REC
37 00037'000512         JMP     ERTN
38                              ;CRITICAL REGION BEGINS HERE.
39
40 00040'031776         LDA     2,?CAC2,3
41 00041'031001         LDA     2,1,2   ;GET WRITE PKT ADDRESS
42                      ?WRITE          ;OUTPUT THE RECORD ON THE LINE PRINTER.
43 00044'000505         JMP     ERTN
44 00045'060407         LEF     0,LCK   ;UNLOCK THE CRITICAL REGION
45 00046'126000         ADC     1,1
46                      ?XMT
47 00051'000500         JMP     ERTN
48 00052'000755         JMP     CONT
49 00053'127710 EOF:    RTN             ;KILL THE  TASK .
50 00054'000000 LCK:    0               ;CRITICAL REGION LOCK
51
52 00055'000074 STACK:  .BLK    2*30.   ;RESERVE TWO  STACKS.
53
```

Figure B-3. Task Code Proper (page1, lines 20-53)

First, a return block is pushed onto the stack (page 1, line 22). The address of a list of two packet addresses, the read and write packets, is passed by the system to each newly initiated task, in AC2. This address is stored in each task's ?OAC2. The task opens File 1 for reading (line 25) and the line printer for writing (line 29). The task then fetches its read packet address again (lines 31 and 32). Having issued the read request, the task then suspends itself until the critical region becomes unlocked, by issuing ?REC.

Control does not return to the task until the mailbox (or lock, LOK) is set to a nonzero value. Since LOK was initialized to -1 earlier, the first task finds the region unlocked. On succeeding passes, the region will be locked while each task uses the line printer. When LOK becomes unlocked, the task becomes readied. As the task becomes readied, the region becomes relocked automatically, since receiving from any mailbox zeros the mailbox's contents. The task fetches its write packet address, and it writes its record to the line printer spool file. After completing the write, the task unlocks the region by transmitting a -1 to LOK (page 1, lines 44-46). Each task continues this cycle until it detects an end-of-file condition. When this happens, control goes to line 34 and then to EOF where the task returns.

Note that the tasks do not close their files. After both tasks reach the respective ends of file and return, there are no active tasks left in the program. Thus, the system terminates the process. This closes the files, and at this time the output files are written on the line printer.

STACK (page 1, line 52) is the start of the two stacks for the two workhorse tasks. Each task receives a stack of 25 words, the minimum possible length.

The next section of code (Figure B-4), page 1 lines 54-60 and page 2, lines 01-12, provides error return processing and miscellany.

```
54                ;ERROR PROCESSING, AND MISCELLANEOUS POINTERS.
55
56 00151'030404 ERTN:  LDA     2,RFEC  ;TERMINATE THE PROCESS, SEND ERROR MSG.
57                                      ;BACK TO CLI.
58                     ?RETURN
59 00154'000775        JMP     ERTN    ;REPORT ?RETURN ERROR IF CAN.
60


 0002 EXAMP
01 00155'150000 RFEC:  ?RFEC+?RFCF+?RFER  ;AC0 CONTAINS ERROR CODE,
02                                        ;CLI IS FATHER, ?RETURN DUE TO ERROR CONDITION
03 00156'000074 IOBF1: .BLK 60.           ;FIRST I/O BUFFER, 120 CHARACTERS LONG.
04 00252'000074 IOBF2: .BLK 60.           ;SECOND I/O BUFFER, 120 CHARACTERS LONG.
05 00346'000350'MESG:  MES1               ;ADDRESS OF 1ST TASK I/O PKT ADDRESSES
06 00347'000352'       MES2               ;ADDRESS OF 2ND TASK I/O PKT ADDRESSES
07                                        ;(SENT IN AC2 TO NEW TASKS).
08 00350'000407'MES1:  READ1              ;FIRST TASK I/O PACKETS
09 00351'000354'       WRIT1
10 00352'000426'MES2:  READ2              ;SECOND TASK I/O PACKETS
11 00353'000370'       WRIT2
12
```

Figure B-4. Error Return Processing, Etc. (page 1, lines 54-60 and page 2, lines 01-12)

Parameter ?RFEC, defined in PARU.SR, indicates to ?RETURN that an error code is being sent. Additionally, CLI is the father, and ?RETURN is due to an error condition. ?RETURN matches the code with its list of error messages in the error file ERMES (each installation has one), terminates the process, and returns to the CLI which displays the message on the console. Hopefully, of course, this sequence (page 1, lines 56-59) will never receive control.

Two I/O buffers, IOBF1 and IOBF2, are reserved as 60-word blocks (page 2, lines 3-4). The operating system always passes a one-word message to each newly initialized task; it gets this message from the task parameter packet, offset ?DAC2. In this program, MESG (line 5) is passed to the first task, and MESG+1 is passed to the second task. These are addresses of lists of two parameter packet addresses, READ1/WRIT1 (page 2, lines 8-9) or READ2/WRIT2 (lines 10-11). Each task selects the appropriate packet address at the proper time by fetching its original AC2 and indexing off this to the proper packet.

The remainder of the program shows all the parameter packets used in the program. The first two of these, the packets used in opening and writing to the line printer, are shown in Figure B-5 (page 2, lines 13-60 and page 3 lines 1-20). Actually the packet ends at line 15; lines 17 through 19 contain the string naming the line printer. Note how this example builds the parameter packets. The .LOC pseudo-op establishes the contents of each packet offset parametrically. This frees you from any concern that actual offsets might be changed, a constant problem if absolute offsets were defined in the program. At the end of each packet, the .LOC pseudo-op updates the counter to ensure that what follows does not get intermixed with the previously-defined packet.

```
13                ;PARAMETER PACKET HEAVEN.
14
15                                          ;LINE PRINTER PACKETS FOR ?OPEN,?WRITE
16
17                A:
18                WRIT1:
19          000354'           .LOC    A+?ICH
20  00354'000000              0                   ;CHANNEL NUMBER ASSIGNED BY SYSTEM
21          000355'           .LOC    A+?ISTI
22  00355'040012              ?OFOT+?RTOS+?ICRF       ;WRITE ONLY, DATA SEN. RECORDS,
23                                                  ;STARTING AT BEGINNING OF BUFFER
24                                                  ;EACH TIME.
25          000356'           .LOC    A+?ISTO
26  00356'000000              0                   ;RESERVED
27          000357'           .LOC    A+?IBAD
28  00357'000334"             IOBF1*2             ;I/O BUFFER POINTER
29          000360'           .LOC    A+?IRES
30  00360'000000              0                   ;RESERVED
31          000361'           .LOC    A+?IRCL
32  00361'000170              120.                ;RECORDS WILL BE 120 CHARS. MAXIMUM
33          000362'           .LOC    A+?IRLR
34  00362'000000              0                   ;BYTES WRITTEN, RETURNED BY SYSTEM.
35          000363'           .LOC    A+?IRNR
36  00363'000000              0                   ;WRITE NEXT RECORD.
37          000364'           .LOC    A+?IRNL
38  00364'000000              0                   ;WRITE NEXT RECORD.
39          000365'           .LOC    A+?IFNP
40  00365'001010"             LFTPR*2 ;BYTEPOINTER TO NAME OF LINE PRINTER.
41          000366'           .LOC    A+?IMRS
42  00366'000000              0                   ;BLOCK SIZE IS IRRELEVANT
43          000367'           .LOC    A+?IDEL
44  00367'177777              -1.                 ;USE STANDARD DELIMITERS
45
46          000370'           .LOC    WRIT1+?IBLT     ;RESUME AFTER WRIT1 PACKET
47
48
```

Figure B-5. ?WRITE Parameter Packets (page 2, lines 13-60 and page 3, lines 1-20)(continues)

```
49                WRIT2:
50                B:
51        000370'         .LOC    B+?ICH  ;SECOND WRITE PACKET
52 00370'000000           0
53        000371'         .LOC    B+?ISTI
54 00371'040012           ?OFOT+?PTDS+?ICRF
55        000372'         .LOC    B+?ISTO
56 00372'000000           0
57        000373'         .LOC    B+?IBAD
58 00373'000524"          IOBF2*2
59        000374'         .LOC    B+?IRES
60 00374'000000           0


 0003 EXAMP
01        000375'         .LOC    B+?IRCL
02 00375'000170           120.
03        000376'         .LOC    B+?IRLR
04 00376'000000           0
05        000377'         .LOC    B+?IRNH
06 00377'000000           0
07        000400'         .LOC    B+?IRNL
08 00400'000000           0
09        000401'         .LOC    B+?IFNP
10 00401'001010"          LPTPR*2
11        000402'         .LOC    B+?IMRS
12 00402'000000           0
13        000403'         .LOC    B+?IDEL
14 00403'177777           -1
15        000404'         .LOC    WRIT2+?IBLT     ;END OF WRIT2 PACKET
16
17 00404'040114 LPTPR:    .TXT "@LPT"
18        050124
19        000000
20
```

Figure B-5. ?WRITE Parameter Packets (page 2, lines 13-60 and page 3, lines 1-20) (concluded)

The two packets are identical except for the buffer pointer contained in each. Each packet is used in both opening and writing to the printer.

The two ?READ parameter packets are shown in Figure B-6. Just as the two ?WRITE packets were identical except for the buffer pointer, the two ?READ packets are identical except for the buffers they reference and the filenames they point to: File 1 and File 2.

```
21                                          ;FIRST I/O PACKET FOR ?OPEN AND ?READ
22
23              READ1:
24              C:
25        000407'           .LOC    C+?ICH
26 00407'000000 0                           ;CHANNEL
27        000410'           .LOC    C+?ISTI
28 00410'040022 ?OFIN+?RTDS+?ICRF           ;READING ONLY, DATA SENSITIVE
29                                          ;RECORDS, STARTING AT NEXT RECORD
30                                          ;EACH TIME.
31        000411'           .LOC    C+?ISTO
32 00411'000000 0
33        000412'           .LOC    C+?IBAD
34 00412'000334" IOBF1*2                    ;I/O BUFFER POINTER
35        000413'           .LOC    C+?IRES
36 00413'000000 0                           ;RESERVED.
37        000414'           .LOC    C+?IRCL
38 00414'000170 120.                        ;120 CHAR. RECORDS MAX.
39        000415'           .LOC    C+?IRLR
40 00415'000000 0                           ;BYTES READ.
41        000416'           .LOC    C+?IRNH
42 00416'000000 0                           ;READ NEXT RECORD
43        000417'           .LOC    C+?IRNL
44 00417'000000 0                           ;DITTO
45        000420'           .LOC    C+?IFNP
46 00420'001046" DATP1*2    ;POINTER TO DATA FILENAME, FILE1
47        000421'           .LOC    C+?IMRS
48 00421'000000 0                           ;BLOCK SIZE IRRELEVANT
49        000422'           .LOC    C+?IDEL
50 00422'177777 -1                          ;USE STANDARD DELIMITERS
51
52        000423'           .LOC    C+?IBLT ;RESUME AFTER END OF ?WRIT1
53
54 00423'043111 DATP1:      .TXT    *FILE1*
55        046105
56        030400
57
58
59              READ2:
60              D:                          ;2ND PACKET SAME AS 1ST, EXCEPT
```

Figure B-6. ?READ Parameter Packets (page 3, lines 21-60 and page 4, lines 01-34) (continues)

```
 0004 EXAMP
01                                              ;FILENAME IS FILE2.
02        000426'           .LOC    D+?ICH
03 00426'000000             0
04        000427'           .LOC    D+?ISTI
05 00427'040022             ?OFIN+?RTDS+?ICRF
06        000430'           .LOC    D+?ISTO
07 00430'000000             0
08        000431'           .LOC    D+?IBAD
09 00431'000524"            IOBF2*2
10        000432'           .LOC    D+?IRES
11 00432'000000             0
12        000433'           .LOC    D+?IRCL
13 00433'000170             120.
14        000434'           .LOC    D+?IRLR
15 00434'000000             0
16        000435'           .LOC    D+?IRNH
17 00435'000000             0
18        000436'           .LOC    D+?IRNL
19 00436'000000             0
20        000437'           .LOC    D+?IFNP
21 00437'001104"            DATP2*2
22        000440'           .LOC    D+?IMRS
23 00440'000000             0
24        000441'           .LOC    D+?IDEL
25 00441'177777             -1
26        000442'           .LOC    READ2+?IBLT     ;RESUME AFTER READ2
27
28 00442'043111 DATP2:      .TXT *FILE2*
29        046105
30        031000
31
32
33
34
```

Figure B-6. ?READ Parameter Packets (page 3, lines 21-60 and page 4, lines 01-34) (concluded)

Finally, the task initiation packet is shown in Figure B-7. A standard packet is used since task queuing is not performed.

The start of task activity is MAIN (offset ?DPC), the first location in the task code proper. If a stack fault occurs (and none should), control goes to the default stack fault handler provided by Link, which gives the message STACK OVERFLOW. Two tasks are initiated.

The last program statement, page 5, line 2, sends control to START when the program is first executed.

A cross-reference listing of the entire program is provided in the final Figure, B-8.

```
35                ;TASK PARAMETER PACKET.
36                TASKP:
37                E:
38        000445'         .LOC    E+?DLNK
39 00445'000001           1                       ;STANDARD PACKET.
40        000446'         .LOC    E+?DPRI
41 00446'000001           1                       ;BOTH TASKS AT PRIORITY 1
42        000447'         .LOC    E+?DID
43 00447'000002           2                       ;ASSIGN I.D.S 2 AND 3
44        000450'         .LOC    E+?DPC
45 00450'000014'          MAIN                    ;STARTING ADDRESS OF TASK CODE
46        000451'         .LOC    E+?DAC2
47 00451'000346'          MESG                    ;SEND LIST OF PKTS. TO EACH TASK
48        000452'         .LOC    E+?DSTB
49 00452'000055'          STACK                   ;STACK BASE
50        000453'         .LOC    E+?DSSZ
51 00453'000024           20.                     ;20 WORDS/STACK
52        000454'         .LOC    E+?DSFLT
53 00454'000151'          ERTN                    ;STACK FAULT SIMPLY TAKES
54                                                ;ERROR RETURN
55        000455'         .LOC    E+?DFLGS
56 00455'000000           0                       ;NO TASK SPECIFICATION FLAGS
57        000456'         .LOC    E+?DRES
58 00456'000000           0                       ;RESERVED FOR SYSTEM
59        000457'         .LOC    E+?DNUM
60 00457'000002           2                       ;INITIATE 2 TASKS


 0005 EXAMP
01
02                        .END    START

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure B-7. ?TASK Parameter Packet (page 4, lines 35-60 and page 5, lines 01-02)

```
  0006 EXAMP

A       000354'        2/17    2/19    2/21    2/25    2/27    2/29    2/31
                       2/33    2/35    2/37    2/39    2/41    2/43
B       000370'        2/50    2/51    2/53    2/55    2/57    2/59    3/01
                       3/03    3/05    3/07    3/09    3/11    3/13
C       000407'        3/24    3/25    3/27    3/31    3/33    3/35    3/37
                       3/39    3/41    3/43    3/45    3/47    3/49    3/52
CONT    000027'        1/31    1/48
D       000426'        3/60    4/02    4/04    4/06    4/08    4/10    4/12
                       4/14    4/16    4/18    4/20    4/22    4/24
DATP1   000423'        3/46    3/54
DATP2   000442'        4/21    4/28
E       000445'        4/37    4/38    4/40    4/42    4/44    4/46    4/48
                       4/50    4/52    4/55    4/57    4/59
EOF     000053'        1/34    1/49
EFTN    000151'        1/15    1/17    1/26    1/30    1/37    1/43    1/47
                       1/56    1/59    4/53
IOBF1   000156'        2/03    2/28    3/34
IOBF2   000252'        2/04    2/58    4/09
LOK     000054'        1/12    1/35    1/44    1/50
LPTPR   000404'        2/40    3/10    3/17
MAIN    000014'        1/22    4/45
MES1    000350'        2/05    2/06
MES2    000352'        2/06    2/10
MESG    000346'        2/05    4/47
READ1   000407'        2/08    3/23
READ2   000426'        2/10    3/59    4/26
RREC    000155'        1/56    2/01
STACK   000055'        1/52    4/49
START   000000'  EN    1/03    1/12    5/02
TASKP   000445'        1/17    4/36
WRIT1   000354'        2/09    2/18    2/46
WRIT2   000370'        2/11    2/49    3/15
.KILL   000002   XN    1/19
.REC    000001   XN    1/37
.TASK   000003   XN    1/17
.XMT    000004   XN    1/15    1/47
?KILL   003047   MC    1/18
?OPEN   002222   MC    1/25    1/29
?READ   002270   MC    1/33
?REC    003320   MC    1/36
?RETU   002415   MC    1/58
?TASK   002770   MC    1/16
?WRIT   002313   MC    1/42
?XCAL   000001         1/15    1/17    1/19    1/26    1/30    1/34    1/37
                       1/43    1/47    1/59
?XMT    003275   MC    1/14    1/46
```

Figure B-8. Program Cross-Reference

End of Appendix

Appendix C
System Tables Built in the User Context

The Link utility will build several types of tables in the user context of every executable program. All of these tables are defined in the user parameter file, PARU.SR. With few exceptions, you must not modify the contents of any of these tables. The text states wherever limited modification is permitted.

Every executable program will have a *User Status Table (UST)* and at least one *Task Control Block (TCB)*. If the program uses general procedures residing in overlays or shared routines, or if it uses primitive overlay calls, then Link will build additional tables into the user context to manage these resources. Finally, if the program specifies Unlabeled Common, Link will reserve this in the tables area. Link symbol ?USTART points to the first word of user code; this is the first word after all system tables (including Unlabeled Common). Figure C-1 shows the order that the Link utility (or Binder) builds these items into the user context.

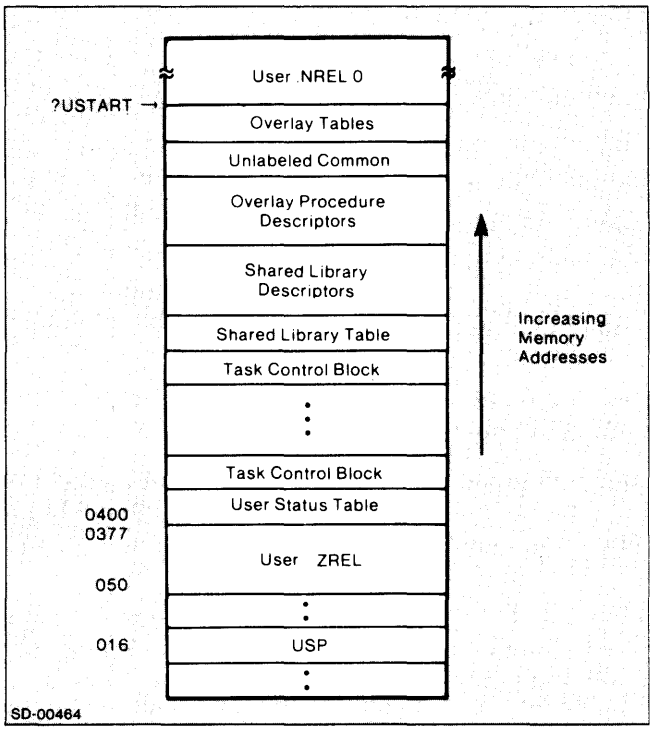


Figure C-1. System Tables Map

User Status Table (UST)

Each program will have a User Status Table (UST). The UST records diverse information pertinent to each program. This information includes task information, the revision level of the program, the starting address of the primitive overlay directory, a count of shared and unshared 2048-byte memory pages, and several pieces of information which are meaningful only to the operating system. Table C-1 illustrates the contents of the UST.

Table C-1. User Status Table

Offset Mnemonic	Contents
USTEZ	Number of memory words to be saved for each task (consult the discussion of TCB offset ?TELN).
USTES	Starting address of task area to be saved (also see TCB offset ?TELN).
USTSS	Reserved for system use.
USTSE	Reserved for system use.
USTS1	Reserved for system use.
USTS2	Reserved for system use.
USTDA	Reserved for system use.
USTFL	Reserved for system use.
USTSL	Address of start of Shared Library Directory.
USTIT	Starting address of ?INTWT task's TCB or -1.
USTRV	Revision level of program.
USTTC	Number of tasks allocated for this program.
USTCT	Address of currently active task's TCB.
USTAC	Starting address of active TCB queue or -1.
USTFC	Starting address of inactive TCB queue.
USTBL	Number of 2048-byte unshared memory pages used by the program.
USTOD	Address of first entry in primitive Overlay Node Table.
USTST	Number of the first shared memory page.
USTSZ	Number of 2048-byte shared memory pages used by the program.

System call ?INTWT reserves a task which gains control on a CTRL-C CTRL-A sequence (see Chapter 9). UST offset USTIT contains the starting address of this task's TCB.

Each program file can have an associated revision level consisting of a major and a minor revision level, each in the range 1 to 255. You can set these levels with either the .REV Macroassembler pseudo-op or with a corresponding high-level language statement. You can read the revision level with the CLI REVISION command. USTRV contains the program's revision level.

Each program file will have from 1 through 32 distinct tasks, and thus from 1 through 32 TCBs allocated to manage the tasks. You set the number of tasks and TCBs with either the .TSK pseudo-op or with the /TASKS= Link function switch. USTCT, USTAC, and USTFC contain the addresses of the currently active TCB, the start of the active TCB queue, and the start of the free TCB queue respectively.

Main memory pages consist of 2048-byte blocks; the first is page zero. USTBL and USTSZ contain the number of unshared and shared memory pages respectively. USTST contains the logical number of the first shared memory page.

Task Control Block (TCB)

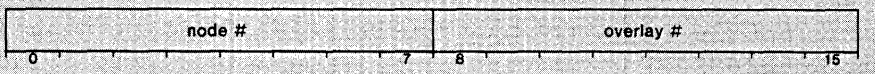
One TCB is allocated for each task in a program. Each TCB structure is listed in Table C-2.

Table C-2. Task Control Block

Offset Mnemonic	Contents
?TLNK	Address of the next TCB in this queue.
?TSTAT	Task event flags; setting any of these places the task into the suspended state:
	?TSPN, ?TSIG — Task has issued a system call whose execution is taking place in system space.
	?TSSG — Task is waiting for an overlay area or a shared routine; or the task has issued an ?XMTW call and the message has not been received; or the task is awaiting a message (?REC).
	?TSSP — Task has been suspended by one of the following calls: ?IDSUS, ?PRSUS, or ?SUS.
	?TSRC — Task has issued ?TRCON and is awaiting a message from the console.
	?TSIW — System action is occurring in preparation for the task's execution.
	?TSGS ... ?TSAB — Flags used by operating system.
	?TSUF — Flag used by high-level languages.
?TSP	Stack pointer.
?TFP	Frame pointer.
?TSL	Stack limit.
?TSO	Address of stack fault handler.
?TAC0	Contents of AC0.
?TAC1	Contents of AC1.
?TAC2	Contents of AC2.
?TAC3	Contents of AC3.
?TPC	Bit 0: state of carry; Bits 1-15: contents of PC.
?TUSP	Contents of location 16, the Unique Storage Position (USP).
?TELN	Starting address of this task's memory save area (or 0 for no save area). If there is a save area, it must be the same size as the save area specified in UST offsets USTEZ and USTES.
?TFPS	Starting address of an 18-word area used to save this task's floating point unit state. System call ?IFPU stores this address in ?TFPS.

(continues)

Table C-2. Task Control Block

Offset Mnemonic	Contents
?TCUD	Description of this task's current general resource (if any). Four descriptors are possible: 1. Bit 0-15=0, no general resource is in use. 2. Bit 0=0, bits 1-15 contain an address which equals or exceeds ?USTART. This is the address of a root procedure entry. 3. Bit 0=0, bits 1-15 contain an address less then ?USTART. This address points to a two-word overlay procedure descriptor: ?OVEDS  ?OVEOF Entry's offset into the overlay. 4. bit 0=1 bits 1-15 contain the address of a two-word shared routine procedure descriptor (Table C-5).
?SLEDS	(used by the system)
?SLEOF	Entry offset into the shared routine.
?TIDPR	Left byte: task ID. Right byte: task priority.
?TSLK	(used by the system)
?TKAD	Address of kill processing routine specified in ?KILAD call by this task.
?TGEX	(used by the system)
?TSYS	System call word.

(concluded)

The Unique Storage Position (USP), location 16, is reserved for system use.

If the User Status Table defines a contiguous area of memory that will be saved for tasks (starting address in USTES, size in USTEZ), then each task can have that area saved for it. To accomplish this, each task wanting the area preserved must define another memory area, the same size as that specified in the UST, where the save area will be stored. Each task desiring this facility must define a separate storage area in memory and specify the start of this area in its TCB offset ?TELN. Initially, ?TELN in each TCB contains 0, indicating no saving of the storage area for this task.

This facility is most useful if UST defines the save area to be in .ZREL memory (i.e., between 50 and 377 inclusive) and each task specifies an unshared .NREL area in ?TELN. This, in effect, gives each task its own private .ZREL area. You should be aware, however, that as the size of the storage area increases, the time required for task scheduling also increases.

Just as each task with a save area must reserve a unique storage area (?TELN), each task wishing to preserve the state of the floating point unit must reserve a unique 18-word storage area. ?TFPS contains the starting address of this area.

If a task has issued a general resource call (?RCALL, etc.) and has control within some resource (root procedure, shared routine, or overlay procedure), then ?TCUD will contain a descriptor of that resource. ?USTART is a Binder symbol that marks the start of user code in the default, unshared partition.

You can find the starting address of a program by using the value contained in TCB offset ?TPC in conjunction with the following procedures involving the DEDIT utility:

1. DEDIT the .PR file.
2. Examine location UST+USTCT (400_8 + 14_8).
3. Note the contents of location 414_8. This is the address of the starting TCB.
4. Add the contents of offset ?TPC (12_8) to the contents of location 414_8. This will give you the address of a word which contains the starting address (low order 15 bits) and the initial status of the carry bit (bit 0).

General - Purpose Task Initiation and Termination Routines

After a ?TASK call, but before control takes the normal return, control is in the task scheduler. With the TCB allocated for the nascent task, yet with this TCB unlinked from both the free and active queues, the task scheduler calls a routine named ?UTSK. If no user-supplied routine with this name is available, then the scheduler calls a dummy routine with this name from URT.LB (the same library containing the task scheduler and task code modules). The dummy routine simply returns control to the caller, with no other effect.

If you specify a routine named ?UTSK in the Link command line, this routine (instead of the dummy ?UTSK from URT.LB) will receive control upon each task initiation. Thus, you have the opportunity to perform some resource allocation to each new task upon its initiation, perhaps by associating a resource with a task ID. When the task scheduler calls ?UTSK, it passes the ?TASK packet address in AC1 and the TCB address in AC2:

Input:

AC1 - ?TASK packet address.

AC2 - TCB address.

Call:

```
EJSR   UTSK          ;AC0 CONTAINS AN
exception-return     ;EXCEPTION CODE ON
normal return        ;EXCEPTION RETURN.
```

In your ?UTSK routine you may destroy the carry and all ACs; the scheduler does not depend on these values being saved. If within the routine you wish to terminate the ?TASK call, then place an exception code in AC0 and send control to the location following the EJSR call; this code will be passed in AC0 to the ?TASK exception return. Otherwise, increment the return location upon completion of ?UTSK processing so that control will take the ?UTSK normal return and the ?TASK call may go to normal completion.

If the ?TASK parameter packet requested task queuing, the scheduler will not call ?UTSK until the task is actually being created.

Just as the scheduler calls ?UTSK upon each task initiation, it calls another routine, ?UKIL, upon each task's termination, with the target task's TCB unlinked from both the free and active queues. This provides you an opportunity to deallocate a resource previously allocated in a ?UTSK routine. Alternatively, it provides a means to do some kill-processing for each task after it has been killed, but before task scheduling occurs.

The task scheduler calls ?UKIL in the following manner:

Input:

AC2 - TCB address.

Call:

```
EJSR    ?UKIL
return
```

As with ?UTSK, ?UKIL may destroy the carry and all ACs. Also as with ?UTSK, if you provide no ?UKIL routine then the system will call a dummy ?UKIL routine from URT.LB which simply returns control to the scheduler.

Shared Library Table (SLT)

If a program uses one or more shared libraries, then the Binder builds a Shared Library Table into the user context. Note that the Link utility does not support shared libraries. Figure C-2 shows the components of the SLT on a large scale. USTSL of the User Status Table points to the start of this table, offset ?SLLDS.

Every SLT contains a 64-word Library Descriptor List followed by a 32-word Memory Page Table, with a one-word terminator at its end. Following these 97 words is the variable portion of the SLT; its size depends upon the number of procedure entries referenced by the program, and the number of libraries and shared routines used by the program.

There are 64 entries in the Library Descriptor List, one for each possible shared library number. Each entry corresponding to a shared library number which is not used by the program (i.e., not specified in the Binder command line creating the program) contains a zero.

Each entry contains the address of a library descriptor corresponding to a shared library that is used.

Following the last entry in the Library Descriptor List is a 32-word table named the *Memory Page Table (MPT).*

The system uses the MPT to keep track of shared routines during the execution of a program. A -1 in an entry indicates that the corresponding memory page is not free for shared routine use. A 0 means that you allocated the page for shared routine use, but that the page is not currently in use. A positive value indicates that the page is in use by a shared routine; each positive value is an address of a *Shared Routine Block Descriptor (SRBD).*

Since shared routines are built as multiples of 2048 bytes, no more than one shared routine can exist in any single memory page. If a shared routine exceeds this, the system places the address of the routine's SRBD in the table entry corresponding to the first memory page used by the routine, and -1's in the following table entries needed to complete the size of the routine.

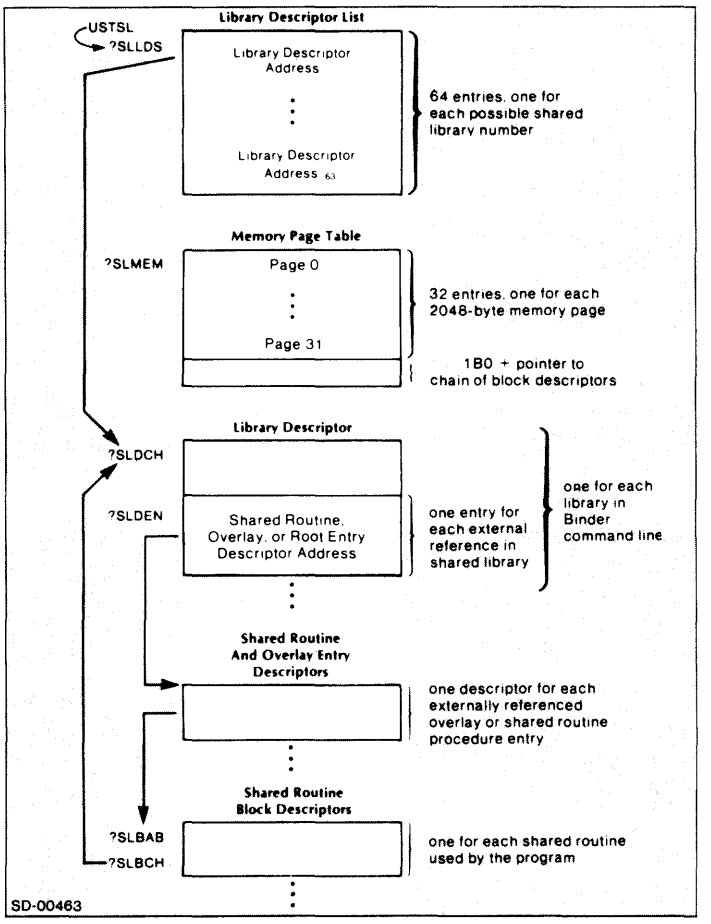


Figure C-2. Shared Library Table Overview

Later, if the system needs to deallocate shared memory pages belonging to a shared routine no longer in use, the system places zeros in the table entries corresponding to pages occupied by the routine. If this routine occupied only one page, then it would change only the routine's SRBD address to 0. Otherwise, both SRBD address and succeeding -1 entries would be changed to zeros.

For each library you specify in the Binder command line creating the program, there is a corresponding *library descriptor;* the address of each library descriptor is listed in the Library Descriptor List described earlier.

Each library descriptor consists of two parts, a fixed four-word initial portion followed by a variable number of entries. Table C-3 shows the structure of the Library Descriptor List.

The system opens each shared library on a different channel. ?SLDCH contains the channel number, and ?SLDNM the number of the library that is opened on that channel. Shared routines in each library may make external references to symbols or procedures outside themselves. The Binder numbers these references from 1 to n, the last or highest numbered external reference. ?SLDLO/?SLDHI list the range of external references made by shared routines called by the program.

Table C-3. Library Descriptor Structure

Offset Mnemonic	Contents
?SLDCH	Channel number (-1, unopened; -2, open-in-progress).
?SLDNM	Library number.
?SLDLO	Lowest resolved external reference.
?SLDHI	Highest resolved external reference.
?SLDEN	Addresses of external symbol references or of external descriptors.
.	
.	
.	

?SLDLO contains the number of the first external reference required by the first shared routine used in the library; ?SLDHI contains the highest numbered external reference. For example, suppose a shared library contains 4 shared routines, SR1 through SR4, and each shared routine makes one external reference, X1 through X4. If a program calls only SR2 and SR3, then this library's descriptor offsets ?SLDLO and ?SLDHI would contain the numbers 2 and 3.

Each external reference in the range ?SLDLO/?SLDHI has a one-word external reference descriptor starting at library descriptor offset ?SLDEN. If bits 1-15 of the descriptor contain a value equal to or greater than ?USTART, then the descriptor is an address of any entry in the root context. Otherwise, it is one of the following:

- 1B0+base address of shared routine procedure entry descriptor; or
- 0B0+base address of overlay procedure entry descriptor.

Shared routine and overlay descriptors are each two words long. Table C-4 illustrates overlay procedure descriptor structure.

Table C-4. Overlay Procedure Descriptor

Offset Mnemonic	Contents
?OVEDS	Bit 0=0 bits 1-6, overlay area number. bits 7-15, overlay number.
?OVEOF	Entry offset into the overlay.

Offset ?OVEOF describes the word displacement into the overlay where the entry is located.

Table C-5 illustrates shared routine entry descriptor structure.

Table C-5. Shared Routine Procedure Descriptor

Offset Mnemonic	Content
?SLEDS	Address of a Shared Routine Block Descriptor (SRBD).
?SLEOF	Entry offset into the shared routine.

Like ?OVEOF, ?SLEOF describes the location of the procedure entry as a word displacement into the shared routine.

There is one Shared Routine Block Descriptor (SRBD) for each shared routine used by the program. Each SRBD is pointed to by at least one Shared Routine Procedure Descriptor, ?SLEDS. Table C-6 outlines SRBD structure.

Table C-6. Shared Routine Block Descriptor

Offset Mnemonic	Contents
?SLBAB	Base address of the memory page containing the shared routine.
?SLBUC	Shared routine use count. Bit zero set if in process of load.
?SLBSZ	Size of memory area for the shared routine.
?SLBFL	Initialized by BIND to be library number.
?SLBBL	Initialized by BIND to be a pointer to a chain of block descriptors. A -1 marks the end.
?SLBHB ?SLBLB	Starting relative block address of the shared routine.
?SLBCH	Address of channel number word, ?SLDCH, in the corresponding Library Descriptor.

?SLBAB contains the starting address of the 2048-byte page containing the shared routine; 0 indicates that the shared routine does not currently reside in main memory. This address may change during the program's execution as the system loads the shared routine into different shared pages at different times.

The use count in ?SLBUC can be in any of the following states:

0 the shared routine is not in use and its main memory can be allocated to other shared routines;

-1 the shared routine has been bound into main memory (Binder /B argument switch);

1-n from 1 through n concurrent calls have been made to this shared routine.

?SLBSZ contains the size of the memory area, in 2048-byte pages, containing or required to contain the shared routine. Each shared routine in a shared library disk file requires some multiple of four disk blocks for storage (since each shared routine is some multiple of 2048-byte long). ?SLBHB and ?SLBLB contain a double precision starting relative block address of the shared routine within the shared library disk file. Since the Shared Library Builder allocates blocks 0-3 for the Shared Library Header (and additional multiples of four blocks only if needed), the first shared routine usually begins at relative block address four.

Finally, ?SLBCH contains the address of the channel number word, ?SLDCH, in the corresponding Library Descriptor.

Primitive Overlay Tables

If the program uses overlays, then Link builds a set of Overlay Tables into the area below ?USTART. Figure C-3 depicts the structure of these tables.

There are two sections in the Primitive Overlay Tables: an initial Overlay Area Table, and a series of Overlay Area Descriptors. The Overlay Area Table is a list of addresses of Overlay Area Descriptor Tables, preceded by a count of the number of Overlay Area Descriptor Tables. UST offset USTOD points to the first Area Descriptor Table for each overlay area in the overlay file; i.e., there is one table for each overlay area in the program. Ordinarily there will be one overlay area for each pair of overlay designators in the Link utility command line. If one or more object files within a pair of descriptors directs Link to build additional overlay area(s), then Link builds additional Overlay Area Descriptors.

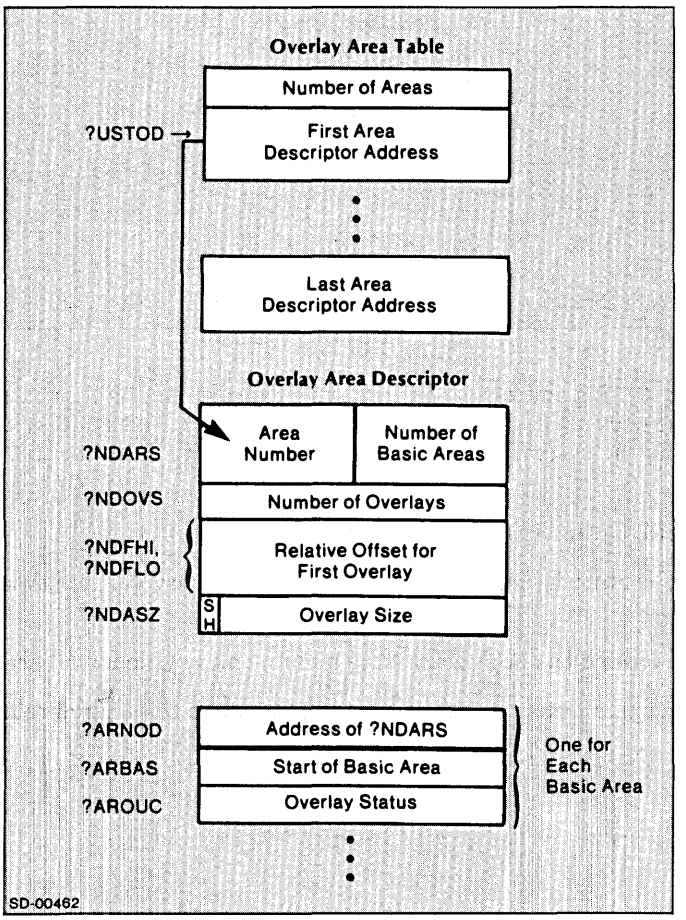


Figure C-3. Overlay Tables

The first offset in each Overlay Area Descriptor Table, ?NDARS, contains the overlay area number being described in the left byte, and the number of basic areas in this overlay area in the right byte. ?NDOVS contains the number of overlays in this area. All overlays in each overlay area are the same size; the contents of ?NDASZ express this size in terms of 256-word (512-byte) disk blocks. Additionally, if the overlay will be read into a shared area, bit 0 of this word is set to 1. ?NDFHI and ?NDFLO contain the relative block offset of overlay 0 for this overlay area. Knowing both the size of each overlay in the overlay area and the relative block position of the first overlay in this area within the total overlay file, the system can then calculate the positions of other overlays in this overlay area.

Offsets ?ARNOD, ?ARBAS, and ?AROUC will be repeated for multiple basic areas within the total overlay area. ?ARNOD merely points to the start of the Overlay Area Descriptor. ?ARBAS contains the starting memory address for the overlay area; this will be page-aligned if the overlay area is in a shared partition. ?AROUC contains the following fields:

Bits	**Meaning**
0	=1, overlay load is in progress. =0, overlay load is not in progress.
1-9	overlay number within the overlay area numbers start at 0.
10-15	use count, the number of concurrent callers of this overlay.

If ?AROUC contains the value 077700, the overlay area is empty.

End of Appendix

Appendix D
Obtaining Current AOS Information

Much of the information documented in this manual changes rapidly as new revisions of AOS are released. AOS maintains several facilities for gaining up-to-date information on your current system. The system call ?SINFO, documented in Chapter 9 of this manual, allows you to get the current revision of your operating system, as well as other selected information. In addition, you can access the files PARU.SR and URT.LB to gain current AOS information.

PARU - User Parameter Listing

All parameters documented in this manual appear in the file PARU.SR, and, at assembly time, the Macroassembler replaces each parameter with its numerical value. You can get a current list of all the parameters and their assembled values simply by assembling PARU.SR. The Macroassembler may, of course, generate error messages for the assembly since PARU.SR is not a stand-alone module of code, but the generated listfile. PARU.LS will show the current parameter values in the data field column. To get this assembled listing and send it to the line printer, issue the following command:

```
XEQ MASM/N/P/L=@LPT PARU
```

The PARU listing is shown in Figure D-1. Since the PARU file for your system may have changes and enhancements, we suggest that you generate and consult it instead, if possible.

User Runtime Module Titles and Entry Points

You may obtain a list of the titles and entry points of all user runtime modules, as well as the size of these modules, by performing a library file analysis of library URT.LB. To perform this analysis and send it to the line printer, issue the following command:

```
XEQ LFE/L=@LPT A URT
```

Complete operating instructions for the Library File Editor are given in *AOS Library File Editor User's Manual.*

```
 0001  PARU      AOS ASSEMBLER REV 03.30          09:47:19 01/16/81
01                ; COPYRIGHT (C) DATA GENERAL CORPORATION 1977,1978,1979,1980
02                ; ALL RIGHTS RESERVED.
03                ; LICENSED MATERIAL-PROPERTY OF DATA GENERAL CORPORATION
04
                          .TITL   PARU
06
07                ;=========================================;
08                ; AOS USER PARAMETER DEFINITION FILE.      ;
09                ;=========================================;
10
11
12
13                ; SYMBOL DEFINITION REQUIREMENTS ARE THE FOLLOWING:
14                ; 1) ALL SYMBOLS OF THE FORM ?XXXX ARE RESERVED FOR
15                ;    DEFINING PARAMETERS USED JOINTLY BY THE OPERATING
16                ;    SYSTEM AND USER ASSEMBLED PROGRAMS.
17                ; 2) ALL SYMBOLS OF THE FORM ERXXX ARE RESERVED FOR DEFINING
18                ;    OPERATING SYSTEM ERROR CODES
19                ; 3) ALL SYMBOLS OF THE FORM USTXX ARE RESERVED FOR DEFINING
20                ;    THE USER STATUS TABLE.
21                ; 4) DON'T ADD ANY SYMBOLS TO PARU UNLESS THEY CONFORM
22                ;    TO THESE RULES.
```

Figure D-1. PARU.LS Parameter Listing (continues)

```
!0002  PARU
01
02
03              ;SYSTEM ERROR CODES  (1-277)
04
05        000001 .DUSR ERICM=    1        ;ILLEGAL SYSTEM COMMAND
06        000002 .DUSR ERFNO=    2        ;CHANNEL NOT OPEN
07        000003 .DUSR EROPR=    3        ;CHANNEL ALREADY OPEN
08        000004 .DUSR ERSAL=    4        ;SHARED I/O REQ NOT MAP SLOT ALIGNED
09        000005 .DUSR ERMEM=    5        ;INSUFFICIENT MEMORY AVAILABLE
10        000006 .DUSR ERADR=    6        ;ILLEGAL STARTING ADDRESS
11        000007 .DUSR EROVN=    7        ;ILLEGAL OVERLAY NUMBER
12        000010 .DUSR ERTIM=    10       ;ILLEGAL TIME ARGUMENT
13        000011 .DUSR ERNOT=    11       ;NO TASK CONTROL BLOCK AVAILABLE
14        000012 .DUSR ERXMT=    12       ;SIGNAL TO ADDRESS ALREADY IN USE
15        000013 .DUSR ERQTS=    13       ;ERROR IN QTASK REQUEST
16        000014 .DUSR ERTID=    14       ;TASK I.D. ERROR
17        000015 .DUSR ERDCH=    15       ;DATA CHANNEL MAP FULL
18        000016 .DUSR ERMPR=    16       ;SYSTEM CALL PARAMETER ADDRESS ERROR
19        000017 .DUSR ERABT=    17       ;TASK NOT FOUND FOR ABORT
20        000020 .DUSR ERIRB=    20       ;INSUFFICIENT ROOM IN BUFFER
21        000021 .DUSR ERSPC=    21       ;FILE SPACE EXHAUSTED
22        000022 .DUSR ERSFT=    22       ;USER STACK FAULT
23        000023 .DUSR ERDDE=    23       ;DIRECTORY DOES NOT EXIST
24        000024 .DUSR ERIFC=    24       ;ILLEGAL FILENAME CHARACTER
25        000025 .DUSR ERFDE=    25       ;FILE DOES NOT EXIST
26        000026 .DUSR ERNAE=    26       ;FILE NAME ALREADY EXISTS
27        000027 .DUSR ERNAD=    27       ;NON-DIRECTORY ARGUMENT IN PATHNAME
28        000030 .DUSR EREOF=    30       ;END OF FILE
29        000031 .DUSR ERDID=    31       ;DIRECTORY DELETE ERROR
30        000032 .DUSR ERWAD=    32       ;WRITE ACCESS DENIED
31        000033 .DUSR ERRAD=    33       ;READ ACCESS DENIED
32        000034 .DUSR ERAWD=    34       ;APPEND AND/OR WRITE ACCESS DENIED
33        000035 .DUSR ERNMC=    35       ;NO CHANNELS AVAILABLE
34        000036 .DUSR ERSRL=    36       ;RELEASE OF NON-ACTIVE SHARED SLOT
35        000037 .DUSR ERPRP=    37       ;ILLEGAL PRIORITY
36        000040 .DUSR ERBMX=    40       ;ILLEGAL MAX SIZE ON PROCESS CREATE
37        000041 .DUSR ERPTY=    41       ;ILLEGAL PROCESS TYPE
38        000042 .DUSR ERCON=    42       ;CONSOLE DEVICE SPECIFICATION ERROR
39        000043 .DUSR ERNSW=    43       ;SWAP FILE SPACE EXHAUSTED
40        000044 .DUSR ERIBS=    44       ;DEVICE ALREADY IN SYSTEM
41        000045 .DUSR ERDNM=    45       ;ILLEGAL DEVICE CODE
42        000046 .DUSR ERSHP=    46       ;ERROR ON SHARED PARTITION SET
43        000047 .DUSR ERRMP=    47       ;ERROR ON REMAP CALL
44        000050 .DUSR ERGSG=    50       ;ILLEGAL GHOST GATE CALL
45        000051 .DUSR ERPRN=    51       ;NUMBER OF PROCESSES EXCEEDS 64.
46        000052 .DUSR ERNEF=    52       ;IPC MESSAGE EXCEEDS BUFFER LENGTH
47        000053 .DUSR ERIVP=    53       ;INVALID PORT NUMBER
48        000054 .DUSR ERNMS=    54       ;NO MATCHING SEND
49        000055 .DUSR ERNOR=    55       ;NO OUTSTANDING RECEIVE
50        000056 .DUSR ERIOP=    56       ;ILLEGAL ORIGIN PORT
51        000057 .DUSR ERIDP=    57       ;ILLEGAL DESTINATION PORT
52        000060 .DUSR ERSEN=    60       ;INVALID SHARED LIBRARY REFERENCE
53        000061 .DUSR ERIRL=    61       ;ILLEGAL RECORD LENGTH SPECIFIED(=0)
54        000062 .DUSR ERARC=    62       ;ATTEMPT TO RELEASE CONSOLE DEVICE
55        000063 .DUSR ERDAI=    63       ;DEVICE ALREADY IN USE
56        000064 .DUSR ERARU=    64       ;ATTEMPT TO RELEASE UNASSIGNED DEVICE
57        000065 .DUSR ERACU=    65       ;ATTEMPT TO CLOSE UNOPEN CHANNEL/DEVICE
58        000066 .DUSR ERITC=    66       ;I/O TERMINATED BY CLOSE
59        000067 .DUSR ERLTL=    67       ;LINE TOO LONG
60        000070 .DUSR ERPAR=    70       ;PARITY ERROR
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0003  PARU
01        000071 .DUSR EREXC=     71      ;RESDENT PROC TRIED TO PUSH (.EXEC)
02        000072 .DUSR ERNDR=     72      ;NOT A DIRECTORY
03        000073 .DUSR ERNSA=     73      ;SHARED I/O REQUEST NOT TO SHARED AREA
04        000074 .DUSR ERSNM=     74      ;ATTEMPT TO CREATE > MAX # SONS
05        000075 .DUSR ERFIL=     75      ;FILE READ ERROR
06        000076 .DUSR ERDTO=     76      ;DEVICE TIMEOUT
07        000077 .DUSR ERIOT=     77      ;WRONG TYPE I/O FOR OPEN TYPE
08        000100 .DUSR ERFTL=     100     ;FILENAME TOO LONG
09        000101 .DUSR ERBOF=     101     ;POSITIONING BEFORE BEGINNING OF FILE
10        000102 .DUSR ERPRV=     102     ;CALLER NOT PRIVILEGED FOR THIS ACTION
11        000103 .DUSR ERSIM=     103     ;SIMULTANEOUS REQUESTS ON SAME CHANNEL
12        000104 .DUSR ERIFT=     104     ;ILLEGAL FILE TYPE
13        000105 .DUSR ERNRD=     105     ;INSUFFICIENT ROOM IN DIRECTORY
14        000106 .DUSR ERILO=     106     ;ILLEGAL OPEN
15        000107 .DUSR ERPRH=     107     ;ATTEMPT TO ACCESS PROC NOT IN HIERARCHY
16        000110 .DUSR ERBLR=     110     ;ATTEMPT TO BLOCK UNBLOCKABLE PROC
17        000111 .DUSR ERPRE=     111     ;INVALID SYSTEM CALL PARAMETER
18        000112 .DUSR ERGES=     112     ;ATTEMPT TO START MULTIPLE GHOSTS
19        000113 .DUSR ERCIU=     113     ;CHANNEL IN USE
20        000114 .DUSR ERICB=     114     ;INSUFFICIENT CONTIGUOUS DISK BLOCKS
21        000115 .DUSR ERSTO=     115     ;STACK OVERFLOW
22        000116 .DUSR ERIBM=     116     ;INCONSISTENT BIT MAP DATA
23        000117 .DUSR ERBSZ=     117     ;ILLEGAL BLOCK SIZE FOR DEVICE
24        000120 .DUSR ERXMZ=     120     ;ATTEMPT TO XMT ILLEGAL MESSAGE
25        000121 .DUSR ERPUF=     121     ;PHYSICAL UNIT FAILURE
26        000122 .DUSR ERPWL=     122     ;PHYSICAL WRITE LOCK
27        000123 .DUSR ERUOL=     123     ;PHYSICAL UNIT OFFLINE
28        000124 .DUSR ERIOO=     124     ;ILLEGAL OPEN OPTION FOR FILE TYPE
29        000125 .DUSR ERNDV=     125     ;TOO MANY OR TOO FEW DEVICE NAMES
30        000126 .DUSR ERMIS=     126     ;DISK AND FILE SYS REV #'S DON'T MATCH
31        000127 .DUSR ERIDD=     127     ;INCONSISTENT DIB DATA
32        000130 .DUSR ERILD=     130     ;INCONSISTENT LD
33        000131 .DUSR ERIDU=     131     ;INCOMPLETE LD
34        000132 .DUSR ERIDT=     132     ;ILLEGAL DEVICE NAME TYPE
35        000133 .DUSR ERPDF=     133     ;ERROR IN PROCESS UST DEFINITION
36        000134 .DUSR ERVIU=     134     ;LD IN USE, CANNOT RELEASE
37        000135 .DUSR ERSRE=     135     ;SEARCH LIST RESOLUTION ERROR
38        000136 .DUSR ERCGF=     136     ;CAN'T GET IPC DATA FROM FATHER
39        000137 .DUSR ERILB=     137     ;ILLEGAL LIBRARY NUMBER GIVEN
40        000140 .DUSR ERRFM=     140     ;ILLEGAL RECORD FORMAT
41        000141 .DUSR ERARG=     141     ;TOO MANY OR TOO FEW ARGUMENTS TO PMGR
42        000142 .DUSR ERIGM=     142     ;ILLEGAL ?GTMES PARAMETERS
43        000143 .DUSR ERICL=     143     ;ILLEGAL CLI MESSAGE
44        000144 .DUSR ERMRD=     144     ;MESSAGE RECEIVE DISABLED
45        000145 .DUSR ERNAC=     145     ;NOT A CONSOLE DEVICE
46        000146 .DUSR ERMIL=     146     ;ATTEMPT TO EXCEED MAX INDEX LEVEL
47        000147 .DUSR ERICN=     147     ;ILLEGAL CHANNEL
48        000150 .DUSR ERNRR=     150     ;NO RECEIVER WAITING
49        000151 .DUSR ERSRR=     151     ;SHORT RECEIVE REQUEST
50        000152 .DUSR ERTIN=     152     ;TRANSMITTER INOPERATIVE
51        000153 .DUSR ERUNM=     153     ;ILLEGAL USER NAME
52        000154 .DUSR ERILN=     154     ;ILLEGAL LINK #
53        000155 .DUSR ERDPE=     155     ;DISK POSITIONING ERROR
54        000156 .DUSR ERTXT=     156     ;MSG TEXT LONGER THAN SPEC'D.
55        000157 .DUSR ERSTR=     157     ;SHORT TRANSMISSION
56        000160 .DUSR ERHIS=     160     ;ERROR ON HISTOGRAM INIT/DELETE
57        000161 .DUSR ERIRV=     161     ;ILLEGAL RETRY VALUE
58        000162 .DUSR ERASS=     162     ;ASSIGN ERROR - ALREADY YOUR DEVICE
59        000163 .DUSR ERPET=     163     ;MAG TAPE REQ PAST LOGICAL END OF TAPE
60        000164 .DUSR ERSTS=     164     ;STACK TOO SMALL (?TASK)
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0004  PARU
01        000165 .DUSR ERTMT=     165     ;TOO MANY TASKS REQUESTED (?TASK)
02        000166 .DUSR ERSOC=     166     ;SPOOLER OPEN RETRY COUNT EXCEEDED
03        000167 .DUSR ERACL=     167     ;ILLEGAL ACL
04        000170 .DUSR ERWPB=     170     ;?STMAP BUFFER INVALID OR WRITE PROTECTE
05        000171 .DUSR ERINP=     171     ;IPC FILE NOT OPENED BY ANOTHER PROC
06        000172 .DUSR ERFPU=     172     ;FPU HARDWARE NOT INSTALLED
07        000173 .DUSR ERPNM=     173     ;ILLEGAL PROCESS NAME
08        000174 .DUSR ERPNU=     174     ;PROCESS NAME ALREADY IN USE
09        000175 .DUSR ERDCT=     175     ;DISCONNECT ERROR (MODEM CONTROLLED)
10        000176 .DUSR ERIPR=     176     ;NONBLOCKING PROC REQUEST ERROR
11        000177 .DUSR ERSNI=     177     ;SYSTEM NOT INSTALLED
12        000200 .DUSR ERLVL=     200     ;MAX DIRECTORY TREE DEPTH EXCEEDED
13        000201 .DUSR ERROO=     201     ;RELEASING OUT-OF-USE OVERLAY
14        000202 .DUSR ERRDL=     202     ;RESOURCE DEADLOCK
15        000203 .DUSR ERE01=     203     ;FILE IS OPEN, CAN'T EXCLUSIVE OPEN
16        000204 .DUSR ERE02=     204     ;FILE IS EXCLUSIVE OPENED, CAN'T OPEN
17        000205 .DUSR ERIPD=     205     ;INIT PRIVILEGE DENIED
18        000206 .DUSR ERMIM=     206     ;MULTIPLE ?IMSG CALLS TO SAME DCT
19        000207 .DUSR ERLNK=     207     ;ILLEGAL LINK
20        000210 .DUSR ERIDF=     210     ;ILLEGAL DUMP FORMAT
21        000211 .DUSR ERXNA=     211     ;EXEC NOT AVAILABLE (MOUNT, ETC.)
22        000212 .DUSR ERXUF=     212     ;EXEC REQUEST FUNCTION UNKNOWN
23        000213 .DUSR ERESO=     213     ;ONLY EXEC'S SONS CAN DO THAT
24        000214 .DUSR ERRBO=     214     ;REFUSED BY OPERATOR
25        000215 .DUSR ERWMT=     215     ;VOLUME NOT MOUNTED
26        000216 .DUSR ERISV=     216     ;ILLEGAL SWITCH VALUE (>65K DECIMAL)
27
28                ;THE NEXT FOUR ERROR CODES MUST BE CONTIGUOUSLY NUMBERED
29
30        000217 .DUSR ERIFN=     217     ;INPUT FILE  DOES NOT EXIST
31        000220 .DUSR EROFN=     220     ;OUTPUT FILE  DOES NOT EXIST
32        000221 .DUSR ERLFN=     221     ;LIST FILE  DOES NOT EXIST
33        000222 .DUSR ERDFN=     222     ;DATA FILE DOES NOT EXIST
34
35        000223 .DUSR ERGFE=     223     ;RECURSIVE GENERIC FILE OPEN FAILURE
36        000224 .DUSR ERNMW=     224     ;NO MESSAGE WAITING
37        000225 .DUSR ERNUD=     225     ;USER DATA AREA DOES NOT EXIST
38        000226 .DUSR ERDVC=     226     ;ILLEGAL DEVICE TYPE FROM AOSGEN
39        000227 .DUSR ERRST=     227     ;AOS RESTART OF SYSTEM CALL
40        000230 .DUSR ERFUR=     230     ;PROBABLY FATAL HARDWARE RUNTIME ERROR
41        000231 .DUSR ERCFT=     231     ;USER COMMERCIAL STACK FAULT
42        000232 .DUSR ERFFT=     232     ;USER FLOATING POINT STACK FAULT
43        000233 .DUSR ERUAE=     233     ;USER DATA AREA ALREADY EXISTS
44        000234 .DUSR ERISO=     234     ; ILLEGAL SCREEN-EDIT REQUEST (PMGR)
45        000235 .DUSR ERDDH=     235     ; "?SEND" DESTINATION DEVICE HELD BY "^S
46        000236 .DUSR EROVR=     236     ; DATA OVERRUN ERROR
47        000237 .DUSR ERCPD=     237     ;CONTROL POINT DIRECTORY MAX SIZE EXCEED
48        000240 .DUSR ERNSD=     240     ;SYS OR BOOT DISK NOT PART OF MASTER LD
49        000241 .DUSR ERUSY=     241     ;UNIVERSAL SYSTEM, YOU CAN'T DO THAT
50        000242 .DUSR EREAD=     242     ;EXECUTE ACCESS DENIED
51        000243 .DUSR ERFIX=     243     ;CAN'T INIT LD, RUN FIXUP ON IT
52        000244 .DUSR ERFAD=     244     ;FILE ACCESS DENIED
53        000245 .DUSR ERDAD=     245     ;DIRECTORY ACCESS DENIED
54        000246 .DUSR ERIAD=     246     ;ATTEMPT TO DEFINE > 1 SPECIAL PROC
55        000247 .DUSR ERIND=     247     ;NO SPECIAL PROCESS IS DEFINED
56        000250 .DUSR ERPRO=     250     ;ATTEMPT TO ISSUE MCA REQUEST WITH
57                                        ;DIRECT I/O IN PROGRESS
58        000251 .DUSR ERDIO=     251     ;ATTEMPT TO ISSUE MCA DIRECT I/O WITH
59                                        ;OUTSTANDING REQUESTS
60        000252 .DUSR ERLTK=     252     ;LAST TASK WAS KILLED
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0005  PARU
01          000253 .DUSR ERLRF=     253      ;RESOURCE LOAD OR RELEASE FAILURE
02          000254 .DUSR ERNNL=     254      ;ZERO LENGTH FILENAME SPECIFIED
03
04                 ;SYNCHRONOUS LINE ERROR CODES
05          000255 .DUSR ERBOV=     ERNNL+1 ;BUFFER OVERFLOW
06          000256 .DUSR ERNAK=     ERBOV+1 ;TRANSMISSION FAILURE (NAK COUNT)
07          000257 .DUSR ERTOF=     ERNAK+1 ;TRANSMISSION FAILURE (TIMEOUTS)
08          000260 .DUSR ERDIS=     ERTOF+1 ;DISCONNECT OCCURRED ON SYNC LINE
09          000261 .DUSR EREOT=     ERDIS+1 ;EOT CHARACTER RECEIVED
10          000262 .DUSR EROTH=     EREOT+1 ;POSSIBLE LOST DATA ON HASP LINE
11          000263 .DUSR ERDCU=     EROTH+1 ;DCU INOPERATIVE (CAN'T BE INITIALIZED)
12          000264 .DUSR ERCNV=     ERDCU+1 ;CONVERSATIONAL REPLY RECEIVED
13          000265 .DUSR EREPL=     ERCNV+1 ;END OF POLLING LIST REACHED
14          000266 .DUSR ERIRT=     EREPL+1 ;ILLEGAL RELATIVE TERMINAL NUMBER
15          000267 .DUSR ERRVI=     ERIRT+1 ;RVI RESPONSE RECEIVED
16          000270 .DUSR ERLIN=     ERRVI+1 ;ILLEGAL LINE NUMBER
17          000271 .DUSR ERPLS=     ERLIN+1 ;NOT ENOUGH SPACE FOR POLL LISTS
18          000272 .DUSR ERCTN=     ERPLS+1 ;CONTENTION SITUATION WHILE BIDDING
19          000273 .DUSR ERSEQ=     ERCTN+1 ;OUT-OF-SEQUENCE GEN ENTRY DURING SINIT
20          000274 .DUSR ERNSL=     ERSEQ+1 ;ATTEMPT TO ENABLE NON-SYNC LINE
21          000275 .DUSR ERIMM=     ERNSL+1 ;NOT ENOUGH MEMORY FOR POLL/SELECT LIST
22          000276 .DUSR EREPE=     ERIMM+1 ;LINE ALREADY ENABLED ON ?SEBL CALL
23          000277 .DUSR ERDSL=     EREPE+1 ;LINE ALREADY DISABLED ON ?SDBL CALL
24          000300 .DUSR ERLNA=     ERDSL+1 ;I/O REQUEST FOR DISABLED LINE
25          000301 .DUSR ERLIS=     ERLNA+1 ;LINE IN SESSION ON ?SSND INITIAL CALL
26          000302 .DUSR ERSCS=     ERLIS+1 ;?SSND CONTINUE WITHOUT LINE IN SESSION
27          000303 .DUSR ERBCT=     ERSCS+1 ;SEND BYTE COUNT EXCEEDS SYSTEM BUFFER
28          000304 .DUSR ERBNK=     ERBCT+1 ;BID ERROR (TOO MANY NAKS)
29          000305 .DUSR ERWAB=     ERBNK+1 ;WABT RECEIVED (HASP LINE ONLY)
30          000306 .DUSR ERBPE=     ERWAB+1 ;USER BUFFER BYTE POINTER INVALID
31          000307 .DUSR ERBRT=     ERBPE+1 ;RETRY COUNT EXCEEDED
32          000310 .DUSR ERETX=     ERBRT+1 ; 'ETX' CODE RECEIVED
33          000311 .DUSR ERISE=     ERETX+1 ;INPUT STATUS ERROR (FORMAT)
34          000312 .DUSR ERFCT=     ERISE+1 ;FAILURE TO CONNECT
35          000313 .DUSR ERUNI=     ERFCT+1 ;UNINTERPRETABLE RESPONSE RECEIVED
36          000314 .DUSR ERENQ=     ERUNI+1 ;ENQ RECEIVED AFTER TIME-OUT
37          000315 .DUSR ERCRC=     ERENQ+1 ;CRC CHECK
38          000316 .DUSR ERINE=     ERCRC+1 ;INITIALIZATION PARAMETER ERROR
39          000317 .DUSR ERTRF=     ERINE+1 ;TRANSMITTER FAILURE ERROR
40          000320 .DUSR ERLNM=     ERTRF+1 ;LINE NOT MULTIPOINT
41          000321 .DUSR ERNCS=     ERLNM+1 ;NOT A CONTROL STATION
42          000322 .DUSR ERNPL=     ERNCS+1 ;POLLING LIST NOT DEFINED
43
44          000323 .DUSR ERITF=     ERNPL+1 ;INCOMPATIBLE LPB TAB FORMAT
45          000324 .DUSR ERPRM=     ERITF+1 ;CANNOT DELETE PERMANENT FILE
46          000325 .DUSR ERSCA=     ERPRM+1 ;SYSTEM CALL ABORT
47          000326 .DUSR ERCAD=     ERSCA+1 ;EXTENDED CONTEXT ALREADY DEFINED
48          000327 .DUSR ERLAB=     ERCAD+1 ;UNREADABLE TAPE LABEL
49          000330 .DUSR ERVOL=     ERLAB+1 ;INCORRECT LABELED TAPE VOLUME MOUNTED
50          000331 .DUSR ERFSI=     ERVOL+1 ;INCORRECT LABELED TAPE FILE SET
51          000332 .DUSR ERSEC=     ERFSI+1 ;INCORRECT LABELED TAPE FILE SECTION NUM
52          000333 .DUSR ERGEN=     ERSEC+1 ;INCORRECT LABELED TAPE FILE GENERATION
53          000334 .DUSR ERVER=     ERGEN+1 ;INCORRECT LABELED TAPE FILE VERSION NUM
54          000335 .DUSR ERNOA=     ERVER+1 ;NO OPERATOR AVAILABLE
55          000336 .DUSR ERREV=     ERNOA+1 ;UNKNOWN LABELED TAPE LABEL REVISION
56          000337 .DUSR ERCAI=     ERREV+1 ;EXTENDED CONTEXT ALREADY INITIALIZED
57          000340 .DUSR ERCNI=     ERCAI+1 ;EXTENDED CONTEXT NOT INITIALIZED
58          000341 .DUSR ERCND=     ERCNI+1 ;EXTENDED CONTEXT NOT DEFINED
59          000342 .DUSR ERMRL=     ERCND+1 ;MEMORY RELEASE ERROR
60          000343 .DUSR ERITP=     ERMRL+1 ;TRANSLATION (?READ/?WRITE) ERROR
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0006  PARU
01          000344 .DUSR ERNAG=     ERITP+1 ;NO SUCH ARGUMENT - ?GTMES
02          000345 .DUSR ERNCF=     ERNAG+1 ;NOT IN CLI FORMAT - ?GTMES
03          000346 .DUSR ERBIF=     ERNCF+1 ;ILLEGAL BIAS FACTOR
04          000347 .DUSR ERTLM=     ERBIF+1 ;CPU TIME LIMIT EXCEEDED
05          000350 .DUSR ERSMX=     ERTLM+1 ;ERROR IN SETTING MAX CPU LIMIT
06          000351 .DUSR ERNM4=     ERSMX+1 ;ELEMENT SIZE NOT A MULTIPLE OF 4
07          000352 .DUSR ERWAK=     ERNM4+1 ;WACK RESPONSE RECEIVED (SYNC LINE)
08          000353 .DUSR ERNAS=     ERWAK+1 ;PROCESS IS NOT A SERVER
09          000354 .DUSR ERCDE=     ERNAS+1 ;CONNECTION DOESN'T EXIST
10          000355 .DUSR ERCTF=     ERCDE+1 ;CONNECTION TABLE FULL
11          000356 .DUSR ERDIU=     ERCTF+1 ;DIRECTORY IN USE-CANNOT DELETE
12          000357 .DUSR ERSHG=     ERDIU+1 ;ATTEMPT TO GROW GHOST SHARED I/O FILE
13          000360 .DUSR ERNIN=     ERSHG+1 ;ILLEGAL DIRECTORY SPECIFICATION
14          000361 .DUSR ERNNA=     ERNIN+1 ;NETWORK NOT AVAILABLE
15          000362 .DUSR ERHAE=     ERNNA+1 ;HOST ALREADY EXISTS
16          000363 .DUSR ERHID=     ERHAE+1 ;ILLEGAL HOST SPECIFICATION
17          000364 .DUSR ERHNE=     ERHID+1 ;HOST DOES NOT EXIST
18          000365 .DUSR ERCAH=     ERHNE+1 ;CAN'T RENAME HOSTS
19          000366 .DUSR EREMB=     ERCAH+1 ;EMPTY MAILBOX ON ?RECNW
20          000367 .DUSR ERRRR=     EREMB+1 ;REMOTE RESOURCE REFERENCE MADE
21          000370 .DUSR ERCMH=     ERRRR+1 ;ATTEMPT TO CREATE MULTIPLE LOCAL HOSTS
22          000371 .DUSR ERNAI=     ERCMH+1 ;NOT AWAITING ?IWKUP
23          000372 .DUSR ERIRP=     ERNAI+1 ;ILLEGAL REMOTE ?PROC PARAMETER(S)
24          000373 .DUSR ERIHN=     ERIRP+1 ;ILLEGAL HOST NAME
25          000374 .DUSR ERNFC=     ERIHN+1 ;NOT PROPER FOR A VIRTUAL CIRCUIT
26          000375 .DUSR ERWSZ=     ERNFC+1 ;HDLC - INVALID WINDOW SIZE
27          000376 .DUSR ERFSZ=     ERWSZ+1 ;INVALID FRAME SIZE
28          000377 .DUSR ERSDA=     ERFSZ+1 ;SEND ACTIVE
29          000400 .DUSR ERCTY=     ERSDA+1 ;INVALID CALL TYPE
30          000401 .DUSR ERDSC=     ERCTY+1 ;REMOTE IS DISCONNECTING
31          000402 .DUSR ERRIE=     ERDSC+1 ;LOCAL RECEIVED INVALID RESPONSE
32          000403 .DUSR ERRCE=     ERRIE+1 ;LOCAL RECEIVED CMDR
33          000404 .DUSR ERCSE=     ERRCE+1 ;LOCAL IS IN "CAN'T SEND"
34                                          ;...CONDITION
35          000405 .DUSR ERLDC=     ERCSE+1 ;LOCAL IS DISCONNECTING
36          000406 .DUSR ERRES=     ERLDC+1 ;LOCAL WAS RESET
37          000407 .DUSR ERBFO=     ERRES+1 ;BUFFER OVERFLOW
38          000410 .DUSR ERRCA=     ERBFO+1 ;RECEIVE ACTIVE
39          000411 .DUSR ERINF=     ERRCA+1 ;INITIALIZATION FAILED
40          000412 .DUSR ERINC=     ERINF+1 ;LOCAL RECEIVED INVALID COMMAND
41          000413 .DUSR ERNHL=     ERINC+1 ;NON-HDLC ENABLE ATTEMPTED
42          000414 .DUSR ERKAD=     ERNHL+1 ;INTERRUPT WAIT TASK ALREADY DEFINED
43          000415 .DUSR ERDSE=     ERKAD+1 ;MAP SLOT ERROR
44          000416 .DUSR ERGBE=     ERDSE+1 ;GET BUFFER ERROR
45          000417 .DUSR ERDIE=     ERGBE+1 ;SYNC DCU INOPERATIVE
46          000420 .DUSR ERFOE=     ERDIE+1 ;ERROR OPENING SLDCU.PR
47          000421 .DUSR ERFRE=     ERFOE+1 ;ERROR READING SLDCU.PR
48          000422 .DUSR ERFCE=     ERFRE+1 ;ERROR CLOSING SLDCU.PR
49          000423 .DUSR ERGME=     ERFCE+1 ;ERROR GETTING MEMORY
50          000424 .DUSR ERUNK=     ERGME+1 ;UNKNOWN ERROR
51          000425 .DUSR ERCBK=     ERUNK+1 ;CONNECTION HAS BEEN BROKEN
52          000426 .DUSR ERNDC=     ERCBK+1 ;ATTEMPTED HDLC CALL WITH NO DCU200
53          000427 .DUSR ERCCS=     ERNDC+1 ;CANNOT CONNECT TO SELF
54          000430 .DUSR ERVNC=     ERCCS+1 ;NO CONNECTION
55
56                  ;DENSITY SELECTION ERROR CODES.
57          000431 .DUSR ERCDN=     ERVNC+1 ;CONTROLLER DOES NOT SUPPORT THIS DENSIT
58          000432 .DUSR ERITD=     ERCDN+1 ;INDECIPHERABLE TAPE DENSITY.
59          000433 .DUSR ERFTM=     ERITD+1 ;FILE/TAPE MISMATCH.
60
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0007  PARU
01                ;MORE LABELLED TAPE ERROR CODES
02          000434 .DUSR ERILL=     ERFTM+1 ;ILLEGAL LABELLED TAPE LEVEL
03          000435 .DUSR ERILC=     ERILL+1 ;ILLEGAL LABELLED TAPE CHARACTER
04          000436 .DUSR ERISQ=     ERILC+1 ;INCORRECT LABELLED TAPE SEQUENCE NUMBER
05          000437 .DUSR ERIBC=     ERISQ+1 ;INCORRECT LABELLED TAPE BLOCK COUNT
06          000440 .DUSR ERVTL=     ERIBC+1 ;VOLID TOO LONG OR NULL
07          000441 .DUSR EROTL=     ERVTL+1 ;OWNER ID TOO LONG
08          000442 .DUSR ERUTL=     EROTL+1 ;USER LABELS TOO LONG OR TOO MANY
09          000443 .DUSR ERREB=     ERUTL+1 ;RECORD LENGTH EXCEEDS BLOCK LENGTH
10
11                ;ARRAY PROCESSOR ERROR CODES
12          000444 .DUSR ERAPB=     ERREB+1 ;AP ALREADY BUSY
13          000445 .DUSR ERAPN=     ERAPB+1 ;AP DOES NOT EXIST
14          000446 .DUSR ERAPL=     ERAPN+1 ;AP WCS IS NOT LOADED
15          000447 .DUSR ERAPR=     ERAPL+1 ;ATTEMPT TO RELEASE A NON-AP PAGE
16          000450 .DUSR ERAPP=     ERAPR+1 ;ATTEMPT TO RELEASE AN AP PAGE
17
18                ;MORE LABELLED TAPE ERRORS
19          000451 .DUSR EROOF=     ERAPP+1 ;ONLY ONE FILE ON ANSI1 TAPE
20          000452 .DUSR ERCDU=     EROOF+1 ;CANNOT DELETE UNEXPIRED FILE
21
22                ;MORE NON-LABELLED TAPE ERRORS
23          000453 .DUSR ERNPC=     ERCDU+1 ; PROCESS CONSOLE DOES NOT EXIST
24          000454 .DUSR ERHTT=     ERNPC+1 ;HISTOGRAM TARGET PROCESS TERMINATION
25          000455 .DUSR ERHIP=     ERHTT+1 ;ERROR DETECTED WHILE HISTOGRAM IN PROGR
26          000456 .DUSR ERIIP=     ERHIP+1 ;INVALID IPC MESSAGE
27          000457 .DUSR ERMUS=     ERIIP+1 ;MULTIPLE USERS OF FILE;CANNOT TRUNCATE
28          000460 .DUSR ERSHR=     ERMUS+1 ;SHARED FILE;CANNOT TRUNCATE
29          000461 .DUSR ERTRC=     ERSHR+1 ;FILE BEING TRUNCATED;CANNOT OPEN
30          000462 .DUSR ERFRM=     ERTRC+1 ;FRAMING ERROR
31          000463 .DUSR ERIDI=     ERFRM+1 ;INTERNAL DIRECTORY INCONSISTENCY
32          000464 .DUSR ERCOF=     ERIDI+1 ;COMMERCIAL FAULT
33          000465 .DUSR ERFXF=     ERCOF+1 ;FIXED POINT FAULT
34          000466 .DUSR ERFLF=     ERFXF+1 ;FLOATING POINT FAULT
35          000467 .DUSR ER9TP=     ERFLF+1 ;CAN'T SELECT EVEN PARITY ON 9 TRACK DRI
36          000470 .DUSR ERTMR=     ER9TP+1 ;TOO MANY TAPE RETRIES.
37          000471 .DUSR ERNDS=     ERTMR+1 ;NO STATISTICS AVAILABLE FOR THIS DEVICE
38          000472 .DUSR ERNUS=     ERNDS+1 ;NO STATISTICS AVAILABLE FOR THIS UNIT
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0008  PARU
01
02              ; SYSTEM CONSTANTS
03
04        000001 .DUSR   ?SIN=   1        ; SYSTEM IMPLEMENTATION NUMBER
05                                        ; 1 = AOS
06                                        ; 2 = AOS/VS
07                                        ; 3 = MP/OS
08                                        ; 4 = RESERVED FOR MUNDIE
09                                        ; 5 = RESERVED FOR HARRINGTON
10                                        ; 6 = RESERVED FOR DUELL
11                                        ; 7 = RESERVED FOR ANDREWS
12
13        000210 .DUSR   ?DFLL=  136.     ; DEFAULT MAXIMUM LINE LENGTH
14        000200 .DUSR   ?MXPL=  128.     ; MAX PATHNAME LENGTH (BYTES)
15        000020 .DUSR   ?MXPN=  16.      ; MAX SIMPLE PROCESS NAME LENGTH (BYTES)
16        000020 .DUSR   ?MXUN=  16.      ; MAX USERNAME LENGTH (BYTES)
17        000040 .DUSR   ?MXHN=  32.      ; MAX HOST NAME LENGTH (BYTES)
18                                        ;  MUST MATCH ?MXFN BECAUSE HOST NAMES
19                                        ;  ARE ACTUALLY FILE ENTRIES
20        000100 .DUSR   ?MXFP=  ?MXHN+?MXUN+?MXPN  ; MAX FULL PROCESS NAME
21        000040 .DUSR   ?MXFN=  32.      ; MAX FILENAME LENGTH (BYTES)
22                                        ; ALL OF THE ABOVE INCLUDE THE
23                                        ; TRAILING NULL BYTE
24
25        000100 .DUSR   ?MXPID= 64.      ; HIGHEST LEGAL PID
26        002000 .DUSR   ?MXIPC= 1024.    ; MAX INTERHOST IPC LENGTH
27        002000 .DUSR   ?MXIO=  1024.    ; MAX INTERHOST I/O XFER LENGTH
28        000377 .DUSR   ?MXACL= 255.     ; MAX ACL LENGTH
29        177770 .DUSR   ?HIDSC= -8.      ; HOST ID SHIFT COUNT
30        077400 .DUSR   ?HMSK=  77400    ; HOST ID MASK
31
32
33              ; ENTRY TYPE RANGES
34
35        000000 .DUSR   ?SMIN=  0.       ; SYSTEM MINIMUM
36        000077 .DUSR   ?SMAX=  63.      ; SYSTEM MAXIMUM
37
38        000100 .DUSR   ?DMIN=  ?SMAX+1  ; DGC MINIMUM
39        000177 .DUSR   ?DMAX=  127.     ; DGC MAXIMUM
40
41        000200 .DUSR   ?UMIN=  ?DMAX+1  ; USER MINIMUM
42        000377 .DUSR   ?UMAX=  255.     ; USER MAXIMUM
43
44              ; SYSTEM ENTRY TYPES
45
46              ; MISC
47
48        000000 .DUSR   ?FLNK=  ?SMIN    ; LINK
49        000001 .DUSR   ?FSDF=  ?FLNK+1  ; SYSTEM DATA FILE
50        000002 .DUSR   ?FMTF=  ?FSDF+1  ; MAG TAPE FILE
51        000003 .DUSR   ?FGFN=  ?FMTF+1  ; GENERIC FILE NAME
52        000004 .DUSR   ?FDCF=  ?FGFN+1  ; DEVICE CONFIGURATION FILE
53        000005 .DUSR   ?FLCF=  ?FDCF+1  ; LINK CONFIGURATION FILE
54
55              ; DIRECTORIES (DO NOT CHANGE THEIR ORDER)
56
57        000012 .DUSR   ?FDIR=  10.      ; DISK DIRECTORY
58        000013 .DUSR   ?FLDU=  ?FDIR+1  ; LD ROOT DIRECTORY
59        000014 .DUSR   ?FCPD=  ?FLDU+1  ; CONTROL POINT DIRECTORY
60        000015 .DUSR   ?FMTV=  ?FCPD+1  ; MAG TAPE VOLUME
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0009  PARU
01
02        000012 .DUSR   ?LDIR=  ?FDIR   ; LOW DIR TYPE
03        000015 .DUSR   ?HDIR=  ?FMTV   ; HIGH DIR TYPE
04
05        000013 .DUSR   ?LCPD=  ?FLDU   ; LOW CONTROL POINT DIR TYPE
06        000014 .DUSR   ?HCPD=  ?FCPD   ; HIGH CONTROL POINT DIR TYPE
07
08               ; UNITS
09
10        000024 .DUSR   ?FDKU=  20.     ; DISK UNIT
11        000025 .DUSR   ?FMCU=  ?FDKU+1 ; MULTIPROCESSOR COMMUNICATIONS UNIT
12        000026 .DUSR   ?FMTU=  ?FMCU+1 ; MAG TAPE UNIT
13        000027 .DUSR   ?FLPU=  ?FMTU+1 ; DATA CHANNEL LINE PRINTER
14        000030 .DUSR   ?FLPD=  ?FLPU+1 ; DATA CHANNEL LP2 UNIT
15
16        000030 .DUSR   ?FLTU=  ?FLPU+1 ; LABELLED MAG TAPE UNIT
17                                       ; ***** NO LONGER USED *****
18
19        000024 .DUSR   ?LUNT=  ?FDKU   ; LOW UNIT TYPE
20        000030 .DUSR   ?HUNT=  ?FLPD   ; HIGH UNIT TYPE
21
22               ; IPC ENTRY TYPES
23
24        000036 .DUSR   ?FIPC=  30.     ; IPC PORT ENTRY
25
26        000040 .DUSR   ?FSPR=  ?FIPC+2 ; SPOOLABLE PERIPHERAL
27        000041 .DUSR   ?FQUE=  ?FSPR+1 ; EXEC'S QUEUES
28        000042 .DUSR   ?FGLT=  ?FQUE+1 ; LABELED TAPE
29
30        000044 .DUSR   ?FPLO=  ?FQUE+3 ;LOW RANGE OF PERIPHERAL TYPES
31        000044 .DUSR   ?FPIL=  ?FPLO   ;  LOW RANGE OF INPUT ONLY TYPES
32        000044 .DUSR   ?FTRA=  ?FPIL   ;    TAPE READER
33        000045 .DUSR   ?FCRA=  ?FTRA+1 ;    CARD READER
34        000051 .DUSR   ?FPIH=  ?FCRA+4 ;  HIGH RANGE OF INPUT ONLY TYPES
35        000052 .DUSR   ?FPOL=  ?FPIH+1 ;  LOW RANGE OF OUTPUT ONLY TYPES
36        000052 .DUSR   ?FTPA=  ?FPOL   ;    TAPE PUNCH
37        000053 .DUSR   ?FPLA=  ?FTPA+1 ;    DIGITAL PLOTTER
38        000054 .DUSR   ?FLPA=  ?FPLA+1 ;    PIO LINE PRINTER
39        000055 .DUSR   ?FLPC=  ?FLPA+1 ;    LP2 LINE PRINTER(PLOTTER)
40        000060 .DUSR   ?FPOH=  ?FLPC+3 ;  HIGH RANGE OF OUTPUT ONLY TYPES
41        000061 .DUSR   ?FCON=  ?FPOH+1 ;  CONSOLE (HARDCOPY OR CRT)
42        000062 .DUSR   ?FPHI=  ?FCON+1 ;HIGH RANGE OF PERIPHERAL TYPES
43
44        000036 .DUSR   ?LIPC=  ?FIPC   ; LOW IPC TYPE
45        000062 .DUSR   ?HIPC=  ?FPHI   ; HIGH IPC TYPE
46
47        000063 .DUSR   ?FNLO=  ?HIPC+1         ;LOW NETWORK FILE TYPE ENTRY
48        000063 .DUSR   ?FREM=  ?FNLO           ; REMOTE HOST - REMA ACCESS
49        000064 .DUSR   ?FHST=  ?FREM+1         ; REMOTE HOST - X25 SVC ACCESS
50        000065 .DUSR   ?FNPN=  ?FHST+1         ; NETWORK PROCESS NAME
51        000066 .DUSR   ?FPVC=  ?FNPN+1         ; REMOTE HOST - X25 PVC ACCESS
52        000067 .DUSR   ?FNHI=  ?FPVC+1         ;HIGH NETWORK FILE TYPE ENTRY
53
54               ; SYNC LINE FILES TYPE
55        000074 .DUSR   ?FSYN=  60.     ; SYNC LINE
56
57               ; SNA FILE TYPES
58        000076 .DUSR   ?FLUG=  62.     ; LOGICAL UNIT GROUP
59
60               ; DGC ENTRY TYPES
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0010  PARU
01
02        000100 .DUSR   ?FUDF=   ?DMIN    ; USER DATA FILE
03        000101 .DUSR   ?FPRG=   ?FUDF+1  ; PROGRAM FILE
04        000102 .DUSR   ?FUPF=   ?FPRG+1  ; USER PROFILE FILE
05        000103 .DUSR   ?FSTF=   ?FUPF+1  ; SYMBOL TABLE FILE
06        000104 .DUSR   ?FTXT=   ?FSTF+1  ; TEXT FILE
07        000105 .DUSR   ?FLOG=   ?FTXT+1  ; SYSTEM LOG FILE (ACCOUNTING FILE)
08        000106 .DUSR   ?FNCC=   ?FLOG+1  ; FORTRAN CARRIAGE CONTROL FILE
09        000107 .DUSR   ?FLCC=   ?FNCC+1  ; FORTRAN CARRIAGE CONTROL FILE
10        000110 .DUSR   ?FFCC=   ?FLCC+1  ; FORTRAN CARRIAGE CONTROL FILE
11        000111 .DUSR   ?FOCC=   ?FFCC+1  ; FORTRAN CARRIAGE CONTROL FILE
12        000112 .DUSR   ?FPRV=   ?FOCC+1  ; AOS/VS PROGRAM FILE
13        000113 .DUSR   ?FWRD=   ?FPRV+1  ; AZ-TEXT FILE TYPE
14        000114 .DUSR   ?FAFI=   ?FWRD+1  ; RESERVED FILE TYPE
15        000115 .DUSR   ?FAWS=   ?FAFI+1  ; RESERVED FILE TYPE
16        000116 .DUSR   ?FBCI=   ?FAWS+1  ; RESERVED FILE TYPE
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0011  PARU
01
02                ; LOG FILE EVENT CODE TYPES
03
04        000000 .DUSR    ?LSMI=  0         ;SYSTEM LOWER LIMIT
05        001777 .DUSR    ?LSMA=  1023.     ;SYSTEM UPPER LIMIT
06        002000 .DUSR    ?LDMI=  1024.     ;DGC LOWER LIMIT
07        003777 .DUSR    ?LDMA=  2047.     ;DGC UPPER LIMIT
08        004000 .DUSR    ?LUMI=  2048.     ;USER LOWER LIMIT
09        010000 .DUSR    ?LUMAX= 4096.     ;USER UPPER LIMIT
10
11                ;PRESENTLY DEFINED DGC LOG FILE EVENT CODES
12
13        000001 .DUSR    ?LSTR=  1         ;START OF LOG FILE
14        000002 .DUSR    ?LEND=  ?LSTR+1   ;END OF LOG FILE
15        000003 .DUSR    ?LACT=  ?LEND+1   ;RUN TIME INFORMATION
16        000004 .DUSR    ?LDER=  ?LACT+1   ;DEVICE ERROR (MAG TAPE OR DISK)
17        000005 .DUSR    ?LPAD=  ?LDER+1   ;PADDING (FILL FILE TO BLOCK BOUNDARY)
18        000006 .DUSR    ?LERC=  ?LPAD+1   ;ERCC (MEMORY ERROR CORRECTION)
19        000007 .DUSR    ?LPFL=  ?LERC+1   ;POWER FAILURE
20        000010 .DUSR    ?FSGF=  ?LPFL+1   ;FIRST SYSTEM MUNDIE
21        000047 .DUSR    ?LSGF=  ?FSGF+37;LAST SYSTEM MUNDIE
22                ;
23                ; EXEC EVENT CODES
24                ;
25        002000 .DUSR    ?LCNT=  ?LDMI     ;CONSOLE CONNECT-TIME INFORMATION
26        002001 .DUSR    ?LMNT=  ?LCNT+1   ;UNIT MOUNT INFORMATION
27        002002 .DUSR    ?LSUP=  ?LMNT+1   ;SUPERUSER LOG-ON
28        002003 .DUSR    ?LPPT=  ?LSUP+1   ;PAGES PRINTED
29                ;
30                ; NETWORKING EVENT CODES
31                ;
32        002004 .DUSR    ?LNCC=  ?LPPT+1   ; NETWORK CONNECTION CLEARED
33        002005 .DUSR    ?LNERR= ?LNCC+1   ; NETWORK ERROR
34        002006 .DUSR    ?LFUC=  ?LNERR+1; FUNCTIONAL-LEVEL RMA COMPLETION
35        002007 .DUSR    ?LFERR= ?LFUC+1   ; FUNCTIONAL-LEVEL USER ERROR
36                ;
37                ; MUNDIE EVENT CODES
38                ;
39        002010 .DUSR    ?FNGF=  ?LFERR+1;FIRST NON-SYSTEM MUNDIE
40        002047 .DUSR    ?LNGF=  ?FNGF+37;LAST NON-SYSTEM MUNDIE
41                ;
42                ; MORE NETWORKING EVENT CODES
43                ;
44        002050 .DUSR    ?LFTA=  ?LNGF+1   ;FUNCTIONAL LEVEL FTA COMPLETIONS
45                ;
46                ; GENERAL USE EVENT CODE
47                ;
48        00205i .DUSR    ?LCOM=  ?LFTA+1   ;GENERAL COMMENT
49                ;
50                ; SNA EVENT CODE
51                ;
52        002052 .DUSR    ?LSNA=  ?LCOM+1   ;SNA ACCOUNTING LOG
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0012  PARU
01                ;STACK DISPLACEMENTS FOR
02                ;       USER RUNTIME RESOURCE MANAGEMENT
03
04
05          000002 .DUSR    ?VRTN=  2       ; CALLER'S VIRTUAL RETURN
06          000001 .DUSR    ?DESC=  1       ; CALLER'S RESOURCE DESCRIPTION
07                                          ;       0       NO RESOURCE
08                                          ;       >=?USTART ROOT
09                                          ;       1B0     SHARED LIBRARY
10                                          ;       0B0     OVERLAY
11          177774 .DUSR    ?OAC0=  -4      ; CALLER'S AC0
12          177775 .DUSR    ?OAC1=  -3      ; CALLER'S AC1
13          177776 .DUSR    ?OAC2=  -2      ; CALLER'S AC2
14          177777 .DUSR    ?OFP=   -1      ; CALLER'S FRAME POINTER
15          000000 .DUSR    ?ORTN=  0       ; CARRY+RETURN ADDRESS
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10013  PARU
01
02                ; PERIPHERAL DEVICE CHARACTERISTICS (IN PIBC1, PIBC2, & PIBC3)
03
04                ;       IN      PIBC1
05
06        000000 .DUSR    ?CST=   0.      ;SIMULATE TABS
07        000001 .DUSR    ?CSFF=  1.      ;SIMULATE FORM FEEDS
08        000002 .DUSR    ?CEPI=  2.      ;REQUIRE EVEN PARITY ON INPUT
09
10                        ; NOTE THAT THE FOLLOWING CHARACTERISTIC MAKES ?CEPI,?CS
11                        ; MEANINGLESS (PARITY IS DONE IN THE HARDWARE)
12                        ; ALSO NON-ALM (NON-ULM) LINES CANNOT USE THIS CHARACTER
13
14        000003 .DUSR    ?C8BT=  3.      ; FULL BYTE ASCII CHARACTERS (8 BIT)
15
16        000004 .DUSR    ?CSPO=  4.      ;SET PARITY ON OUTPUT (EVEN ONLY)
17        000005 .DUSR    ?CRAF=  5.      ;SEND RUBOUTS AFTER FORM FEEDS
18        000006 .DUSR    ?CRAT=  6.      ;SEND RUBOUTS AFTER TABS
19        000007 .DUSR    ?CRAC=  7.      ;SEND RUBOUTS AFTER CR AND NL
20        000010 .DUSR    ?CNAS=  8.      ;NON ANSI STANDARD DEVICE
21        000011 .DUSR    ?COTT=  9.      ;CONVERT ESC CHARACTER (FOR OLD TTY'S)
22        000012 .DUSR    ?CEOL=  10.     ;DO NOT AUTO CR/LF AT END OF LINE
23        000013 .DUSR    ?CUCO=  11.     ;OUTPUT UPPER CASE ONLY DEVICE
24        000014 .DUSR    ?CLT=   12.     ;LEADER ON OPEN, TRAILER ON CLOSE
25        000015 .DUSR    ?CFF=   13.     ;FORM FEED ON OPEN
26                ; THE FOLLOWING TWO BITS MUST NOT BE MOVED :
27        000016 .DUSR    ?CEB0=  14.     ;ECHO MODE BIT 0
28        000017 .DUSR    ?CEB1=  15.     ;ECHO MODE BIT 1
29
30
31                ;       IN      PIBC2
32
33        000000 .DUSR    ?CULC=  0.      ;INPUT UPPER/LOWER CASE DEVICE
34        000001 .DUSR    ?CPM=   1.      ;DEVICE IS IN PAGE MODE
35        000002 .DUSR    ?CNRM=  2.      ;DISABLE MESSAGE RECEPTION
36        000003 .DUSR    ?CMOD=  3.      ;DEVICE ON MODEM INTERFACE
37                ; THE FOLLOWING FOUR BITS MUST NOT BE MOVED :
38        000004 .DUSR    ?CDT0=  4.      ;DEVICE TYPE BIT 0
39        000005 .DUSR    ?CDT1=  5.      ;DEVICE TYPE BIT 1
40        000006 .DUSR    ?CDT2=  6.      ;DEVICE TYPE BIT 2
41        000007 .DUSR    ?CDT3=  7.      ;DEVICE TYPE BIT 3
42        000010 .DUSR    ?CTO=   8.      ;DEVICE TIME-OUTS ENABLED
43        000011 .DUSR    ?CTSP=  9.      ;CRA- NO TRAILING BLANK SUPPRESSION
44        000012 .DUSR    ?CPBN=  10.     ;CRA- PACKED FORMATE ON BINARY READ
45        000013 .DUSR    ?CESC=  11.     ;ESC CHARACTER PRODUCES INTERRUPT
46        000014 .DUSR    ?CWRP=  12.     ;HARDWARE WRAPS AROUND ON LINE TOO LONG
47        000015 .DUSR    ?CFKT=  13.     ;FUNCTION KEYS ARE INPUT DELIMITERS
48        000016 .DUSR    ?CNNL=  14.     ;CRA- NO <NL> AT END OF CARD IMAGE
49                ;               15.     ;(RESERVED FOR FUTURE USE) (EXTEND PACKE
50
51
52
53                ;       ECHO MODES :
54                ;       0=      NO ECHO
55                ;       1=      STRAIGHT ECHO
56                ;       2=      ECHO CONTROL CHARS AS ^B ^F (ETC.), ESC AS $
57                ;       3=      (RESERVED FOR FUTURE USE)
58
59        000001 .DUSR    ?CEOS=  1B(?CEB1)       ;STRAIGHT ECHO BIT MASK
60        000002 .DUSR    ?CEOC=  1B(?CEB0)       ;CNTRL SPECIAL ECHO BIT MASK
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0014  PARU
01
02
03                ;       DEVICE TYPES : (FOR RUBOUT ECHO & CURSOR CONTROLS)
04                ;
05                ;       PIBC2   CHARACTERS TO :
06                ;       DEVICE  MODEL   MOVE    MOVE    ERASE   RUBOUT
07                ;       TYPE :  # :     LEFT:   RIGHT:  LINE:   ECHO:
08                ;
09                ;       0       4010A   (NONE)  (NONE)  (NONE)  SHIFT O
10                ;       0       6040    (NONE)  (NONE)  (NONE)  SHIFT O
11                ;       1       4010I   ^Z      ^Y      ^K      ^Z,SPACE,^Z
12                ;       2       6012    ^Y      ^X      ^K      ^Y,SPACE,^Y
13                ;       3       6052    ^Y      ^X      ^K      ^Y,SPACE,^Y
14                ;       4       ----    ESC,D   ESC,C   ESC,K   ESC,D,SPACE,ESC,
15                ;       5-15.   (FOR FUTURE EXPANSION)          SHIFT O
16
17
18                ;       MISC CONSTANTS
19
20        000025 .DUSR    ?LDEL=  "U-100  ; LINE DELETE CHARACTER
21        000004 .DUSR    ?EOFC=  "D-100  ; END-OF-FILE CHARACTER
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0015  PARU
01
02
03                ; ACCESS CONTROL SPECIFICATIONS
04
05                ; ACCESS PRIVILEGE BIT POSITIONS
06
07          000013 .DUSR    ?FAOB=  11.       ; OWNER ACCESS
08          000014 .DUSR    ?FAWB=  12.       ; WRITE ACCESS
09          000015 .DUSR    ?FAAB=  13.       ; APPEND ACCESS
10          000016 .DUSR    ?FARB=  14.       ; READ ACCESS
11          000017 .DUSR    ?FAEB=  15.       ; EXECUTE ACCESS
12
13                ; ACCESS PRIVILEGE MASKS
14
15          000020 .DUSR    ?FACO=  1B(?FAOB)         ; OWNER
16          000010 .DUSR    ?FACW=  1B(?FAWB)         ; WRITE
17          000004 .DUSR    ?FACA=  1B(?FAAB)         ; APPEND
18          000002 .DUSR    ?FACR=  1B(?FARB)         ; READ
19          000001 .DUSR    ?FACE=  1B(?FAEB)         ; EXECUTE
20
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0016  PARU
01
02
03              ; LOGICAL RECORD FORMAT TYPES
04
05        000001 .DUSR    ?ORDY=  1               ;DYNAMIC
06        000002 .DUSR    ?ORDS=  2               ;DATA SENSITIVE
07        000003 .DUSR    ?ORFX=  3               ;FIXED LENGTH
08        000004 .DUSR    ?ORVR=  4               ;VARIABLE LENGTH
09        000005 .DUSR    ?ORUN=  5               ;UNDEFINED
10        000006 .DUSR    ?ORVB=  6       ;IBM VARIABLE BLOCK - VARIABLE RECORD
11
12
13
14
15              ; ?GOPEN PACKET
16
17        000000 .DUSR    ?OPFL=  0       ;FLAGS IN TO ?GOPEN
18        000000 .DUSR    ?OPCH=  ?OPFL   ;CHANNEL NUMBER
19        000001 .DUSR    ?OPTY=  1       ;FORMAT (LH) AND FILE TYPE (RH)
20        000002 .DUSR    ?OPFC=  2       ;FILE CONTROL PARAMETERS
21        000003 .DUSR    ?OPEH=  3       ;FILE EOF (HIGH)
22        000004 .DUSR    ?OPEL=  4       ;FILE EOF (LOW)
23
24        000002 .DUSR    ?OPPH=  2       ;PORT NUMBER (HIGH)
25        000003 .DUSR    ?OPPL=  3       ;PORT NUMBER (LOW)
26
27        000005 .DUSR    ?OPLT=  ?OPEL+1 ;LENGTH OF PACKET
28
29
30              ; FLAGS IN BIT POINTERS
31
32        000000 .DUSR    ?OPBE=  0.              ;EXCLUSIVE OPEN
33        000001 .DUSR    ?OPDF=  1               ;INHIBIT INITIAL FORM FEED
34        000002 .DUSR    ?OPD0=  2.              ;DENSITY BITS.
35        000003 .DUSR    ?OPD1=  3.              ;
36        000004 .DUSR    ?OPD2=  4.              ;
37        000005 .DUSR    ?OPPC=  5.              ;PARITY CONTROL.
38        000006 .DUSR    ?OPBP=  6.              ;PRIVATE OPEN
39
40              ; FLAGS IN MASKS
41
42        100000 .DUSR    ?OPME=  1B(?OPBE)       ;EXCLUSIVE OPEN
43        040000 .DUSR    ?OPMD=  1B(?OPDF)       ;INHIBIT INITIAL FORM FEED
44        004000 .DUSR    ?OPDL=  1B(?OPD2)       ;800 DENSITY MODE.
45        010000 .DUSR    ?OPDM=  2B(?OPD2)       ;1600 DENSITY MODE.
46        014000 .DUSR    ?OPAM=  3B(?OPD2)       ;AUTO DENSITY MATCHING MODE.
47        002000 .DUSR    ?OPEP=  1B(?OPPC)       ;EVEN PARITY FOR 7 TRACK DRIVES.
48        001000 .DUSR    ?OPMP=  1B(?OPBP)       ;PRIVATE OPEN
49
50              ;       GENERIC FILE TYPE INDICES (IF TYPE = ?FGFN)
51
52        000001 .DUSR    ?GIN=1                  ;GENERIC INPUT
53        000002 .DUSR    ?GOUT=2                 ;GENERIC OUTPUT
54        000003 .DUSR    ?GLST=3                 ;GENERIC LIST
55        000004 .DUSR    ?GDATA=4                ;GENERIC DATA
56        000005 .DUSR    ?GNULL=5                ;NULL
57
58
59
60              ;       CHANNEL ASSIGNMENTS
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0017  PARU
01
02        000000 .DUSR   ?LOCHN=0                ;LOWEST ASSIGNABLE CHANNEL NO.
03        000077 .DUSR   ?HICHN=77               ;HIGHEST ACTIVE (REAL) CHANNEL N
04        000100 .DUSR   ?NULL=?HICHN+1          ;NULL CHANNEL ASSIGNMENT
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10018  PARU
01                ;
02                ; SYSTEM RECORD I/O PACKET FOR ALL DISK AND MAG. TAPE
03                ; AND MCA REQUESTS FROM EITHER THE GHOST OR USER CONTEXTS.
04                ;       USED FOR ?RDB/?WRB
05
06        000000 .DUSR    ?PSTI=  0       ;RECORD COUNT (RIGHT), STATUS IN (LEFT)
07        000001 .DUSR    ?PSTO=  ?PSTI+1 ;RESERVED (LEFT) PRIORITY (RIGHT)
08        000002 .DUSR    ?PCAD=  ?PSTO+1 ;WORD ADDRESS FOR DATA
09        000003 .DUSR    ?PRES=  ?PCAD+1 ;RESERVED WORD
10        000004 .DUSR    ?PRNH=  ?PRES+1 ;RECORD NUMBER (HIGH)
11                                        ;LINK # (MCA)
12        000005 .DUSR    ?PRNL=  ?PRNH+1 ;RECORD NUMBER (LOW)
13                                        ;RETRY COUNT (MCA)
14        000006 .DUSR    ?PRCL=  ?PRNL+1 ;MAX LENGTH OF EACH RECORD (MAG TAPE)
15                                        ;BYTE COUNT IN LAST BLOCK (DISK WRITES)
16                                        ;BYTE COUNT (MCA)
17
18        000007 .DUSR    ?PBLT=  ?PRCL+1 ;PACKET SIZE
19
20
21                ; PHYSICAL I/O EXTENSION TO RDB/WRB PACKET
22
23        000007 .DUSR    ?PRBB=  ?PRCL+1         ;RELATIVE BLOCK # OF BAD BLOCK
24        000010 .DUSR    ?PCS1=  ?PRBB+1         ;CONTROLLER STATUS WORD 1
25        000011 .DUSR    ?PCS2=  ?PCS1+1         ;  "    "     "  "   "   2
26        000012 .DUSR    ?PCS3=  ?PCS2+1
27        000013 .DUSR    ?PCS4=  ?PCS3+1
28        000014 .DUSR    ?PCS5=  ?PCS4+1
29        000015 .DUSR    ?PCS6=  ?PCS5+1
30        000016 .DUSR    ?PCS7=  ?PCS6+1
31        000017 .DUSR    ?PCS8=  ?PCS7+1
32
33        000020 .DUSR    ?PPBL=  ?PCS8+1         ;PACKET SIZE
34
35
36
37                ; STATUS IN
38
39        000000 .DUSR    ?IDIO=  0               ;MCA PROTOCOL
40        000001 .DUSR    ?ENOR=  1               ;ENABLE VFU LOAD / OVERRIDE LEOT
41        000002 .DUSR    ?SAFE=  2               ;THIS IS A SAFE WRITE.
42
43                ; STATUS IN MASKS
44
45        100000 .DUSR    ?IMIO=  1B?IDIO         ; SUSPEND MCA PROTOCOL
46        040000 .DUSR    ?ENOV=  1B?ENOR         ; ENABLE VFU / OVERRIDE LEOT
47        020000 .DUSR    ?SAFM=  1B?SAFE         ; SAFE WRITE MASK.
48
49                ; BIT DISPLACEMENTS
50
51        000000 .DUSR    ?DIO=   ?PSTI*16.+?IDIO ;SUSPEND MCA PROTOCOL (SET)
52        000001 .DUSR    ?ENVR=  ?PSTI*16.+?ENOR ;ENABLE VFU / OVERRIDE LEOT
53        000002 .DUSR    ?SAFB=  ?PSTI*16.+?SAFE ;SAFE WRITE BIT DISPLACEMENT
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10019  PARU
01                ;
02                ; GENERAL USER I/O PACKET
03                ;
04                ;       USED FOR ?OPEN/?READ/?WRITE/?CLOSE
05                ;
06
07        000000 .DUSR    ?ICH=   0       ;CHANNEL NUMBER
08        000001 .DUSR    ?ISTI=  ?ICH+1  ;STATUS WORD (IN)
09        000002 .DUSR    ?ISTO=  ?ISTI+1 ;RIGHT=FILE TYPE, B7=SHARED BIT, B6-B0=R
10        000003 .DUSR    ?IBAD=  ?ISTO+1 ;BYTE POINTER TO BUFFER
11        000004 .DUSR    ?IRES=  ?IBAD+1 ;RESERVED
12        000004 .DUSR    ?IFLG=  ?IRES   ;WORD OF FLAGS
13        000005 .DUSR    ?IRCL=  ?IRES+1 ;RECORD LENGTH
14        000006 .DUSR    ?IRLR=  ?IRCL+1 ;RECORD LENGTH (RETURNED)
15        000007 .DUSR    ?IRNH=  ?IRLR+1 ;RECORD NUMBER (HIGH)
16        000010 .DUSR    ?IRNL=  ?IRNH+1 ;RECORD NUMBER (LOW)
17        000011 .DUSR    ?IFNP=  ?IRNL+1 ;BYTE POINTER TO FILE NAME
18        000012 .DUSR    ?IMRS=  ?IFNP+1 ;PHYSICAL RECORD SIZE - 1 (BYTES)
19        000013 .DUSR    ?IDEL=  ?IMRS+1 ;DELIMITER TABLE ADDRESS
20
21        000014 .DUSR    ?IBLT=  ?IDEL+1 ;PACKET LENGTH
22
23                ;?ISTI BIT DEFINITIONS
24
25        000000 .DUSR    ?IPLB=  0       ;PACKET LENGTH BIT (0 => SHORT PACKET)
26        000001 .DUSR    ?ICFB=  1       ;CHANGE FORMAT BIT (0 => DEFAULT)
27        000001 .DUSR    ?ICDM=  1       ;DUMP MODE BIT (ON ?CLOSE ONLY)
28        000002 .DUSR    ?IPTB=  2       ;POSITIONING TYPE (0 => RELATIVE)
29        000003 .DUSR    ?IBIB=  3       ;BINARY I/O
30        000004 .DUSR    ?IFOB=  4       ;FORCE OUTPUT
31        000005 .DUSR    ?IOEX=  5       ;EXCLUSIVE OPEN
32        000006 .DUSR    ?IIPS=  6       ;IPC NO WAIT BIT
33        000007 .DUSR    ?PDLM=  7       ;PRIORITY REQUEST
34        000010 .DUSR    ?APBT=  8.      ;OPEN FILE FOR APPENDING
35        000011 .DUSR    ?OF1B=  9.      ;OPEN TYPE BIT 1
36        000012 .DUSR    ?OF2B=  10.     ;OPEN TYPE BIT 2
37        000013 .DUSR    ?OPIB=  11.     ;OPEN FOR INPUT
38        000014 .DUSR    ?OPOB=  12.     ;OPEN FOR OUTPUT
39        000015 .DUSR    ?RF1B=  13.     ;RECORD FORMAT BIT 1
40        000016 .DUSR    ?RF2B=  14.     ;RECORD FORMAT BIT 2
41        000017 .DUSR    ?RF3B=  15.     ;RECORD FORMAT BIT 3
42
43                ;?ISTO BIT DEFINITIONS
44
45        000007 .DUSR    ?SHAP=  7       ;SHARED BIT
46
47                ;?IRES BIT DEFINITIONS
48
49        100000 .DUSR    ?OIBM=  1B0.    ;OPEN AS AN IBM LABELLED TAPE
50        040000 .DUSR    ?OANS=  1B1.    ;OPEN AS AN ANSI LABELLED TAPE
51        000002 .DUSR    ?IDD0=  ?OPD0   ;RESERVED FOR DENSITY SETTING.
52        000003 .DUSR    ?IDD1=  ?OPD1
53        000004 .DUSR    ?IDD2=  ?OPD2
54        000005 .DUSR    ?IBEP=  ?OPPC   ;EVEN PARITY
55        000006 .DUSR    ?IBPO=  ?OPBP   ;PRIVATE OPEN
56
57                ;?ISTI BIT MASKS
58
59        100000 .DUSR    ?IPKL=  1B?IPLB ;EXTENDED PACKET (IF SET)
60        040000 .DUSR    ?ICRF=  1B?ICFB ;CHANGE RECORD FORMAT (IF SET)
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0020  PARU
01        040000 .DUSR    ?CDMP=   1B?ICDM ;SET DUMP BIT (ONLY ON ?CLOSE)
02        020000 .DUSR    ?IPST=   1B?IPTB ;RECORD POSITIONING TYPE (1 = ABSOLUTE)
03        010000 .DUSR    ?IBIN=   1B?IBIB ;BINARY I/O
04        004000 .DUSR    ?IFOP=   1B?IFOB ;FORCE OUTPUT
05        002000 .DUSR    ?IEXO=   1B?IOEX ;EXCLUSIVE OPEN
06        001000 .DUSR    ?IIPC=   1B?IIPS ;IPC NO WAIT BIT
07        000400 .DUSR    ?PDEL=   1B?PDLM ;PRIORITY OPEN-I/O
08        000200 .DUSR    ?APND=   1B?APBT ;OPEN FILE FOR APPENDING
09        000100 .DUSR    ?OFCR=   1B?OF1B ;ATTEMPT CREATE BEFORE OPEN
10        000040 .DUSR    ?OFCE=   1B?OF2B ;CORRECT ERROR ON CREATE OR OPEN
11        000020 .DUSR    ?OFIN=   1B?OPIB ;OPEN FOR INPUT
12        000010 .DUSR    ?OFOT=   1B?OPOB ;OPEN FOR OUTPUT
13        000030 .DUSR    ?OFIO=   ?OFIN+?OFOT      ;OPEN FOR INPUT AND OUTPUT
14
15                ;?ISTO BIT MASKS
16
17        000400 .DUSR    ?SHOP=   1B?SHAP          ;SHARED OPEN
18
19                ;?IRES BIT MASKS
20
21        014000 .DUSR    ?IDAM=   ?OPAM            ;DENSITY AUTOMATICLY SELECTED
22        004000 .DUSR    ?ID8=    ?OPDL            ;DENSITY 800 BPI
23        010000 .DUSR    ?ID16=   ?OPDM            ;DENSITY 1600 BPI
24        002000 .DUSR    ?IMEP=   ?OPEP            ;EVEN PARITY
25        001000 .DUSR    ?IMPO=   ?OPMP            ;PRIVATE OPEN
26
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10021  PARU
01        000007 .DUSR   ?RFBS=  1B?RF1B+1B?RF2B+1B?RF3B ;RECORD FORMAT MASK
02
03               ; RECORD FORMAT FIELD DEFINITIONS
04
05        000001 .DUSR   ?RTDY=  (?ORDY)B?RF3B           ;DYNAMIC
06        000002 .DUSR   ?RTDS=  (?ORDS)B?RF3B           ;DATA SENSITIVE
07        000003 .DUSR   ?RTFX=  (?ORFX)B?RF3B           ;FIXED LENGTH
08        000004 .DUSR   ?RTVR=  (?ORVR)B?RF3B           ;VARIABLE LENGTH
09        000005 .DUSR   ?RTUN=  (?ORUN)B?RF3B           ;UNDEFINED
10        000006 .DUSR   ?RTVB=  (?ORVB)B?RF3B           ;IBM VARIABLE BLOCK VARI
11
12               ;I/O EXTENSION PARAMETERS
13
14               ;I/O PACKET OFFSETS FOR EXTENSION PACKET ADDRESSES
15
16        000014 .DUSR   ?ETSP=  ?IBLT           ;SCREEN MANAGEMENT PACKET
17        000015 .DUSR   ?ETFT=  ?IBLT+1         ;SELECTED FIELD TRANSLATION PACK
18        000016 .DUSR   ?ETLT=  ?IBLT+2         ;LABELED TAPE PACKET
19        000017 .DUSR   ?ENET=  ?IBLT+3         ;NETWORK I/O PACKET
20
21        000004 .DUSR   ?ETSZ=  ?ENET-?IBLT+1
22
23        000034 .DUSR   ?ETMX=  ?IBLT+16.       ;MAXIMUM I/O PACKET SIZE
24
25               ;SCREEN MANAGEMENT PACKET OFFSETS
26
27        000000 .DUSR   ?ESFC=  0               ;STATUS/FUNCTION
28        000001 .DUSR   ?ESEP=  1               ;EDIT POSITION
29        000002 .DUSR   ?ESCR=  2               ;<COLUMN><ROW> FOR CURSOR POS.
30
31               ;STATUS/FUNCTION BITS
32
33        100000 .DUSR   ?ESSE=  1B0             ;SCREEN EDIT
34        040000 .DUSR   ?ESRD=  1B1             ;REDISPLAY
35        020000 .DUSR   ?ESNR=  1B2             ;DROP TYPE-AHEAD
36        010000 .DUSR   ?ESED=  1B3             ;NO ECHO FOR DEIMITERS
37        004000 .DUSR   ?ESCP=  1B4             ;CURSOR POSITIONING
38        002000 .DUSR   ?ESDD=  1B5             ;READ ENDED IN DOUBLE DELIMITER
39        001000 .DUSR   ?ESRP=  1B6             ;RETURN CURSOR POSITION AFTER I/
40        000400 .DUSR   ?ESNE=  1B7             ;NO ECHO ON INPUT
41
42               ;SELECTED FIELD TRANSLATION PACKET OFFSETS
43
44        000000 .DUSR   ?EFNF=  0               ;RIGHT BYTE - NUMBER OF FIELDS
45                                               ;LEFT BYTE - RESERVED
46               ;THE FOLLOWING THREE WORD PACKET IS DUPLICATED FOLLOWING ?EFNF
47               ;FOR EACH FIELD TO BE TRANSLATED
48
49        000000 .DUSR   ?EFFP=  0               ;BYTE OFFSET TO FIELD
50        000001 .DUSR   ?EFLN=  1               ;FIELD LENGTH
51        000002 .DUSR   ?EFTY=  2               ;TRANSLATION TYPE (SEE BELOW)
52        000003 .DUSR   ?EFMAX= ?EFTY+1         ;# OF ENTRIES IN PACKET - MUST
53                                               ;BE UPDATED IF PACKET SIZE CHANG
54
55               ;TRANSLATION TYPES FOR SELECTED TRANSLATION
56
57        000000 .DUSR   ?ASEB=  0               ;ASCII TO EBCDIC
58        000001 .DUSR   ?EBAS=  1               ;EBCDIC TO ASCII
59        000002 .DUSR   ?ALAU=  2               ;ASCII LOWER TO ASCII UPPER
60        000003 .DUSR   ?AUAL=  3               ;ASCII UPPER TO ASCII LOWER
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0022  PARU
01        000004 .DUSR   ?SPAR=  4                ;STRIP PARITY
02        000005 .DUSR   ?AEPR=  5                ;ADD EVEN PARITY
03        000006 .DUSR   ?AOPR=  6                ;ADD ODD PARITY
04        000007 .DUSR   ?TRMAX= ?AOPR+1          ;NUMBER OF TRANSLATION TYPES
05
06               ;CODES 7-17 ARE RESERVED.  ANY VALUE > 17 IS ASSUMED TO BE THE
07               ;ADDRESS OF A USER SUPPLIED TRANSLATION TABLE.
08
09               ;LABELED TAPE EXTENSION PACKET OFFSETS
10
11        000000 .DUSR   ?ELVL=  0                ;VOLUME IDENTIFIER
12        000003 .DUSR   ?ELGN=  3                ;GENERATION NUMBER
13        000004 .DUSR   ?ELVR=  4                ;VERSION NUMBER
14        000005 .DUSR   ?ELCR=  5                ;CREATION DATE
15        000006 .DUSR   ?ELRE=  6                ;RETENTION PERIOD
16        000007 .DUSR   ?ELCT=  7                ;LEFT BYTE - NUMBER OF USER TRAI
17                                                ;1B0 - LONG EXTENDED PACKET
18                                                ;RIGHT BYTE - NUMBER OF USER HEA
19        000010 .DUSR   ?ELUH=  10               ;BYTE ADDRESS OF USER HEADER LAB
20        000011 .DUSR   ?ELUT=  11               ;BYTE ADDRESS OF USER TRAILER LA
21        000012 .DUSR   ?ELAC=  ?ELUT+1          ;RIGHT BYTE = ACCESS CODE
22                                                ;LEFT BYTE = LEVEL OF LABELLING
23        000013 .DUSR   ?ELFS=  ?ELAC+1          ;BYTE PTR TO FILE SET IDENTIFER
24        000014 .DUSR   ?ELR1=  ?ELFS+1          ;RESERVED
25        000015 .DUSR   ?ELR2=  ?ELR1+1          ;RESERVED
26        000016 .DUSR   ?ELLN=  ?ELR2+1          ;LENGTH OF LABEL PACKET
27
28               ;MASKS TO BE USED IN LABELLED TAPE EXTENSION
29
30        100000 .DUSR   ?ELEP=  1B0.             ;LONG (EXTENDED) EXTENDED PACKET
31        000400 .DUSR   ?ELL1=  1B7.             ;LEVEL 1
32        001000 .DUSR   ?ELL2=  2B7.             ;LEVEL 2
33        001400 .DUSR   ?ELL3=  3B7.             ;LEVEL 3
34
35
36               ; PACKET FOR FILE TRUNCATION (?GTRUNC) SYSTEM CALL
37
38        000000 .DUSR   ?TEFW=  0                ;RESERVED
39        000001 .DUSR   ?TEFH=  ?TEFW+1          ;RESERVED
40        000002 .DUSR   ?TEFM=  ?TEFH+1          ;HIGH ORDER WORD OF BYTE COUNT
41        000003 .DUSR   ?TEFL=  ?TEFM+1          ;LOW ORDER WORD OF BYTE COUNT
42        000004 .DUSR   ?TLTH=  ?TEFL+1          ;PACKET LENGTH
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10023  PARU
01
02                ;
03                ; PACKET FOR DIRECTORY ENTRY CREATION (?CREATF)
04                ;
05
06
07        000000 .DUSR    ?CFTYP= 0               ;ENTRY TYPE (RH) AND RECORD FORM
08        000001 .DUSR    ?CTIM=  1               ;POINTER TO TIME BLOCK
09        000002 .DUSR    ?CACP=  2               ;POINTER TO INITIAL ACL
10        000003 .DUSR    ?CPOR=  3               ;PORT NUMBER (IPC TYPES ONLY)
11        000003 .DUSR    ?CHFS=  3               ;HASH FRAME SIZE (DIRECTORY TYPE
12        000003 .DUSR    ?CHID=  3               ; HOST ID (?FREM TYPE FILES ONLY
13        000003 .DUSR    ?CCPS=  3               ;FILE CONTROL PARAMETER (OTHERS)
14        000004 .DUSR    ?CLAU=  4               ;RESERVED
15        000005 .DUSR    ?CMSH=  5               ;MAX SPACE ALLOCATED (?FCPD)
16        000006 .DUSR    ?CMSL=  6               ;MAX SPACE ALLOCATED (LOW)
17        000005 .DUSR    ?CDEH=  5               ;RESERVED
18        000006 .DUSR    ?CDEL=  6               ;FILE ELEMENT SIZE
19        000007 .DUSR    ?CMIL=  7               ;MAXIMUM INDEX LEVEL DEPTH
20
21        000010 .DUSR    ?CLTH=  ?CMIL+1         ;LENGTH OF THE PARAMETER BLOCK
22
23                ;
24                ; TIME BLOCK
25                ;
26
27        000000 .DUSR    ?TCTH=  0               ;FILE CREATION TIME (HIGH)
28        000001 .DUSR    ?TCTL=  1               ;FILE CREATION TIME (LOW)
29        000002 .DUSR    ?TATH=  2               ;TIME LAST ACCESSED (HIGH)
30        000003 .DUSR    ?TATL=  3               ;TIME LAST ACCESSED (LOW)
31        000004 .DUSR    ?TMTH=  4               ;TIME LAST MODIFIED (HIGH)
32        000005 .DUSR    ?TMTL=  5               ;TIME LAST MODIFIED (LOW)
33
34        000006 .DUSR    ?TBLT=  ?TMTL+1         ;LENGTH OF TIME BLOCK
35
36                ;
37                ; LENGTH OF A USER DATA AREA
38                ;
39        000200 .DUSR    ?LNUD=  128.            ;LENGTH OF UDA
40
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10024  PARU
01                ;
02                ; PACKET FOR PROCESS CREATION (?PROC)
03                ;       WARNING: DON'T CHANGE SEQUENCE ?PPCR THROUGH
04                ;                ?PDFP WITHOUT CHANGING ?PROC CODE
05                ;                IN GHOST
06
07        000000 .DUSR    ?PFLG=  0       ;FLAGS (SEE BELOW)
08        000001 .DUSR    ?PSNM=  1       ;PROGRAM FILE BYTE POINTER
09        000002 .DUSR    ?PIPC=  2       ;MESSAGE HEADER POINTER
10        000003 .DUSR    ?PNM=   3       ;PROCESS NAME BYTE POINTER
11        000004 .DUSR    ?PMEM=  4       ;MAXIMUM SIZE IN BLOCKS
12        000005 .DUSR    ?PPRI=  5       ;PRIORITY
13        000006 .DUSR    ?PDIR=  6       ;DEFAULT DIRECTORY BYTE POINTER
14        000007 .DUSR    ?PCON=  7       ;CONSOLE DEVICE
15        000010 .DUSR    ?PCAL=  10      ;MAX # MULTIPLEXED .SYSTM CALLS
16        000011 .DUSR    ?PUNM=  11      ;USER NAME BYTE POINTER
17        000012 .DUSR    ?PPRV=  12      ;PRIV BITS
18        000013 .DUSR    ?PPCR=  13      ;MAXIMUM NUMBER OF SONS OR BROTHERS THAT
19                                        ;CAN BE CREATED
20        000014 .DUSR    ?PIFP=  14      ;POINTER TO INPUT FILE
21        000015 .DUSR    ?POFP=  15      ;POINTER TO OUTPUT FILE
22        000016 .DUSR    ?PLFP=  16      ;POINTER TO LIST FILE
23        000017 .DUSR    ?PDFP=  17      ;POINTER TO DATA FILE
24        000036 .DUSR    ?SMCH=  36      ;MAX CPU TIME LIMIT (HIGH)
25        000037 .DUSR    ?SMCL=  37      ;MAX CPU TIME LIMIT (LOW)
26
27        000020 .DUSR    ?PLTH= ?PDFP+1  ;LENGTH OF PACKET
28        000040 .DUSR    ?PXLT= ?PLTH+20 ;LENGTH OF PACKET + EXTENDER
29
30                ; PROCESS CREATE FLAGS
31
32        100000 .DUSR    ?PFPP=  1B0     ;PARALLEL PROC
33        040000 .DUSR    ?PFDB=  1B1     ;ENTER AT DEBUGGER
34        020000 .DUSR    ?PFEX=  1B2     ;BLOCK ON THIS PROCESS EXECUTION
35        010000 .DUSR    ?PFPM=  1B3     ;PRIV BITS ARE A MASK OF CALLER'S BITS
36        004000 .DUSR    ?PFPX=  1B4     ;INDICATES PRESENCE OF PACKET EXTENDER
37        002000 .DUSR    ?PFDA=  1B5     ;DO NOT PASS ?DACL
38                ;               1B6      RESERVED FOR AOS/VS (OPTIONAL BRK FILE)
39        000400 .DUSR    ?PFBS=  1B7     ;BLOCK THE NEWLY CREATED PROCESS
40
41        000002 .DUSR    ?PFRS=  1B14    ;RES PROC
42        000001 .DUSR    ?PFRP=  1B15    ;RES PREEMPTIBLE PROCESS
43
44
45                ; DEFINE PROC PRIV BITS
46
47        000001 .DUSR    ?PVPC=  1B15    ;CAN CREATE AN UNLIMITED NO OF PROCESSES
48        000002 .DUSR    ?PVTY=  1B14    ;CAN CREATE PROC OF ANY TYPE
49        000004 .DUSR    ?PVSU=  1B13    ;CAN BECOME SUPERUSER
50        000010 .DUSR    ?PVPR=  1B12    ;CAN CREATE/CHANGE PRI AT ANY VALUE
51        000020 .DUSR    ?PVPP=  1B11    ;PERIPHERAL PROC PRIV
52        000040 .DUSR    ?PVEX=  1B10    ;CAN CREATE PROC AND NOT BLK
53        000100 .DUSR    ?PVUI=  1B9     ;CAN SPECIFY A NEW USER NAME ON .PROC
54        000200 .DUSR    ?PVDV=  1B8     ;CAN ACCESS DEVICES
55        000400 .DUSR    ?PVIP=  1B7     ;CAN USE IPC PRIMITIVES
56        001000 .DUSR    ?PVIF=  1B6     ;CAN BECOME A SPECIAL PROCESS
57        002000 .DUSR    ?PVSP=  1B5     ;SUPERUSER PROCESS PRIV
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0025  PARU
01
02                ;
03                ; PACKET TO OBTAIN PROCESS STATISTICS (?RUNTM)
04
05                ; **** WARNING ****
06                ; THIS PACKET'S DISPLACEMENT ORDER DUPLICATED IN ?PSTAT CALL PAR
07                ;
08
09                ;
10
11                ;       ?RUNTM
12                ;       <ERROR RTN>
13                ;       <GOOD RTN>
14
15                ;       AC2= ADDR OF PACKET TO RECEIVE THE STATISTICS
16
17        000000 .DUSR    ?GRRH=  0               ;ELAPSED TIME IN SEC (HIGH)
18        000001 .DUSR    ?GRRL=  1               ;ELAPSED TIME (LOW)
19        000002 .DUSR    ?GRCH=  2               ;CPU TIME IN MSECS (HIGH)
20        000003 .DUSR    ?GRCL=  3               ;CPU TIME (LOW)
21        000004 .DUSR    ?GRIH=  4               ;I/O USAGE (HIGH)
22        000005 .DUSR    ?GRIL=  5               ;I/O USAGE (LOW)
23        000006 .DUSR    ?GRPH=  6               ;PAGE-MSECS HIGH
24        000007 .DUSR    ?GRPL=  7               ;PAGE-MSECS LOW
25
26        000010 .DUSR    ?GRLTH= ?GRPL+1         ;PACKET SIZE
27
28
29                ; DEFINE PARAM PACKET FOR HISTOGRAM INIT
30
31
32                ;       ?IHIST
33                ;       <ERROR RTN>
34                ;       <GOOD RTN>
35
36                ; AC0= POINTER TO PROCESS HISTOGRAM
37                ; AC1= FLAG TO DESCRIBE TYPE POINTER IN AC0
38                ; AC2= PARAM PACKET ADDR
39
40        000000 .DUSR    ?HIST=  0               ;START OF HISTOGRAM RANGE
41        000001 .DUSR    ?HIEND= 1               ;END OF HISTOGRAM RANGE
42        000002 .DUSR    ?HIWDS= 2               ;WORD GROUPING FOR HISTOGRAM
43        000003 .DUSR    ?HIBUF= 3               ;ARRAY TO RECEIVE HISTOGRAM
44
45        000004 .DUSR    ?HILTH= ?HIBUF+1        ;PACKET SIZE
46
47                ; DEFINE ARRAY OFFSETS FOR HISTOGRAM
48                ; ARRAY CONSISTS OF FIXED PREFACE CONTAINING GLOBAL INFO
49                ; FOLLOWED BY THE ACTUAL HISTOGRAM ARRAY
50
51        000000 .DUSR    ?HTTH=  0               ;TOTAL INTERVALS HIGH
52        000001 .DUSR    ?HTTL=  1               ;TOTAL INTERVALS LOW
53        000002 .DUSR    ?HPRH=  2               ;INTERVALS WITHIN TARGET PROC,
54                                                ; BUT OUTSIDE RANGE (HIGH)
55        000003 .DUSR    ?HPRL=  3               ; " (LOW)
56        000004 .DUSR    ?HAPH=  4               ;INTERVALS IN ANOTHER PROC (HIGH
57        000005 .DUSR    ?HAPL=  5               ; " LOW
58        000006 .DUSR    ?HSBH=  6               ;INTERVALS WITHIN SYSTEM ,
59                                                ;BUT NOT SYSTEM IDLE LOOP (HIGH)
60        000007 .DUSR    ?HSBL=  7               ; " LOW
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0026  PARU
01        000010 .DUSR   ?HSIH=  10              ;INTERVALS IN IDLE LOOP (HIGH)
02        000011 .DUSR   ?HSIL=  11              ; " LOW
03
04        000012 .DUSR   ?HARAY= 12              ; ARRAY START
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10027  PARU
01                ; DEFINE PARAMETERS FOR EXTENDED HISTOGRAM INIT
02
03
04                ;       ?IXHIST
05                ;       <ERROR RTN>
06                ;       <GOOD RTN>
07
08                ; AC0= POINTER TO PROCESS TO HISTOGRAM
09                ; AC1= HISTOGRAM TYPE & FLAG TO DESCRIBE TYPE POINTER IN AC0
10                ; AC2= PARAM PACKET ADDR
11                ;
12                ; HISTOGRAM TYPES
13                ;
14                ;
15        000000 .DUSR    ?HTXX = 0               ;TYPE 0: RESERVED
16        000001 .DUSR    ?HTTP = 1               ;TYPE 1: TARGET, PRIMARY
17        000002 .DUSR    ?HTAP = ?HTTP+1         ;TYPE 2: ALL PIDS
18        000003 .DUSR    ?HTTG = ?HTAP+1         ;TYPE 3: RESERVED
19        000004 .DUSR    ?HTAS = ?HTTG+1         ;TYPE 4: RESERVED
20        000005 .DUSR    ?HTSY = ?HTAS+1         ;TYPE 5: RESERVED
21
22        000005 .DUSR    ?HTMX = ?HTSY           ;MAXIMUM DEFINED TYPE
23
24                ;
25                ;PACKET DESCRIPTOR
26                ;
27                ;
28
29        000000 .DUSR    ?HDAT = 0               ;ADDRESS OF DATA AREA
30        000001 .DUSR    ?HRE1 = ?HDAT+1         ;RESERVED WORD 1(MUST BE ZERO)
31        000002 .DUSR    ?HRE2 = ?HRE1+1         ;RESERVED WORD 2(MUST BE ZERO)
32                ;
33                ; PACKET OFFSETS FOR TYPE 1 HISTOGRAMS.
34                ;
35        000003 .DUSR    ?HRDC = ?HRE2+1         ;COUNT OF RANGE DESCRIPTORS
36        000004 .DUSR    ?HRD =  ?HRDC+1         ;START OF RANGE DESCRIPTORS
37
38                ;
39                ;RANGE DESCRIPTOR TEMPLATE
40                ;
41                ;
42        000000 .DUSR    ?HRST = 0               ;STARTING ADDRESS OF RANGE
43        000001 .DUSR    ?HREN = ?HRST+1         ;ENDING ADDRESS OF RANGE
44        000002 .DUSR    ?HRIS = ?HREN+1         ;INTERVAL SIZE WITHIN RANGE
45        000003 .DUSR    ?HRRE = ?HRIS+1         ;USED BY SYSTEM
46
47        000004 .DUSR    ?HRSZ = ?HRRE+1         ;RANGE DESCRIPTOR SIZE
48
49                ;
50                ;RESERVED PARAMETERS
51                ;
52                ;
53        000000 .DUSR    ?HSPC = 0               ;RESERVED
54        000001 .DUSR    ?HSVM = ?HSPC+1         ;RESERVED
55        000002 .DUSR    ?HSVL = ?HSVM+1         ;RESERVED
56        000003 .DUSR    ?HSBT = ?HSVL+1         ;RESERVED
57
58                ;
59                ;RESERVED PARAMETERS
60                ;
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0028  PARU
01                ;
02          000000 .DUSR   ?HSTY = 0                ;RESERVED
03          000001 .DUSR   ?HSR1 = 1                ;RESERVED (USED BY SYSTEM)
04
05          000002 .DUSR   ?HSOV = 2                ;RESERVED
06          000003 .DUSR   ?HSPS = 3                ;RESERVED
07          000004 .DUSR   ?HSPE = 4                ;RESERVED
08          000005 .DUSR   ?HSR5 = 5                ;(RESERVED--SHOULD BE ZERO)
09          000006 .DUSR   ?HSIS = 6                ;RESERVED
10          000007 .DUSR   ?HSR7 = 7                ;(RESERVED--SHOULD BE ZERO)
11
12          000002 .DUSR   ?HSAH = 2                ;RESERVED
13          000003 .DUSR   ?HSAL = 3                ;RESERVED
14          000004 .DUSR   ?HSMS = 4                ;RESERVED
15          000005 .DUSR   ?HSKY = 5                ;RESERVED
16          000006 .DUSR   ?HSR6 = 6                ;(RESERVED--SHOULD BE ZERO)
17                ;.DUSR   ?HSR7 = 7                ;(RESERVED--SHOULD BE ZERO)
18
19                ;.DUSR   ?HSAH = 2                ;(RESERVED--SHOULD BE -1)
20                ;.DUSR   ?HSAL = 3                ;RESERVED
21          000004 .DUSR   ?HSMN = 4                ;RESERVED
22          000005 .DUSR   ?HSMX = 5                ;RESERVED
23                ;.DUSR   ?HSIS = 6                ;RESERVED
24                ;.DUSR   ?HSR7 = 7                ;(RESERVED--SHOULD BE ZERO)
25
26                ;.DUSR   ?HSAH = 2                ;(RESERVED--SHOULD BE -1)
27                ;.DUSR   ?HSAL = 3                ;RESERVED
28          000004 .DUSR   ?HSR4 = 4                ;(RESERVED--SHOULD BE ZERO)
29                ;.DUSR   ?HSR5 = 5                ;(RESERVED--SHOULD BE ZERO)
30                ;.DUSR   ?HSR6 = 6                ;(RESERVED--SHOULD BE ZERO)
31                ;.DUSR   ?HSR7 = 7                ;(RESERVED--SHOULD BE ZERO)
32
33          000010 .DUSR   ?HSSZ = ?HSR7+1          ;RESERVED
34
35                ;
36                ;LIMITS ON NUMBER OF RANGE DESCRIPTORS
37                ;
38                ;
39          000377 .DUSR   ?HMXR = (1024.-?HRD)/?HRSZ       ;TYPE 1
40          000177 .DUSR   ?HMXS = (1024.-?HRD)/?HSSZ       ;RESERVED
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10029  PARU
01                ;
02                ;DATA BUFFER DESCRIPTORS
03                ;
04                ;
05                        ;BUFFER DESCRIPTOR FOR PRIMARY PC HISTOGRAMMING
06          000000 .DUSR  ?HTOH = 0                ;TOTAL TICKS (HIGH)
07          000001 .DUSR  ?HTOL = ?HTOH+1          ;TOTAL TICKS (LOW)
08          000002 .DUSR  ?HIDH = ?HTOL+1          ;TICKS IN IDLE LOOP (HIGH)
09          000003 .DUSR  ?HIDL = ?HIDH+1          ;TICKS IN IDLE LOOP (LOW)
10          000004 .DUSR  ?HPMH = ?HIDL+1          ;TICKS IN TARGET PRIMARY (HIGH)
11          000005 .DUSR  ?HPML = ?HPMH+1          ;TICKS IN TARGET PRIMARY (LOW)
12          000006 .DUSR  ?HGHH = ?HPML+1          ;TICKS IN TARGET GHOST (HIGH)
13          000007 .DUSR  ?HGHL = ?HGHH+1          ;TICKS IN TARGET GHOST (LOW)
14          000010 .DUSR  ?HATH = ?HGHL+1          ;TICKS IN AOS ON BEHALF OF TARGE
15          000011 .DUSR  ?HATL = ?HATH+1          ;TICKS IN AOS ON BEHALF OF TARGE
16          000012 .DUSR  ?HOPH = ?HATL+1          ;TICKS IN OTHER PROCESSES' PRIMA
17          000013 .DUSR  ?HOPL = ?HOPH+1          ;TICKS IN OTHER PROCESSES' PRIMA
18          000014 .DUSR  ?HOGH = ?HOPL+1          ;TICKS IN OTHER PROCESSES' GHOST
19          000015 .DUSR  ?HOGL = ?HOGH+1          ;TICKS IN OTHER PROCESSES' GHOST
20          000016 .DUSR  ?HAOH = ?HOGL+1          ;TICKS IN AOS ON BEHALF OF OTHER
21                                                 ;PROCESSES (HIGH)
22          000017 .DUSR  ?HAOL = ?HAOH+1          ;TICKS IN AOS ON BEHALF OF OTHER
23                                                 ;PROCESSES (LOW)
24          000020 .DUSR  ?HSYH = ?HAOL+1          ;OTHER TICKS IN AOS (HIGH)
25          000021 .DUSR  ?HSYL = ?HSYH+1          ;OTHER TICKS IN AOS (LOW)
26          000022 .DUSR  ?HRNH = ?HSYL+1          ;TICKS IN AT LEAST ONE SPECIFIED
27                                                 ;RANGE (HIGH)
28          000023 .DUSR  ?HRNL = ?HRNH+1          ;TICKS IN AT LEAST ONE SPECIFIED
29                                                 ;RANGE (LOW)
30          000024 .DUSR  ?HCNT = ?HRNL+1          ;START OF COUNTERS FOR RANGES
31                        ;BUFFER DESCRIPTOR FOR ALL PROCESS HISTOGRAMS
32                ;.DUSR  ?HTOH = 0                ;TOTAL TICKS (HIGH)
33                ;.DUSR  ?HTOL = ?HTOH+1          ;TOTAL TICKS (LOW)
34                ;.DUSR  ?HIDH = ?HTOL+1          ;TICKS IN IDLE LOOP (HIGH)
35                ;.DUSR  ?HIDL = ?HIDH+1          ;TICKS IN IDLE LOOP (LOW)
36                ;.DUSR  ?HPMH = ?HIDL+1          ;NOT USED
37                ;.DUSR  ?HPML = ?HPMH+1          ;NOT USED
38                ;.DUSR  ?HGHH = ?HPML+1          ;NOT USED
39                ;.DUSR  ?HGHL = ?HGHH+1          ;NOT USED
40                ;.DUSR  ?HATH = ?HGHL+1          ;NOT USED
41                ;.DUSR  ?HATL = ?HATH+1          ;NOT USED
42                ;.DUSR  ?HOPH = ?HATL+1          ;NOT USED
43                ;.DUSR  ?HOPL = ?HOPH+1          ;NOT USED
44                ;.DUSR  ?HOGH = ?HOPL+1          ;NOT USED
45                ;.DUSR  ?HOGL = ?HOGH+1          ;NOT USED
46                ;.DUSR  ?HAOH = ?HOGL+1          ;NOT USED
47                ;.DUSR  ?HAOL = ?HAOH+1          ;NOT USED
48                ;.DUSR  ?HSYH = ?HAOL+1          ;OTHER TICKS IN AOS (HIGH)
49                ;.DUSR  ?HSYL = ?HSYH+1          ;OTHER TICKS IN AOS (LOW)
50                ;.DUSR  ?HRNH = ?HSYL+1          ;NOT USED
51                ;.DUSR  ?HRNL = ?HRNH+1          ;NOT USED
52                ;.DUSR  ?HCNT = ?HRNL+1          ;NOT USED
53                ;                 ?HCNT+1        ;NOT USED
54                ;                 ?HCNT+2        ;NOT USED
55                ;                 ?HCNT+3        ;NOT USED
56          000030 .DUSR  ?HPID = ?HCNT+4          ;START OF COUNTERS FOR EACH PID
57
58          000630 .DUSR  ?HAPS = ?HPID+(6*64.)    ;DATA BUFFER SIZE FOR ALL PID
59                                                 ;HISTOGRAMS
60
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0030  PARU
01
02
03                          ;RESERVED PARAMETERS
04          000024 .DUSR    ?HCMH = ?HRNL+1             ;RESERVED
05          000025 .DUSR    ?HCML = ?HCMH+1             ;RESERVED
06          000026 .DUSR    ?HIWH = ?HCML+1             ;RESERVED
07          000027 .DUSR    ?HIWL = ?HIWH+1             ;RESERVED
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10031  PARU
01
02                ; USER PACKET DEFINITION FOR ?PSTAT
03                ; RETURNS INTERESTING ITEMS FROM PROCESS TABLE
04
05                ; USAGE:
06                ;       ?PSTAT
07                ;       <ERROR RETURN>
08                ;       <NORMAL RETURN>
09
10                ;       AC2 = ADDRESS OF USER PACKET TO RECEIVE INFO
11
12        000000 .DUSR    ?PSFP = 0         ;PROCESS ID OF FATHER
13        000001 .DUSR    ?PSSN = ?PSFP+1   ;PROCESS ID'S OF SONS AS BIT ARRAY
14                                          ;IF BIT I = 1, THEN A SON HAS
15                                          ;PROCESS ID OF I
16        000020 .DUSR    ?PSAL = 16.       ;LENGTH OF BIT ARRAY IN WORDS
17        000021 .DUSR    ?PSNR = ?PSSN+?PSAL;# OF PROC'S TASKS SUSPENDED ON ?IREC
18        000022 .DUSR    ?PSNS = ?PSNR+1   ;RESERVED FOR FUTURE SYSTEM USE
19        000023 .DUSR    ?PSST = ?PSNS+1   ;PROCESS STATUS WORD
20        000024 .DUSR    ?PSQF = ?PSST+1   ;PROCESS PRIORITY ENQUEUE FACTOR
21        000025 .DUSR    ?PSFL = ?PSQF+1   ;PROCESS FLAG WORD
22        000026 .DUSR    ?PSF2 = ?PSFL+1   ;2ND FLAG WORD
23        000027 .DUSR    ?PSF3 = ?PSF2+1   ;3RD FLAG WORD
24        000030 .DUSR    ?PSPR = ?PSF3+1   ;PRIORITY
25        000031 .DUSR    ?PSBK = ?PSPR+1   ;# PAGES IN PROCESS' UNSHARED AREA
26        000032 .DUSR    ?PSPS = ?PSBK+1   ;# PAGES IN PROCESS' SHARED AREA
27        000033 .DUSR    ?PSSF = ?PSPS+1   ;PAGE # OF BEGINNING OF SHARED AREA
28        000034 .DUSR    ?PSMB = ?PSSF+1   ;# PAGES IN PROCESS' SYSTEM UNSHARED ARE
29        000035 .DUSR    ?PSYS = ?PSMB+1   ;# PAGES IN PROCESS' SYSTEM SHARED AREA
30        000036 .DUSR    ?PSYT = ?PSYS+1   ;PAGE # OF BEGINNING OF SYSTEM SHARED AR
31        000037 .DUSR    ?PSSW = ?PSYT+1   ;# OF SHARED PAGES LOADED WHEN PROCESS
32                                          ;IMAGE LAST LOADED
33        000040 .DUSR    ?PSPV = ?PSSW+1   ;PROCESS'S PRIVILEGE BITS
34        000041 .DUSR    ?PSEX = ?PSPV+1   ;TIME SLICE EXPONENT FOR THE PROC
35        000042 .DUSR    ?PSF4 = ?PSEX+1   ;4TH FLAG WORD
36        000043 .DUSR    ?PSPD = ?PSF4+1   ;PROCESS ID
37        000044 .DUSR    ?PSMX = ?PSPD+1   ;MAX # OF PAGES ALLOWED, INCLUDES SYSTEM
38        000045 .DUSR    ?PSSL = ?PSMX+1   ;# OF SUB-SLICES IN TIME SLICE
39
40                ;THE FOLLOWING DUPLICATE THE ?RUNTM PARAMETER PACKET
41        000046 .DUSR    ?PSRH = ?PSSL+1   ;ELAPSED TIME IN SEC (HIGH)
42        000047 .DUSR    ?PSRL = ?PSRH+1   ;ELAPSED TIME (LOW)
43        000050 .DUSR    ?PSCH = ?PSRL+1   ;CPU TIME IN MILLISECS (HIGH)
44        000051 .DUSR    ?PSCL = ?PSCH+1   ;CPU TIME (LOW)
45        000052 .DUSR    ?PSIH = ?PSCL+1   ;I/O USAGE (HIGH)
46        000053 .DUSR    ?PSIL = ?PSIH+1   ;I/O USAGE (LOW)
47        000054 .DUSR    ?PSPH = ?PSIL+1   ;PAGE-MILLS (HIGH)
48        000055 .DUSR    ?PSPL = ?PSPH+1   ;PAGE-MILLS (LOW)
49
50        000056 .DUSR    ?PSLTH = ?PSPL+1  ;PACKET SIZE
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0032  PARU
01
02
03
04                ; FILE STATUS PACKET (?FSTAT)
05
06          000000 .DUSR    ?STYP=   0         ;RECORD FORMAT (LEFT), FILE TYPE (RIGHT)
07          000001 .DUSR    ?STIM=   1         ;-1
08          000002 .DUSR    ?SACP=   2         ;-1
09          000003 .DUSR    ?SPNH=   3         ;PORT NUMBER (HIGH) (IPC)
10          000003 .DUSR    ?SHFS=   3         ;HASH FRAME SIZE (DIRECTORIES)
11          000003 .DUSR    ?SDCU=   3         ;DEV CODE (LEFT), UNIT # (RIGHT) (UNIT)
12          000003 .DUSR    ?SCPS=   3         ;FILE CONTROL PARAMETERS (OTHERS)
13          000004 .DUSR    ?SLAU=   4         ;RESERVED
14          000004 .DUSR    ?SPNL=   4         ;PORT NUMBER (LOW) (IPC)
15          000005 .DUSR    ?SMSH=   5         ;MAX SPACE AVAILABLE (DP) (?FCPD AND
16          000006 .DUSR    ?SMSL=   6         ; ?FLDU ONLY)
17          000005 .DUSR    ?SDEH=   5         ;FILE ELEMENT SIZE (HIGH)
18          000006 .DUSR    ?SDEL=   6         ;FILE ELEMENT SIZE (LOW)
19          000007 .DUSR    ?SMIL=   7         ;MAX NUMBER OF INDEX LEVELS
20          000010 .DUSR    ?STCH=   10        ;DATE OF FILE CREATION
21          000011 .DUSR    ?STCL=   11        ;TIME OF FILE CREATION
22          000012 .DUSR    ?STAH=   12        ;DATE OF LAST FILE ACCESS
23          000013 .DUSR    ?STAL=   13        ;TIME OF LAST FILE ACCESS
24          000014 .DUSR    ?STMH=   14        ;DATE OF LAST FILE MODIFICATION
25          000015 .DUSR    ?STML=   15        ;TIME OF LAST FILE MODIFICATION
26          000016 .DUSR    ?SSTS=   16        ;FILE STATUS WORD
27          000017 .DUSR    ?SEFH=   17        ;FILE SIZE (BYTES) (HIGH)
28          000020 .DUSR    ?SEFL=   20        ;FILE SIZE (BYTES) (LOW)
29          000021 .DUSR    ?SFAH=   21        ;FIRST LOGICAL DISK ADDRESS (HIGH)
30          000022 .DUSR    ?SFAL=   22        ;FIRST LOGICAL DISK ADDRESS (LOW)
31          000023 .DUSR    ?SIDX=   23        ;CURRENT NUMBER OF INDEX LEVELS
32          000024 .DUSR    ?SOPN=   24        ;OPEN COUNT
33          000025 .DUSR    ?SCSH=   25        ;SPACE CURRENTLY ALLOCATED (DP) (?FCPD
34          000026 .DUSR    ?SCSL=   26        ; AND ?FLDU ONLY)
35          000027 .DUSR    ?SLTH=   ?SCSL+1   ;LENGTH OF PACKET
36
37
38                ;
39                ; ?SSTS BITS
40                ;
41
42          100000 .DUSR    ?FSHB=   1B0       ; FILE IS SHARED
43          040000 .DUSR    ?FMDB=   1B1       ; FILE HAS BEEN MODIFIED
44          002000 .DUSR    ?FPRM=   1B5       ; FILE IS PERMANENT
45          001000 .DUSR    ?FDLE=   1B6       ; DELETE FILE ON LAST CLOSE
46          000400 .DUSR    ?FUDA=   1B7       ; FILE HAS A USER DATA AREA
47          000002 .DUSR    ?FARA=   1B14.     ; ALL USERS HAVE READ ACCESS
48          000001 .DUSR    ?FAEA=   1B15.     ; ALL USERS HAVE EXECUTE ACCESS
49
50                ;
51                ; ?SSTS BIT POINTERS
52                ;
53
54          000340 .DUSR    ?BSHB=   ?SSTS*16.+0
55          000341 .DUSR    ?BMDB=   ?SSTS*16.+1
56          000345 .DUSR    ?BPRM=   ?SSTS*16.+5
57          000346 .DUSR    ?BDLE=   ?SSTS*16.+6
58          000347 .DUSR    ?BUDA=   ?SSTS*16.+7
59          000356 .DUSR    ?BARA=   ?SSTS*16.+14.
60          000357 .DUSR    ?BAEA=   ?SSTS*16.+15.
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0033  PARU
01
02
03                ; DIRECTORY SEARCH PACKET (FOR ?GNFN -- GET NEXT FILE NAME)
04
05        000000 .DUSR    ?NFKY = 0         ; KEY -- ZERO TO INITIALIZE, OTHERWISE
06                                          ; SET BY SYSTEM ON PREVIOUS CALL
07        000001 .DUSR    ?NFNM = 1         ; BYTE POINTER TO AREA TO RECEIVE NAME
08        000002 .DUSR    ?NFTP = 2         ; BYTE POINTER TO TEMPLATE OR -1
09
10        000003 .DUSR    ?NFLN = 3         ; LENGTH OF PACKET
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10034  PARU
01                ;
02                ; PACKET TO GET INITIAL MESSAGE (?GTMES)
03                ;
04
05
06        000000 .DUSR    ?GREQ=  0               ; REQUEST TYPE (SEE BELOW)
07        000001 .DUSR    ?GNUM=  ?GREQ+1         ; ARGUMENT NUMBER
08        000002 .DUSR    ?GSW=   ?GNUM+1         ; BYTE PTR TO POSSIBLE SWITCH
09        000003 .DUSR    ?GRES=  ?GSW+1          ; BYTE PTR TO AREA TO RECEIVE
10                                                ; SWITCH
11        000004 .DUSR    ?GTLN=  ?GRES+1         ; PACKET LENGTH
12
13                ; REQUEST TYPES (?GREQ)
14
15        000000 .DUSR    ?GMES=  0               ; GET ENTIRE MESSAGE
16        000001 .DUSR    ?GCMD=  ?GMES+1         ; GET CLI COMMAND
17        000002 .DUSR    ?GCNT=  ?GCMD+1         ; GET ARGUMENT COUNT
18        000003 .DUSR    ?GARG=  ?GCNT+1         ; GET ARGUMENT
19        000004 .DUSR    ?GTSW=  ?GARG+1         ; TEST SWITCH
20        000005 .DUSR    ?GSWS=  ?GTSW+1         ; TEST SWITCHES
21
22
23                ; FLAGS RETURNED ON ?GFLG TYPE CALLS
24
25        100000 .DUSR    ?GFCF=  1B0             ; CLI FORMAT
26
27                ; BY CONVENTION, PROGRAMS CALLABLE FROM EXEC USE BITS 1 & 2
28                ; IF ?GFCF IS 0.
29
30        040000 .DUSR    ?GFEX=  1B1             ; FROM EXEC IF ON
31
32                ;IF ?GFEX IS ON, ?GFXB GIVES JOB'S BATCH/INTERACTIVE STATUS
33
34        020000 .DUSR    ?GFXB=  1B2             ; ON=BATCH, OFF=INTERACTIVE
35
36                ; IN ADDITION, IF CLI IS INVOKED WITH ?GFCF 0, BOTH ?GFXB & ?GFE
37                ; EQUAL TO ZERO => EXECUTE COMMAND PASSED IN MESSAGE AND RETURN.
38
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10035  PARU
01                ;
02                ; TERMINATION MESSAGE FORMAT
03                ;
04                ;       BITS 0-4        RETURN FLAGS
05                ;       BITS 5-7        TERMINATION CODE
06                ;       BITS 8-15       PID TERMINATING PROCESS
07                ;
08
09                ;
10                ; MASKS TO OBTAIN VARIOUS FIELDS FROM TERM MSG
11                ;
12
13        174000 .DUSR    ?RMSK=  3784    ; MASK FOR RETURN FLAGS
14        003400 .DUSR    ?TMSK=  7B7     ; MASK FOR TERM/OBIT CODE
15        000377 .DUSR    ?PMSK=  377     ; MASK FOR PID
16
17                ;
18                ; SHIFT CODES TO MOVE FIELDS TO LOW ORDER WORD AREA
19                ;
20
21        177765 .DUSR    ?RCFA=  -11.    ; RETURN CODE FIELD ALIGNMENT
22        177771 .DUSR    ?TCFA=   -7.    ; TERM CODE FIELD ALIGNMENT
23
24                ;
25                ; TERMINATION/OBITUARY DEFINITIONS
26                ;
27
28        000000 .DUSR    ?TSELF= 0B7     ; SELF TERMINATION OCCURRED
29        000400 .DUSR    ?TRAP=  1B7     ; USER TRAP OCCURRED
30        001000 .DUSR    ?TCIN=  2B7     ; TERMINATED BY CONSOLE INTERRUPT
31        001400 .DUSR    ?TSUP=  3B7     ; TERMINATED BY SUPERIOR PROCESS
32        002000 .DUSR    ?TAOS=  4B7     ; TERMINATED BY AOS
33        002400 .DUSR    ?TBCX=  5B7     ; CUSTOMER CONNECTION BROKEN
34        003000 .DUSR    ?TCCX=  6B7     ; CUSTOMER CHAINED
35        003400 .DUSR    ?TEXT=  7B7     ; EXTENDED TERMINATION CODE
36
37
38                ;
39                ; EXTENDED CODES - CODE DEFINITIONS AND MESSAGE FORMATS
40                ;
41
42        000010 .DUSR    ?TABR=  8.      ; CUSTOMER DID ?TABT
43        000100 .DUSR    ?T32T=  100     ; RESERVED FOR AOS/VS
44        000101 .DUSR    ?TR32=  101     ; RESERVED FOR AOS/VS
45
46                ;
47                ; ?TABT MESSAGE LAYOUT
48                ;
49
50        000000 .DUSR    ?ABCD=  0               ; CODE (?TABR)
51        000001 .DUSR    ?ABPH=  1               ; GLOBAL PORT FROM IREC ENTRY
52        000002 .DUSR    ?ABPL=  2
53        000003 .DUSR    ?ABLP=  3               ; LOCAL PORT FROM IREC ENTRY
54
55        000004 .DUSR    ?ABLN=  ?ABLP-?ABCD+1   ; ABORT MESSAGE LENGTH
56
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0036  PARU
01
02              ; FLAGS FOR RETURN TO CLI (?RETURN)
03
04
05        100000 .DUSR    ?RFCF=   20B4            ; CLI FORMAT
06        020000 .DUSR    ?RFWA=    4B4            ; WARNING (SEVERITY=1)
07        040000 .DUSR    ?RFER=   10B4            ; ERROR   (SEVERITY=2)
08        060000 .DUSR    ?RFAB=   14B4            ; ABORT   (SEVERITY=3)
09        010000 .DUSR    ?RFEC=    2B4            ; ERROR CODE FLAG. IF SET,
10                                                 ; AC0 CONTAINS ERROR CODE
11
12              ;
13              ; PACKET TO GET SYSTEM INFORMATION (?SINFO)
14              ;
15
16        000000 .DUSR    ?SIRN=   0       ; SYSTEM REV, LEFT BYTE=MAJOR,RIGHT BYTE
17        000001 .DUSR    ?SIMM=   1       ; LENGTH OF PHYSICAL MEMORY (HPAGE)
18        000002 .DUSR    ?SILN=   2       ; BYTE POINTER TO RECEIVE MASTER LDU NAM
19        000003 .DUSR    ?SIID=   3       ; BYTE POINTER TO RECEIVE SYSTEM IDENTIF
20
21        000010 .DUSR    ?SIPL=   10      ; PACKET LENGTH
22
23              ;
24              ; FLAGS FOR ?SDLM CALL
25              ;
26
27        000000 .DUSR    ?SDDN=   0B0     ;AC0 = BYTE PTR TO DEVICE NAME
28        100000 .DUSR    ?SDCN=   1B0     ;AC0 = CHANNEL NUMBER
29        000200 .DUSR    ?SDTO=   1B8.    ;SET DELIM. TBL FOR OUTPUT
30        000100 .DUSR    ?SDTI=   1B9.    ;SET DELIM. TBL FOR INPUT
31        000040 .DUSR    ?SDTP=   1B10.   ;SET DELIM. TBL FOR PRIORITY READ
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0037  PARU
01                ;
02                ;       ?OPER CALL FLAGS
03                ;
04
05        100000 .DUSR    ?MFWT=  1B0             ;WAIT FOR RESPONSE IF SET
06
07
08
09                ; DEFINE USER DEVICE DCT OFFSETS
10
11        000000 .DUSR    ?UDVXM= 0               ;TCB ADDR OF IMSG WAITER
12        000001 .DUSR    ?UDVMS= 1               ;DEVICE MASK
13        000002 .DUSR    ?UDVIS= 2               ;INTERRUPT SERVICE ROUTINE ADDR
14        000003 .DUSR    ?UDVBX= 3               ;IXMT/IMSG MAILBOX
15        000004 .DUSR    ?UDDRS= 4               ;POWERFAIL RESTART ROUTINE
16
17        000000 .DUSR    ?UDBM=  0               ;BMC MAP REQUEST.
18        002000 .DUSR    ?UDDA=  1B5             ;DCH A MAP REQUEST.
19        002040 .DUSR    ?UDDB=  1B5+1B10.       ;DCH B MAP REQUEST.
20        002100 .DUSR    ?UDDC=  1B5+2B10.       ;DCH C MAP REQUEST.
21        002140 .DUSR    ?UDDD=  1B5+3B10.       ;DCH D MAP REQUEST.
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0038  PARU
01                ;
02                ;DEFINITIONS FOR ?EXEC (EXEC REQUEST)
03                ;
04
05                ;FUNCTION CODES
06
07        000001 .DUSR ?XFMUN=    1         ;MOUNT A UNIT
08        000002 .DUSR ?XFMLT=    2         ;MOUNT A LABELED TAPE
09        000003 .DUSR ?XFDUN=    3         ;DISMOUNT A UNIT OR LABELED TAPE
10                               ;4           - RESERVED
11        000005 .DUSR ?XFSUB=    5         ;SUBMIT A BATCH JOB
12        000006 .DUSR ?XFLPT=    6         ;SUBMIT A PRINT FILE
13        000007 .DUSR ?XFPTP=    7         ;SUBMIT A PAPER TAPE PUNCH FILE
14                               ;10          - RESERVED
15        000011 .DUSR ?XFPLT=    11        ;SUBMIT A PLOT FILE
16        000012 .DUSR ?XFHAM=    12        ;SUBMIT A HAMLET FILE
17        000013 .DUSR ?XFRJ8=    13        ; - RESERVED
18        000014 .DUSR ?XFFTA=    14        ;SUBMIT A FTA FILE
19        000015 .DUSR ?XFXUN=    15        ;EXTENDED MOUNT A UNIT
20        000016 .DUSR ?XFXML=    16        ;EXTENDED MOUNT A LABELED TAPE
21        000017 .DUSR ?XFHOL=    17        ;HOLD A QUEUE ENTRY
22        000020 .DUSR ?XFUNH=    20        ;UNHOLD A QUEUE ENTRY
23        000021 .DUSR ?XFCAN=    21        ;CANCEL A QUEUE ENTRY
24        000022 .DUSR ?XFSTS=    22        ;OBTAIN RELATIONSHIP TO EXEC
25                               ;23          - RESERVED
26
27                ;THE FOLLOWING FUNCTIONS ARE RESERVED FOR INTERNAL USE
28        000024 .DUSR ?XFLO=     24        ;LABELED TAPE OPEN
29        000025 .DUSR ?XFLC=     25        ;LABELED TAPE CLOSE
30        000026 .DUSR ?XFWV=     26        ;WRONG VOLUME
31        000027 .DUSR ?XFNV=     27        ;MOUNT NEXT VOLUME
32                               ;30          - RESERVED
33        000031 .DUSR ?XFSV=     31        ;MOUNT SPECIFIC VOLUME
34
35
36        000001 .DUSR ?XFMIN=    ?XFMUN    ;LOWEST  ASSIGNED FUNCTION CODE
37        000031 .DUSR ?XFMAX=    ?XFSV     ;HIGHEST ASSIGNED FUNCTION CODE
38
39
40
41
42
43                ;FIRST WORD OF PACKET IS ALWAYS THE FUNCTION
44        000000 .DUSR ?XRFNC=    0         ;FUNCTION
45
46                ;REMAINDER OF PACKET FOR ?XFMUN
47        000001 .DUSR ?XMUL=     1         ;BYTE POINTER TO LOGICAL NAME
48        000002 .DUSR ?XMUT=     2         ;BYTE POINTER TO OPERATOR TEXT
49
50                ;REMAINDER OF PACKET FOR ?XFXUN
51        000003 .DUSR ?XMUQ=     3         ;- RESERVED
52        000004 .DUSR ?XMUR=     4         ;- RESERVED
53        000005 .DUSR ?XMUF=     5         ;FLAG WORD
54        000006 .DUSR ?XMUS=     6         ;- RESERVED
55        000007 .DUSR ?XMUE=     7         ;- RESERVED
56        000010 .DUSR ?XMUX=     ?XMUE+1   ;LENGTH OF EXTENDED UNIT MOUNT PACKET
57
58                ;REMAINDER OF PACKET FOR ?XFMLT
59        000001 .DUSR ?XMLL=     ?XMUL     ;BYTE POINTER TO LOGICAL NAME
60        000002 .DUSR ?XMLT=     ?XMUT     ;BYTE POINTER TO OPERATOR TEXT
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0039  PARU
01        000003 .DUSR ?XMLV=     3        ;BYTE POINTER TO VOLIDS
02
03               ;REMAINDER OF PACKET FOR ?XFXML
04        000004 .DUSR ?XMLR=     4        ;- RESERVED
05        000005 .DUSR ?XMLF=     ?XMUF    ;FLAG WORD
06        000006 .DUSR ?XMLS=     6        ;- RESERVED
07        000007 .DUSR ?XMLE=     7        ;- RESERVED
08        000010 .DUSR ?XMLX=     ?XMLE+1  ;LENGTH OF EXTENDED LABEL MOUNT PACKET
09
10               ;MOUNT STATUS BITS (FOR RESERVED FUNCTIONS)
11        000000 .DUSR ?XMFC=     0        ;CHECK VOLID
12        000001 .DUSR ?XMFI=     1        ;IBM
13               ;      ?OPD0=    2        ;DENSITY - SAME AS
14               ;      ?OPD1=    3        ;DENSITY - ?OPEN DENSITY
15               ;      ?OPD2=    4        ;DENSITY - BITS
16        000005 .DUSR ?XMFR=     5        ;READ ONLY
17
18               ;EXTENDED MOUNT PACKET FLAG BITS (WORDS ?XMLF AND ?XMUF)
19        040000 .DUSR ?XMIBM=    1B(?XMFI)        ;IBM TAPE
20        002000 .DUSR ?XMRO=     1B(?XMFR)        ;READ ONLY
21
22               ;BIT POINTERS FOR FLAGS WORD IN EXTENDED MOUNT PACKETS
23        000121 .DUSR ?BIBM=     (?XMUF*20)+?XMFI ;IBM
24        002120 .DUSR ?BRDO=     (?XMUF*20)+?XMRO ;READ ONLY
25
26               ;REMAINDER OF PACKET FOR ?XFDUN
27        000001 .DUSR ?XDUL=     1        ;BYTE POINTER TO LOGICAL NAME
28        000002 .DUSR ?XDUT=     ?XMUT    ;BYTE POINTER TO OPERATOR TEXT (0 IF NON
29
30               ;REMAINDER OF PACKET FOR ?XFSUB THRU ?XFRJ8
31        000001 .DUSR ?XTYP=     1        ;BYTE POINTER TO QUEUE NAME
32        000002 .DUSR ?XDAT=     2        ;DATE ENQUEUED (RETURNED)
33        000003 .DUSR ?XTIM=     3        ;TIME ENQUEUED (RETURNED)
34        000004 .DUSR ?XLMT=     4        ;LIMIT (PAGES, BI-SECONDS, ETC.)
35        000005 .DUSR ?XPRI=     5        ;PRIORITY (0-377) -1 TO DEFAULT
36                                         ; (RETURNED IF DEFAULTED)
37        000006 .DUSR ?XFGS=     6        ;FLAGS
38        000000 .DUSR   ?XFSH=   0        ;HOLD THIS SEQUENCE NUMBER
39                               ;1          - RESERVED
40                               ;2          - RESERVED
41                               ;3          - RESERVED
42                               ;4          - RESERVED
43        000005 .DUSR   ?XFDA=   5        ;DELETE AFTER PROCESSING
44        000006 .DUSR   ?XFNR=   6        ;NOT RESTARTABLE ON SYSTEM/EXEC FAILURE
45                               ;7          - RESERVED
46                               ;8.         - RESERVED
47                               ;9.         - RESERVED
48        000012 .DUSR   ?XFBI=   10.      ;OUTPUT IS IN BINARY
49        000013 .DUSR   ?XFOP=   11.      ;REQUIRES OPERATOR ON DUTY
50        000014 .DUSR   ?XFNO=   12.      ;NOTIFY SUBMITTOR WHEN DONE
51        000015 .DUSR   ?XFRA=   13.      ;INTERPRET ?XAFD AND ?XAFT
52        000016 .DUSR   ?XFTI=   14.      ;TITLES OPTION
53        000017 .DUSR   ?XFFO=   15.      ;FOLDING OPTION
54        000007 .DUSR ?XSEQ=     7        ;SEQUENCE NUMBER (RETURNED)
55        000010 .DUSR ?XRES=     10       ; - RESERVED
56        000011 .DUSR ?XFBP=     11       ;BP JOBNAME IF BATCH, FORMS NAME IF OUTP
57        000012 .DUSR ?XPBP=     12       ;BP PATHNAME OF FILE (MUST BE COMPLETE)
58        000013 .DUSR ?XAFD=     13       ;AFTER DATE (0 IF NONE)
59        000014 .DUSR ?XAFT=     14       ;AFTER TIME (0 IF NONE)
60        000015 .DUSR ?XXW0=     15       ;EXTENDER WORD 0
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0040  PARU
01          000016 .DUSR ?XXW1=     16       ;EXTENDER WORD 1
02          000017 .DUSR ?XXW2=     17       ;EXTENDER WORD 2
03          000020 .DUSR ?XXW3=     20       ;EXTENDER WORD 3
04          000021 .DUSR ?XLTH=     21       ;LENGTH OF PACKET
05
06                 ;PACKET OFFSETS FOR ?XFHOL THRU ?XFSTS
07          000001 .DUSR    ?XFP1=  1        ;FIRST PARAMETER
08          000002 .DUSR    ?XFP2=  2        ;SECOND PARAMETER
09          000003 .DUSR    ?XFP3=  3        ;THIRD PARAMETER
10
11                 ;STATUS BIT DEFINITIONS (FOR RESERVED FUNCTIONS)
12          000060 .DUSR ?XSCV=     ?XFP3*20+?XMFC  ;VALIDATE VOLID
13          000061 .DUSR ?XSIBM=    ?XFP3*20+?XMFI  ;IBM TAPE
14          000062 .DUSR ?XSDEN=    ?XFP3*20+?OPD0  ;DENSITY BITS
15          000063 .DUSR ?XSD1=     ?XFP3*20+?OPD1  ;DENSITY BITS
16          000064 .DUSR ?XSD2=     ?XFP3*20+?OPD2  ;DENSITY BITS
17          000065 .DUSR ?XSRO=     ?XFP3*20+?XMFR  ;READONLY
18
19                 ;BIT POINTERS FOR ?XFGS
20          000140 .DUSR    ?BFSH=  ?XFGS*20+?XFSH  ;HOLD SEQUENCE NUMBER
21          000145 .DUSR    ?BFDA=  ?XFGS*20+?XFDA  ;DELETE AFTER PROCESSING
22          000146 .DUSR    ?BFNR=  ?XFGS*20+?XFNR  ;NOT RESTARTABLE
23          000152 .DUSR    ?BFBI=  ?XFGS*20+?XFBI  ;BINARY
24          000153 .DUSR    ?BFOP=  ?XFGS*20+?XFOP  ;OPERATOR
25          000154 .DUSR    ?BFNO=  ?XFGS*20+?XFNO  ;NOTIFY
26          000155 .DUSR    ?BFRA=  ?XFGS*20+?XFRA  ;AFTER
27          000156 .DUSR    ?BFTI=  ?XFGS*20+?XFTI  ;TITLES
28          000157 .DUSR    ?BFFO=  ?XFGS*20+?XFFO  ;FOLD
29
30                 ;BIT MASKS FOR ?XFGS
31          100000 .DUSR    ?BMSH=  1B(?XFSH)       ;HOLD SEQUENCE NUMBER
32          002000 .DUSR    ?BMDA=  1B(?XFDA)       ;DELETE AFTER PROCESSING
33          001000 .DUSR    ?BMNR=  1B(?XFNR)       ;NOT RESTARTABLE
34          000040 .DUSR    ?BMBI=  1B(?XFBI)       ;BINARY
35          000020 .DUSR    ?BMOP=  1B(?XFOP)       ;OPERATOR
36          000010 .DUSR    ?BMNO=  1B(?XFNO)       ;NOTIFY
37          000004 .DUSR    ?BMRA=  1B(?XFRA)       ;AFTER
38          000002 .DUSR    ?BMTI=  1B(?XFTI)       ;TITLES
39          000001 .DUSR    ?BMFO=  1B(?XFFO)       ;FOLD
40
41                 ;FTA TRANSFER OPTIONS FLAGS (IN WORD ?XXW2)
42          000000 .DUSR    ?X2CM=  0  ;COMPRESSION REQUESTED
43          000001 .DUSR    ?X2RC=  1  ;PROCESS ONLY IF SOURCE MORE RECENT
44          000002 .DUSR    ?X2AP=  2  ;APPEND SOURCE TO DESTINATION IF IT EXISTS
45          000003 .DUSR    ?X2SD=  3  ;DELETE THE SOURCE AFTER TRANSFER
46          000004 .DUSR    ?X2DD=  4  ;DELETE THE DESTINATION BEFORE TRANSFER
47          000005 .DUSR    ?X2RM=  5  ;USE RECORD MODE TRANSFER
48          000006 .DUSR    ?X2TO=  6  ;LET CONNECTION TIMEOUT AFTER TRANSFER
49          000007 .DUSR    ?X2CP=  7  ;RESTART AT LAST CHECKPOINT AFTER FAILURE
50
51                 ;BIT POINTERS FOR ?XXW2 FTA FLAGS
52          000360 .DUSR    ?B2CM=  ?XXW2*20+?X2CM  ;COMPRESS
53          000361 .DUSR    ?B2RC=  ?XXW2*20+?X2RC  ;RECENT
54          000362 .DUSR    ?B2AP=  ?XXW2*20+?X2AP  ;APPEND
55          000363 .DUSR    ?B2SD=  ?XXW2*20+?X2SD  ;DELETE SOURCE AFTER TRANSFER
56          000364 .DUSR    ?B2DD=  ?XXW2*20+?X2DD  ;DELETE DESTINATION BEFORE TRANS
57          000365 .DUSR    ?B2RM=  ?XXW2*20+?X2RM  ;RECORD MODE
58          000366 .DUSR    ?B2TO=  ?XXW2*20+?X2TO  ;TIMEOUT
59          000367 .DUSR    ?B2CP=  ?XXW2*20+?X2CP  ;RESTART AT CHECK POINT
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10041  PARU
01
02                ;
03                ; INTERPROCESS COMMUNICATION SYSTEM (IPC) PARAMETERS
04                ;
05
06                ; HIGHEST LEGAL LOCAL PORT NUMBER
07
08        000177 .DUSR    ?IMPRT= 127.      ;MAX LEGAL USER LOCAL PORT #
09        000377 .DUSR    ?MXLPN= 255.      ;MAX LEGAL LOCAL PORT #
10
11
12                ; IPC MESSAGE HEADER
13
14        000000 .DUSR    ?ISFL=  0         ;SYSTEM FLAGS
15        000001 .DUSR    ?IUFL=  1         ;USER FLAGS
16
17                ; PORT NUMBERS FOR ?ISEND
18        000002 .DUSR    ?IDPH=  2         ;DESTINATION PORT NUMBER (HIGH)
19        000003 .DUSR    ?IDPL=  3         ;DESTINATION PORT NUMBER (LOW)
20        000004 .DUSR    ?IOPN=  4         ;ORIGIN PORT NUMBER
21
22                ; PORT NUMBERS FOR ?IREC
23        000002 .DUSR    ?IOPH=  2         ;ORIGIN PORT NUMBER (HIGH)
24        000003 .DUSR    ?IOPL=  3         ;ORIGIN PORT NUMBER (LOW)
25        000004 .DUSR    ?IDPN=  4         ;DESTINATION PORT NUMBER
26
27        000005 .DUSR    ?ILTH=  5         ;LENGTH OF MESSAGE OR BUFFER (IN WORDS)
28        000006 .DUSR    ?IPTR=  6         ;POINTER TO MESSAGE/BUFFER
29
30        000007 .DUSR    ?IPLTH= ?IPTR+1 ;LENGTH OF HEADER
31
32        000007 .DUSR    ?IRLT=  7         ;?IS.R RECEIVE BUFFER LENGTH
33        000010 .DUSR    ?IRPT=  10        ;?IS.R RECEIVE BUFFER POINTER
34
35        000011 .DUSR    ?IPRLTH= ?IRPT+1 ; LENGTH OF ?IS.R HEADER
36
37                ; SYSTEM FLAG BIT MASKS
38
39        100000 .DUSR    ?IFSTM= 1B0       ;SEND TO SELF
40        100000 .DUSR    ?IFRFM= 1B0       ;RECEIVE FROM SELF
41        040000 .DUSR    ?IFSOV= 1B1       ;=1 => SPOOL MESSAGE IF BUFFER TOO SMALL
42        020000 .DUSR    ?IFNBK= 1B2       ;=1 => RETURN ERROR IF NO WAITING MESSAG
43        010000 .DUSR    ?IFNSP= 1B3       ;=1 => DO NOT SPOOL MESSAGE
44
45                ;SYSTEM FLAG BIT POINTERS
46
47        000000 .DUSR    ?IBSTM= ?ISFL*16.+0
48        000000 .DUSR    ?IBRFM= ?ISFL*16.+0
49        000001 .DUSR    ?IBSOV= ?ISFL*16.+1
50        000002 .DUSR    ?IBNBK= ?ISFL*16.+2
51        000003 .DUSR    ?IBNSP= ?ISFL*16.+3
52
53                ;
54                ; SYSTEM PORT NUMBERS
55                ;
56
57        000010 .DUSR    ?SPTM=  10        ;PORT FOR TRAP MESSAGES TO FATHER
58        000011 .DUSR    ?SPRM=  11        ;PORT FOR RUNTIME MESSAGES TO FATHER
59        000200 .DUSR    ?SIGP=  ?IMPRT+1 ;DEST. PORT FOR INITIAL GHOST MESSAGE
60        000012 .DUSR    ?SPPT=  12        ;PORT FOR TERMINATION OF INFOS ASSOCIATE
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0042  PARU
01        000013 .DUSR   ?SRIM=  13      ;PORT FOR SYSTEM/REMA INTERFACE MESSAGES
02
03
04               ;
05               ; MOVE BYTES PACKET DEFINITION
06               ;
07        000000 .DUSR   ?MBOA=  0       ; OTHER PROCESS BYTE ADDRESS
08        000001 .DUSR   ?MBMA=  1       ; CALLING PROCESS BYTE ADDRESS
09        000002 .DUSR   ?MBPD=  2       ; TARGET PID (1B0->GHOST)
10        000003 .DUSR   ?MBNB=  3       ; NUMBER OF BYTES TO MOVE
11
12        000004 .DUSR   ?MBLN=  ?MBNB-?MBOA+1   ; MOVE BYTES PACKET LENGTH
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0043  PARU
01
02                ; USER STATUS TABLE (UST) TEMPLATE
03
04        000400 .DUSR    UST=     400      ; START OF USER STATUS AREA
05
06        000000 .DUSR    USTEZ=   0        ; EXTENDED VARIABLE  WORD COUNT
07        000001 .DUSR    USTES=   1        ; EXTENDED VARIABLE PAGE 0 START
08        000002 .DUSR    USTSS=   2        ; SYMBOLS START
09        000003 .DUSR    USTSE=   3        ; SYMBOLS END
10        000004 .DUSR    USTS1=   4        ; SYSTEM WORD
11        000005 .DUSR    USTS2=   5        ; SYSTEM WORD
12        000006 .DUSR    USTDA=   6        ; DEB ADDR OR -1
13        000007 .DUSR    USTFL=   7        ; FLAG WORD (SEE DEFINITION BELOW)
14        000010 .DUSR    USTSL=   10       ; SHARED LIBRARY LIST POINTER
15        000011 .DUSR    USTIT=   11       ; INTERRUPT ADDRESS
16        000012 .DUSR    USTRV=   12       ; REVISION OF PROGRAM
17        000013 .DUSR    USTTC=   13       ; NUMBER OF TASKS (1 TO 32.)
18        000014 .DUSR    USTCT=   14       ; CURRENTLY ACTIVE TCB
19        000015 .DUSR    USTAC=   15       ; START OF ACTIVE TCB CHAIN
20        000016 .DUSR    USTFC=   16       ; START OF FREE TCB CHAIN
21        000017 .DUSR    USTBL=   17       ; # IMPURE BLKS
22        000020 .DUSR    USTOD=   20       ; OVLY DIRECTORY ADDR
23        000021 .DUSR    USTST=   21       ; SHARED STARTING BLK #
24        000022 .DUSR    USTSZ=   22       ; SHARED SIZE IN BLKS
25
26        000022 .DUSR    USTEN=   USTSZ    ; LAST ENTRY
27
28
29                ; REDEFINITIONS FOR SYSTEM USAGE
30        000006 .DUSR    USTGB=   USTDA    ;GHOST BKPT ENTRY ADDRESS
31        000002 .DUSR    USTGC=   USTSS    ;GHOST CLEAN UP ADDRESS FOR TERMS
32        000012 .DUSR    USTGD=   USTRV    ;GHOST DISPATCHER ADDR
33
34
35
36                ; UST FLAGS
37
38        100000 .DUSR    ?UFIN=   1B0      ;GHOST IS INITIALIZED AND CAN BE ENTERED
39        040000 .DUSR    ?UFIP=   1B1      ;GHOST INIT IS IN PROGRESS- MUST WAIT FO
40        020000 .DUSR    ?UFDR=   1B2      ;INHIBIT SCHEDULING
41        010000 .DUSR    ?UFDB=   1B3      ;PROCESS IS BEING DEBUGGED
42        004000 .DUSR    ?UFID=   1B4      ;PROCESS IS IN DEBUGGER
43        002000 .DUSR    ?UFPH=   1B5      ;PRIMARY TASK RUNTIME HOLD ON SCHEDULING
44        001000 .DUSR    ?UFHP=   1B6      ;HOLD PRIMARY TASKS - RUN GHOST ONLY
45        000400 .DUSR    ?UFED=   1B7      ;AN EXTENDED CONTEXT HAS BEEN DEFINED
46        000200 .DUSR    ?UFHD=   1B8      ;HOLD PRIMARY TASK FOR PMGR
47
48                ; BIT POINTERS FOR USTFL
49
50        000160 .DUSR    ?BUIN=   USTFL*16.+0.
51        000161 .DUSR    ?BUIP=   USTFL*16.+1.
52        000162 .DUSR    ?BUDR=   USTFL*16.+2.
53        000163 .DUSR    ?BUDB=   USTFL*16.+3
54        000164 .DUSR    ?BUID=   USTFL*16.+4
55        000165 .DUSR    ?BUPH=   USTFL*16.+5.
56        000166 .DUSR    ?BUHP=   USTFL*16.+6.
57        000167 .DUSR    ?BUED=   USTFL*16.+7.
58        000170 .DUSR    ?BUHD=   USTFL*16.+8.
59
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10044  PAPU
01
02                ; TASK CONTROL BLOCK (TCB) TEMPLATE
03                ; ?TSP THROUGH ?TCUD MUST BE KEPT IN ORDER AND CONTIGUOUS
04                ; ?TSP THROUGH ?TPC ARE ORDERED FOR RESTORE INSTRUCTION
05
06
07        000000 .DUSR    ?TLNK=  0         ;LINK (NOTE= MUST BE OFFSET 0 !!!)
08        000001 .DUSR    ?TSTAT= 1         ;STATUS BITS
09        000002 .DUSR    ?TSP=   2         ;STACK POINTER
10        000003 .DUSR    ?TFP=   3         ;FRAME POINTER
11        000004 .DUSR    ?TSL=   4         ;STACK LIMIT
12        000005 .DUSR    ?TSO=   5         ;FAULT HANDLER
13        000006 .DUSR    ?TAC0=  6         ;AC0
14        000007 .DUSR    ?TAC1=  7         ;AC1
15        000010 .DUSR    ?TAC2=  10        ;AC2
16        000011 .DUSR    ?TAC3=  11        ;AC3
17        000012 .DUSR    ?TPC=   12        ;PC AND CARRY
18        000013 .DUSR    ?TUSP=  13        ;USP
19        000014 .DUSR    ?TELN=  14        ;TCB EXTENTION ADDR
20        000015 .DUSR    ?TFPS=  15        ;FPU SAVE AREA POINTER
21        000016 .DUSR    ?TCUD=  16        ;CURRENT DESCRIPTOR
22        000017 .DUSR    ?TSYS=  17        ;SYSTEM CALL WORD
23        000020 .DUSR    ?TIDPR= 20        ;ID (LEFT BYTE) PRIORITY (RIGHT BYTE)
24        000021 .DUSR    ?TSLK=  21        ;SYSTEM CALL LINK
25        000022 .DUSR    ?TKAD=  22        ;KILL POST PROCESSING ROUTINE
26        000023 .DUSR    ?TGEX=  23        ;GHOST EXTENSION ADDRESS
27
28        000024 .DUSR    ?TLN=   ?TGEX-?TLNK+1    ;LENGTH OF TCB
29        000016 .DUSR    ?TGXL=  ?TSYS-?TSP+1     ;LENGTH OF TCB EXTENSION
30
31                ; TASK STATUS BITS
32
33        100000 .DUSR    ?TSPN=  1B0       ; TASK PENDED
34        040000 .DUSR    ?TSSG=  1B1       ; WAITING FOR OVERLAY AREA OR .XMTW/.REC
35        020000 .DUSR    ?TSSP=  1B2       ; SUSPENDED
36        010000 .DUSR    ?TSRC=  1B3       ; WAITING FOR TRCON
37        004000 .DUSR    ?TSIG=  1B4       ; IN GHOST CONTEXT
38        002000 .DUSR    ?TSIW=  1B5       ; WAITING FOR GHOST INIT TO COMPLETE
39        001000 .DUSR    ?TSGS=  1B6       ; GHOST WAITING FOR SYNCH.
40        000400 .DUSR    ?TSAB=  1B7       ; PENDED AWAITING ?GABORT
41        000200 .DUSR    ?TSUF=  1B8       ; PEND BIT AVAILABLE DIRECTLY TO USERS
42        000100 .DUSR    ?TSYN=  1B9       ; SYNCHRONOUS I/O COMPLETION BIT
43        000040 .DUSR    ?TSUN=  1B10      ;TUS HAS BEEN INITIALIZED
44        000020 .DUSR    ?TSSH=  1B11      ; TSH HAS BEEN INITIALIZED
45        000010 .DUSR    ?TSXR=  1B12      ; TASK PENDED ON XMT OR REC
46        000004 .DUSR    ?TSWI=  1B13      ; TASK AWAITING ?IWKUP
47
48                ; TASK STATUS BIT POINTERS
49
50        000020 .DUSR    ?BTPN= ?TSTAT*16.+0
51        000021 .DUSR    ?BTSG= ?TSTAT*16.+1
52        000022 .DUSR    ?BTSP= ?TSTAT*16.+2
53        000023 .DUSR    ?BTRC= ?TSTAT*16.+3
54        000024 .DUSR    ?BTIG= ?TSTAT*16.+4
55        000025 .DUSR    ?BTIW= ?TSTAT*16.+5
56        000026 .DUSR    ?BTGS= ?TSTAT*16.+6
57        000027 .DUSR    ?BTAB= ?TSTAT*16.+7
58        000030 .DUSR    ?BTUF= ?TSTAT*16.+8.
59        000031 .DUSR    ?BSYN= ?TSTAT*16.+9.
60        000032 .DUSR    ?BSUN= ?TSTAT*16.+10.
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0045  PARU
01        000033 .DUSR    ?BSSH= ?TSTAT*16.+11.
02        000034 .DUSR    ?BSXR= ?TSTAT*16.+12.
03        000035 .DUSR    ?BSWI= ?TSTAT*16.+13.
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10046  PARU
01                ;MACROS FOR USER RUNTIME MANAGEMENT
02
03                .MACRO  ?EURT   ;ENTER A USER RUNTIME ROUTINE
04                **      INC     3,3             ;ASSUME SUCCESS RETURN
05                ** .DO  "^1"=="
06                **      SAVE    0               ;NO STACK TEMPORARIES
07                ** .ENDC
08                ** .DO  "^1"<>"
09                **      SAVE    ^1              ;RESERVE STACK TEMPORARIES
10                ** .ENDC
11                **      ?SURTM
12                %
13
14                .MACRO  ?SURTM  ;SET USER RUNTIME MODE
15                ;**********************
16                ;SET USER RUNTIME MODE
17                ;**********************
18                **      .EXTD   ?URTB
19                **      LDA     3,?URTB         ;BIT POINTER
20                **      BTO     3,3             ;SET BIT
21                **      LDA     3,FP            ;LOAD FRAME POINTER
22                %
23
24                .MACRO  ?RSAVE          ;RESOURCE CALL STACK ALLOCATION
25                **                      ;FOR ?DESC AND ?VRTN
26                **      SAVE    2+^1    ;ARGUMENT = USER'S TEMPORARIES
27                %
28
29          000003 .DUSR  ?TMP=   3       ;RESOURCE CALL TEMP.'S BEGIN HERE
30
31                ;PACKET FOR TASK DEFINITION (?TASK)
32
33          000000 .DUSR  ?DLNK=  0       ;NON-ZERO = SHORT PACKET, ZERO = EXTENDE
34          000001 .DUSR  ?DPRI=  1       ;PRIORITY, ZERO TO USE CALLER'S
35          000002 .DUSR  ?DID=   2       ;I.D., ZERO FOR NONE
36          000003 .DUSR  ?DPC=   3       ;STARTING ADDRESS OR RESOURCE ENTRY
37          000004 .DUSR  ?DAC2=  4       ;INITIAL AC2 CONTENTS
38          000005 .DUSR  ?DSTB=  5       ;STACK BASE, MINUS ONE FOR NO STACK
39          000006 .DUSR  ?DSSZ=  6       ;STACK SIZE, IGNORED IF NO STACK
40          000007 .DUSR  ?DSFLT= 7       ;STACK FAULT ROUTINE ADDR OR -1 IF SAME
41          000010 .DUSR  ?DFLGS= 10      ;FLAGS
42          100000 .DUSR  ?DFL0=  180     ;RESERVED FOR SYSTEM
43          040000 .DUSR  ?DFLRC= 181     ;RESOURCE CALL TASK
44          000001 .DUSR  ?DFL15= 1815    ;RESERVED FOR SYSTEM
45          000011 .DUSR  ?DRES=  11      ;RESERVED FOR SYSTEM
46          000012 .DUSR  ?DNUM=  12      ;NUMBER OF TASKS TO CREATE
47          000013 .DUSR  ?DSLTH= ?DNUM+1 ;LENGTH OF SHORT PACKET
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0047  PARU
01
02              ;PACKET ENTENSION FOR TASK QUEUEING
03
04        000013 .DUSR   ?DSH=    13       ;STARTING HOUR, -1 IF IMMEDIATE
05        000014 .DUSR   ?DSMS=   14       ;STARTING SECOND IN HOUR, IGNORED IF IMM
06        000015 .DUSR   ?DCC=    15       ;NUMBER OF TIMES TO CREATE TASK(S)
07        000016 .DUSR   ?DCI=    16       ;CREATION INCREMENT  IN SECONDS
08        000017 .DUSR   ?DXLTH=  ?DCI+1   ;LENGTH OF EXTENDED PACKET
09
10
11              ;BIT POINTER TO TASK DEF BITS
12
13        000201 .DUSR   ?DFBRC=  ?DFLGS*16.+1     ;RESOURCE CALL
14
15
16              ;PACKET FOR DEFINING THE EXTENDED CONTEXTS FOR A PROCESS
17
18        000000 .DUSR   ?ECSUN=  0        ;TUS START (PAGE #)
19        000001 .DUSR   ?ECZUN=  1        ;TUS SIZE (# PAGES)
20        000002 .DUSR   ?ECSSH=  2        ;TSH START (PAGE #)
21        000003 .DUSR   ?ECZSH=  3        ;TSH SIZE (# PAGES)
22        000004 .DUSR   ?ECLTH=  ?ECZSH+1         ;LENGTH OF PACKET
23
24
25              ; SCL,SVC TRAP CODES
26
27        000000 .DUSR   ?SCEG=   0                ;ENTER GHOST
28        000001 .DUSR   ?SCRM=   1                ;REMAP A SLOT
29        000002 .DUSR   ?SCXS=   2                ;EXIT PRIMARY SCHED MODE
30        000003 .DUSR   ?SCST=   3                ;SCHEDULE A TASK
31        000004 .DUSR   ?SCGX=   4                ;GHOST EXIT
32        000005 .DUSR   ?SCSM=   5                ;SET MASK
33        000006 .DUSR   ?SCLE=   6                ;LEF ENABLE
34        000007 .DUSR   ?SCLD=   7                ;LEF DISABLE
35        000010 .DUSR   ?SCLS=   10               ;LEF STATUS
36        000011 .DUSR   ?SCAC=   11               ;ADDR CHECK
37        000012 .DUSR   ?SCGS=   12               ;GHOST RESCHEDULE
38        000013 .DUSR   ?SCET=   13               ;ENTER TASK FROM GHOST SCHED MOD
39
40
41              ; DEFINE IMPLICIT SCL CALLS (SCL AC FIELDS IMPLY FUNCTION)
42
43        107510 .DUSR   ?SCLB=   107510           ;SCL 0,1 = DEBUGGER BREAKPOINT
44        113510 .DUSR   ?SCLG=   113510           ;SCL 0,2 = GHOST BREAKPOINT
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0048  PARU
01                ;USER OVERLAY DATA BASE
02
03        177777 .DUSR    ?NDNUM= -1        ;NUMBER OF NODES
04
05                                ;PER NODE
06        000000 .DUSR    ?NDARS= 0         ;RIGHT BYTE - NUMBER OF AREAS IN NODE
07                                          ;LEFT BYTE  - NODE NUMBER
08        000001 .DUSR    ?NDOVS= 1         ;NUMBER OF OVERLAYS FOR NODE
09        000002 .DUSR    ?NDFHI= 2         ;FILE BLOCK HIGH FOR FIRST OVERLAY
10        000003 .DUSR    ?NDFLO= 3         ;FILE BLOCK LOW FOR FIRST OVERLAY
11        000004 .DUSR    ?NDASZ= 4         ;SIZE OF EACH OVERLAY (256. WORD BLOCKS)
12                                          ; (1B0 MEANS SHARED)
13        000005 .DUSR    ?NDFAR= 5         ;DISPLACEMENT OF FIRST AREA DESCRIPTOR
14
15                                ;PER AREA IN NODE
16        000000 .DUSR    ?ARNOD= 0         ;ADDRESS OF NODE DESCRIPTOR
17        000001 .DUSR    ?ARBAS= 1         ;BASE ADDRESS OF AREA
18        000002 .DUSR    ?AROUC= 2         ;AREA USAGE (077700 MEANS EMPTY)
19                                          ;BIT 0 - LOADING
20                                          ;  1-9 - OVERLAY #
21                                          ;10-15 - USE COUNT
22        077700 .DUSR    ?AREPY= 077700    ;EMPTY AREA AND OVERLAY NUM. MASK
23        000003 .DUSR    ?ARNXT= 3         ;DISPL. TO NEXT AREA DESCR. IN NODE
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10049  PARU
01
02
03              ;USER SHARED LIBRARY DATA BASE
04
05              ;TABLE POINTED TO BY USTSL
06
07                                  ;FOR EACH LIBRARY 0->63.
08        000000 .DUSR    ?SLLDS= 0       ;LIBRARY DESCRIPTORS (NULL=0)
09                                  ;FOR EACH 1K BLOCK OF MEMORY
10        000100 .DUSR    ?SLMEM= 64.     ;MEMORY TABLE (BLOCK 0 -> 31.)
11                                          ;       >0      FREE WITH BLOCK DESCRIPT
12                                          ;       0       FREE
13                                          ;       -1      TAKEN
14                                          ;       1B0 + ADDR OF BLOCK DESC CHAIN
15                                          ;               (END OF TABLE)
16              ;LIBRARY DESCRIPTORS (?SLLDS)
17
18        000000 .DUSR    ?SLDCH= 0       ;CHANNEL NUMBER (UNOPENED -1)
19                                          ;               (OPENING -2)
20        000001 .DUSR    ?SLDNM= 1       ;LIBRARY NUMBER
21        000002 .DUSR    ?SLDLO=2        ;LOW EXTERNAL REFERENCE NUMBER
22        000003 .DUSR    ?SLDHI= 3       ;HIGH EXT. REF. NUMBER
23
24                                  ;FOR EACH EXT. REF. ?SLDLO->?SLDHI
25        000004 .DUSR    ?SLDEN= 4       ;EXTERNAL REF. DESCRIPTORS
26                                          ; 1B0 SHARED LIBRARY
27                                          ; 0B0 OVERLAY
28                                          ; >=?USTART ROOT
29
30              ;OVERLAY DESCRIPTORS
31
32        000000 .DUSR    ?OVEDS= 0       ;OVERLAY NODE/NUMBER
33        000001 .DUSR    ?OVEOF= 1       ;OFFSET INTO OVERLAY FOR ENTRY
34
35              ;SHARED LIBRARY ENTRY DESCRIPTIONS
36
37        000000 .DUSR    ?SLEDS= 0       ;CORRESPONDING BLOCK DESCRIPTOR
38        000001 .DUSR    ?SLEOF= 1       ;OFFSET INTO BLOCK
39
40              ;BLOCK DESCRIPTORS
41
42        000000 .DUSR    ?SLBAB= 0       ;AREA BASE (1K ADDR) (NULL 0)
43                                          ;                   (LOCKED-- 1K ADDR)
44        000001 .DUSR    ?SLBUC= 1       ;USE COUNT (NULL-- 0, LOCKED-- -1)
45        000002 .DUSR    ?SLBSZ= 2       ;AREA SIZE (# OF 1K BLOCKS)
46        000003 .DUSR    ?SLBFL= 3       ;LRU FORWARD (CIRCLE LINKED)
47                                          ;       INIT TO LIBRARY NUMBER FOR GHOST
48        000004 .DUSR    ?SLBBL= 4       ;LRU BACKWARD (CIRCLE LINKED)
49                                          ;       INIT TO NEXT BLOCK DESC FOR GHOS
50        000005 .DUSR    ?SLBHB= 5       ;FILE BLOCK ADDRESS HIGH
51        000006 .DUSR    ?SLBLB= 6       ;                   LOW
52        000007 .DUSR    ?SLBCH= 7       ;ADDRESS OF CHANNEL NUMBER WORD (?SLLCH)
53
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0050  PARU
01
02
03                ;SYNC CALL PARAMETER PACKET OFFSET DEFINITIONS
04
05                        ;ENABLE CALL PACKET ENTRIES
06        000000          .DUSR   ?SCHN=   0          ;CHANNEL NUMBER
07        000001          .DUSR   ?SSTI=   ?SCHN+1    ;INPUT LINE DESCRIPTOR
08        000002          .DUSR   ?STOC=   ?SSTI+1    ;CONNECT TIMEOUT COUNT
09        000003          .DUSR   ?SMDI=   ?STOC+1    ;MULTIDROP INFO
10        000004          .DUSR   ?SELN=   ?SMDI+1    ;ENABLE PACKET LENGTH
11
12                ;ADDITIONS TO SEBL PACKET FOR NETWORKING
13
14        000004          .DUSR   ?SWSZ=   ?SMDI+1    ; WINDOW SIZE
15        000005          .DUSR   ?SFSZ=   ?SWSZ+1    ; FRAME SIZE
16        000006          .DUSR   ?SFTM=   ?SFSZ+1    ; FRAME TIMEOUT
17        000007          .DUSR   ?SRTY=   ?SFTM+1    ; RETRY COUNT
18        000010          .DUSR   ?SPRA=   ?SRTY+1    ; PRIMARY ADDRESS
19        000011          .DUSR   ?SSDA=   ?SPRA+1    ; SECONDARY ADDRESS
20        000012          .DUSR   ?SHLN=   ?SSDA+1    ; EXTENDED ENABLE PACKET LENGTH
21
22                        ;ENABLE CALL LINE DESCRIPTOR FLAGS
23        000000          .DUSR   ?SDPP=0B0           ;POINT-TO-POINT LINE
24        100000          .DUSR   ?SDMD=1B0           ;MULTI-DROP LINE
25        000000          .DUSR   ?SDSC=0B1           ;SECONDARY/TRIBUTARY STATION
26        040000          .DUSR   ?SDPR=1B1           ;PRIMARY/CONTROL STATION
27        000000          .DUSR   ?SEBC=0B2           ;LINE CODE=EBCDIC
28        020000          .DUSR   ?SASC=1B2           ;LINE CODE=ASCII
29        000100          .DUSR   ?SAS8=1B9           ;LINE CODE=ASCII (8-BIT)
30        000000          .DUSR   ?SNPR=0B5           ;PARITY=NONE
31        002000          .DUSR   ?SOPR=1B5           ;       =ODD
32        004000          .DUSR   ?SEPR=2B5           ;       =EVEN
33        000000          .DUSR   ?SCRC=0B7           ;BCC=CRC16
34        000400          .DUSR   ?SCIT=1B7           ;    =CCITT16
35        001000          .DUSR   ?SLRC=2B7           ;    =LRC
36        000000          .DUSR   ?SBSC=0B15          ;PROTOCOL=BSC
37        000001          .DUSR   ?SHSP=1B15          ;PROTOCOL=HASP MULTILEAVING
38        000002          .DUSR   ?SSARM=2B15         ;HDLC ASYNCH RESPONSE MODE
39        000003          .DUSR   ?SSABM=3B15         ;HDLC ASYNCH BALANCED MODE
40
41                        ;SEND AND RECEIVE CALLS
42
43        000000          .DUSR   ?SSIS=0             ;I/O INPUT STATUS
44        000001          .DUSR   ?SBUP=   ?SSIS+1    ;DATA BUFFER POINTER
45        000002          .DUSR   ?SBYC=   ?SBUP+1    ;DATA BYTE COUNT
46        000003          .DUSR   ?SDAD=   ?SBYC+1    ;DEVICE ADDRESS
47        000003          .DUSR   ?SIRL=   ?SDAD      ;INTERMEDIATE RECORD LENGTH
48        000004          .DUSR   ?SBYM=   ?SIRL+1    ;MAXIMUM BUFFER SIZE
49        000005          .DUSR   ?STOV=   ?SBYM+1    ;TIMEOUT OVERRIDE
50        000006          .DUSR   ?SSNL=   ?STOV+1    ;LENGTH OF ?SSND PKT
51
52                ;ADDITIONS TO SRCV/SSND PACKET FOR NETWORKING.
53
54        000006          .DUSR   ?SEXT=   ?STOV+1    ;EXTENDED ERROR INDICATOR
55        000007          .DUSR   ?SIND=   ?SEXT+1    ;FRAME INDICATOR
56        000010          .DUSR   ?SHLT=   ?SIND+1    ;LENGTH OF EXTENDED ?SSND PACKET
57
58                        ;?SSND CONTROL FIELD FORMATS
59                        ;--------------------------
60
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0051  PARU
01        000000          .DUSR   ?NTRN=0B0        ;NON-TRANSPARENT
02        100000          .DUSR   ?TRAN=1B0        ;TRANSPARENT
03
04        000000          .DUSR   ?STXB=0B1        ;STX BLOCK
05        040000          .DUSR   ?SOHB=1B1        ;SOH BLOCK
06
07        000000          .DUSR   ?ETBB=0B2        ;ETB BLOCK
08        020000          .DUSR   ?ETXB=1B2        ;ETX BLOCK
09
10        010000          .DUSR   ?SITB=1B3        ;ITB
11        004000          .DUSR   ?SCON=1B4        ;CONVERSATIONAL MODE
12
13        000000          .DUSR   ?CINT=0B15       ;INITIAL CALL
14        000001          .DUSR   ?CONT=1B15       ;CONTINUE CALL
15        000002          .DUSR   ?SWAK=2B15       ;SEND WACK
16        000003          .DUSR   ?SEOT=3B15       ;SEND EOT
17        000004          .DUSR   ?SDET=4B15       ;SEND DLE/EOT
18        000005          .DUSR   ?STTD=5B15       ;SEND TTD
19        000006          .DUSR   ?SINT=6B15       ;SEND RVI
20        000007          .DUSR   ?SDIS=7B15       ; PERFORM LOGICAL DISCONNECT
21
22                        ;SEND CONTROL FIELD MASK
23        174000          .DUSR   ?SCMK=?TRAN!?SOHB!?ETXB!?SITB!?SCON
24
25                        ;?SRCV CONTROL FIELD FORMATS
26                        ;--------------------------
27
28        000040          .DUSR   ?SDAC=1B10.      ;DEVICE ADDRESS CHARACTERS
29        000100          .DUSR   ?SSLR=1B9.       ;SELECT ADRS RECEIVED
30        000200          .DUSR   ?SPLR=1B8.       ;POLL ADRS RECEIVED
31        000400          .DUSR   ?SACK=1B7        ;SEND ACK & RECV
32        001000          .DUSR   ?SNAK=1B6        ;SEND NAK & RECV
33        002000          .DUSR   ?SRVI=1B5        ;SEND RVI & RECV
34
35                        ;?SRCV INITIAL CALL TYPES
36        000002          .DUSR   ?SPWK=2B15       ;SEND WACK AND WAIT FOR POLL/SEL
37        000005          .DUSR   ?SPET=5B15       ;SEND EOT AND WAIT FOR POLL/SELE
38        000006          .DUSR   ?SPRV=6B15       ;SEND RVI AND WAIT FOR POLL/SELE
39        000007          .DUSR   ?SPNK=7B15       ;SEND NAK AND WAIT FOR POLL/SELE
40
41                        ;RECV CONTROL FIELD MASK
42        003400          .DUSR   ?RCMK=?SACK!?SNAK!?SRVI
43
44                        ;RECV INITIAL CALL TYPE MASK
45        000007          .DUSR   ?RTMK=?SPWK!?SPET!?SPRV!?SPNK
46
47                        ;'GET ERROR STATISTICS' CALL PACKET DEFINITIONS
48
49        000000          .DUSR   ?SNKC=0          ;TOTAL NAKS RECEIVED
50        000001          .DUSR   ?STTO=1          ;TOTAL TIMEOUTS WHILE RECEIVING
51        000002          .DUSR   ?SBER=2          ;TOTAL BCC ERRORS
52        000003          .DUSR   ?SGLN=3          ;LENGTH OF ?SGES PKT
53
54                        ;HDLC GET STATISTICS CALL PACKET DEFINITIONS
55                        ;
56                        ;  NOTE - FOR ANYONE MAKING CHANGES TO THE HDLC GES
57                        ;         CALL PACKET DEFINITIONS,  X25 REQUIRES THAT
58                        ;         THE 1ST 12 PARAMETERS BE IN THE SAME ORDER
59                        ;         AND THE SAME POSITION AS THEY WERE ORIGINALLY
60                        ;         DEFINED!!!
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0052  PARU
01
02
03
04        000000          .DUSR   ?STHI=0          ;TIME SPAN - HIGH ORDER
05        000001          .DUSR   ?STLO=1          ;TIME SPAN - LOW ORDER
06        000002          .DUSR   ?SIXH=2          ;# I FRAMES TRANSMITTED - HI ORD
07        000003          .DUSR   ?SIXL=3          ;# I FRAMES TRANSMITTED - LO ORD
08        000004          .DUSR   ?SBSH=4          ;# BYTES TRANSMITTED - HI ORDER
09        000005          .DUSR   ?SBSL=5          ;# BYTES TRANSMITTED - LO ORDER
10        000006          .DUSR   ?SIHI=6          ;# I FRAMES RECEIVED - HI ORDER
11        000007          .DUSR   ?SILO=7          ;# I FRAMES RECEIVED - LO ORDER
12        000010          .DUSR   ?SBRH=10         ;# BYTES RECEIVED - HI ORDER
13        000011          .DUSR   ?SBRL=11         ;# BYTES RECEIVED - LO ORDER
14        000012          .DUSR   ?SIRX=12         ;# I FRAMES RETRANSMITTED
15        000013          .DUSR   ?STOR=13         ;# I FRAMES TIMED OUT/RETRANSMIT
16        000014          .DUSR   ?SRNR=14         ;# RNR FRAMES RECEIVED
17        000015          .DUSR   ?SRNT=15         ;# RNR FRAMES TRANSMITTED
18        000016          .DUSR   ?SRJR=16         ;# REJ FRAMES RECEIVED
19        000017          .DUSR   ?SRJT=17         ;# REJ FRAMES TRANSMITTED
20        000020          .DUSR   ?SRSR=20         ;# LINK RESETS RECEIVED
21        000021          .DUSR   ?SRST=21         ;# LINK RESETS TRANSMITTED
22        000022          .DUSR   ?SFCS=22         ;# FRAME CHECK SEQUENCE ERRORS
23        000023          .DUSR   ?SDOR=23         ;# DATA OVERRUNS
24        000024          .DUSR   ?SGHL=24         ;LENGTH  OF PACKET
25
26                ;
27                ;       HDLC GET STATISTICS CALL FUNCTION CODES
28                ;
29                ;               NOTE: THESE CODES MUST NOT BE CHANGED WITHOUT
30                ;                     MAKING CODE CHANGES TO SHOV2.SR
31                ;
32        000000          .DUSR   ?SHRS=0          ;RESET STATISTIC ACCUMULATORS
33        000001          .DUSR   ?SHRR=1          ;READ & RESET STAT ACCUMULATORS
34        000002          .DUSR   ?SHRD=2          ;READ STATISTIC ACCUMULATORS
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0053  PARU
01                ;PARAMETERS FOR AOS FORMS CONTROL UTITLTY
02
03        000000          .DUSR   ?VFTYP  =0              ;CONTAINS "FC" FOR UDA
04                                                        ;TYPE 'FORMS CONTROL'
05        000001          .DUSR   ?VFREV  =?VFTYP+1       ;UDA FORMAT REV NO.
06        000002          .DUSR   ?VFCPL  =?VFREV+1       ;CHARS/LINE
07        000003          .DUSR   ?VFLPP  =?VFCPL+1       ;LINES/PAGE
08        000004          .DUSR   ?VFTOF  =?VFLPP+1       ;TOP OF FORM LINE #
09        000005          .DUSR   ?VFBOF  =?VFTOF+1       ;BOTTOM OF FORM LINE #
10        000006          .DUSR   ?VFTBS  =?VFBOF+1       ;TABS TABLE
11                                                        ;TABLE ENTRY
12        000011          .DUSR   ?VFWCH  =9.             ;WORDS/[TAB OR CHAN] ARE
13        000017          .DUSR   ?VF01   =?VFTBS+?VFWCH  ;CHAN 1 LINE # TBL
14
15                        ;LENGTH OF UDA USED FOR FORMS INFO
16        000174          .DUSR   ?VFLEN  =?VF01+(?VFWCH*12.)-?VFTYP+1
17
18                        ;DEFINE CURRENT UDA FORMAT REVISION NUMBER
19        000000          .DUSR   ?VFNO   =0              ;REV 0
20
21
22                ;       DEFINITION OF THE LABEL DATA BLOCK - USED IN THE ?LABEL
23                ;
24        000000          ?LBFG=  0               ;STATUS BITS
25        000001          ?LBDV=  ?LBFG+1         ;BYTE POINTER TO THE MAG TAPE UN
26        000002          ?LBVD=  ?LBDV+1         ;BYTE POINTER TO THE VOLUME IDEN
27        000003          ?LBST=  ?LBVD+1         ;LEFT BYTE - LABEL LEVEL
28                                                ;RIGHT BYTE - NUMBER OF USER VOL
29        000004          ?LBUV=  ?LBST+1         ;BYTE POINTER TO USER VOLUME LAB
30        000005          ?LBOI=  ?LBUV+1         ;BYTE POINTER TO OWNER IDENTIFIE
31        000006          ?LBAC=  ?LBOI+1         ;LEFT BYTE - RESERVED
32                                                ;RIGHT BYTE - ACCESSIBLITY
33        000007          ?LBR1=  ?LBAC+1         ;RESERVED WORD
34        000010          ?LBR2=  ?LBR1+1         ;RESERVED WORD
35        000011          ?LBLN=  ?LBR2+1         ;LENGTH OF THE PACKET
36
37                ;       DEFINE THE STATUS BITS
38                ;
39        000000  .DUSR   ?LBIB=  0               ;IBM FORMAT
40        000001  .DUSR   ?LBSC=  1               ;SCRATCH THIS TAPE
41        000002  .DUSR   ?LBB0=  ?OPD0           ;DENSITY BIT 0
42        000003  .DUSR   ?LBB1=  ?OPD1           ;DENSITY BIT 1
43        000004  .DUSR   ?LBB2=  ?OPD2           ;DENSITY BIT 2
44                ;
45                ;       DEFINE BIT MASKS
46                ;
47        100000  .DUSR   ?LBIM=  1B(?LBIB)       ;IBM FORMAT
48        040000  .DUSR   ?LBSC=  1B(?LBSB)       ;SCRATCH THIS TAPE
49        004000  .DUSR   ?LB8=   ?OPDL           ;800 BPI
50        010000  .DUSR   ?LB16=  ?OPDM           ;1600 BPI
51        014000  .DUSR   ?LBAM=  ?OPAM           ;AUTOMATIC MATCH
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
10054  PARU
01
02                ;
03                ; CONNECTION MANAGER DEFINITIONS
04                ;
05
06                ;
07                ; ?CON AC1 BIT FLAGS
08                ;
09
10        000000          .DUSR   ?CPID=  0               ; PID/PROCESS NAME FLAG
11        000001          .DUSR   ?COBIT= 1               ; OBITUARY NOTIFICATION
12
13                ;
14                ; ?CON BIT MASKS
15                ;
16        100000          .DUSR   ?MCPID= 1B(?CPID)
17        040000          .DUSR   ?MCOBIT=1B(?COBIT)
18
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0055  PARU
01
02                ; ?IWKUP PACKET DEFINITIONS
03
04        000000          .DUSR   ?IPID=   0        ; TARGET PROCESS ID
05        000002          .DUSR   ?ITCB=   ?IPID+2  ; TARGET TCB ADDRESS
06        000004          .DUSR   ?IAC0=   ?ITCB+2  ; AC0 TO PASS TO TARGET
07        000006          .DUSR   ?IAC1=   ?IAC0+2  ; AC1 TO PASS TO TARGET
08        000010          .DUSR   ?IAC2=   ?IAC1+2  ; AC2 TO PASS TO TARGET
09        000011          .DUSR   ?IERR=   ?IAC2+1  ; ERROR INDICATOR
10                                                  ;    -1 -> START TASK AT ERROR RE
11                                                  ; ELSE -> START TASK AT GOOD RET
12                        ; ?IWKUP PACKET LENGTH
13        000012          .DUSR   ?IWLN=   ?IERR-?IPID+1
14
15
16                ;
17                ; ACCESS REMOTE RESOURCE MESSAGE PACKET DEFINITION
18                ;
19        000000          .DUSR   ?RPID=   0                ; PID
20        000002          .DUSR   ?RTCB=   ?RPID+2          ; TCB ADDR
21                        ; THE FOLLOWING OFFSETS MUST MATCH IN POSITION
22                        ;  WITH THE ?GREMA PACKET OFFSETS
23        000003          .DUSR   ?RHID=   ?RTCB+1          ; HOST ID
24        000004          .DUSR   ?RSYS=   ?RHID+1          ; SYS CALL WORD
25        000006          .DUSR   ?RAC0=   ?RSYS+2          ; CONTENTS OF AC0
26        000010          .DUSR   ?RAC1=   ?RAC0+2          ; CONTENTS OF AC1
27        000012          .DUSR   ?RAC2=   ?RAC1+2          ; CONTENTS OF AC2
28        000014          .DUSR   ?RXTN=   ?RAC2+2          ; EXTENSION WORD
29        000016          .DUSR   ?RXT2=   ?RXTN+2          ; EXTENSION WORD 2
30
31        000017          .DUSR   ?RPLN=   ?RXT2-?RPID+1    ; PACKET LENGTH
32
33
34                ;
35                ; ?GREMA PACKET DEFINITION
36                ;
37                        ; THESE MUST MATCH IN POSITION WITH THE
38                        ;  ACCESS REMOTE RESOURCE PACKET DEFINITIONS
39        000000          .DUSR   ?GHID=   0                ; HOST ID
40        000001          .DUSR   ?GSYS=   ?GHID+1          ; SYS CALL WORD
41        000003          .DUSR   ?GAC0=   ?GSYS+2          ; CONTENTS OF AC0
42        000005          .DUSR   ?GAC1=   ?GAC0+2          ; CONTENTS OF AC1
43        000007          .DUSR   ?GAC2=   ?GAC1+2          ; CONTENTS OF AC2
44        000011          .DUSR   ?GXTN=   ?GAC2+2          ; EXTENSION WORD
45        000013          .DUSR   ?GXT2=   ?GXTN+2          ; EXTENSION WORD 2
46
47        000014          .DUSR   ?GRLN=   ?GXT2-?GHID+1    ; PACKET LENGTH
48
49                ;
50                ; REMOTE ACCESS PROCESS PRIVILEGE BIT MASKS
51                ;
52        000001          .DUSR   ?BMSU=   1B15             ; SUPERUSER STATUS
53        000002          .DUSR   ?BMSP=   1B14             ; SUPERPROCESS STATUS
54                ;
55                ; REMOTE ACCESS PROCESS PRIVILEGE BIT POINTERS
56                ;
57        000077          .DUSR   ?BPSU=   ?RHID*16.+15.    ; SUPERUSER
58        000076          .DUSR   ?BPSP=   ?RHID*16.+14.    ; SUPERPROCESS
59
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
!0056  PARU
**00000 TOTAL ERRORS, 00000 FIRST PASS ERRORS
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0057 PARU

ER9TP  000467          7/35     7/36
ERABT  000017          2/19
ERACL  000167          4/03
ERACU  000065          2/57
ERADR  000006          2/10
ERAPB  000444          7/12     7/13
ERAPL  000446          7/14     7/15
ERAPN  000445          7/13     7/14
ERAPP  000450          7/16     7/19
ERAPR  000447          7/15     7/16
ERARC  000062          2/54
ERARG  000141          3/41
ERARU  000064          2/56
ERASS  000162          3/58
ERAWD  000034          2/32
ERBCT  000303          5/27     5/28
ERBFO  000407          6/37     6/38
ERBIF  000346          6/03     6/04
ERBLR  000110          3/16
ERBMX  000040          2/36
ERBNK  000304          5/28     5/29
ERBOF  000101          3/09
ERBOV  000255          5/05     5/06
ERBPE  000306          5/30     5/31
ERBRT  000307          5/31     5/32
ERBSZ  000117          3/23
ERCAD  000326          5/47     5/48
ERCAH  000365          6/18     6/19
ERCAI  000337          5/56     5/57
ERCBK  000425          6/51     6/52
ERCCS  000427          6/53     6/54
ERCDE  000354          6/09     6/10
ERCDN  000431          6/57     6/58
ERCDU  000452          7/20     7/23
ERCFT  000231          4/41
ERCGF  000136          3/38
ERCIU  000113          3/19
ERCMH  000370          6/21     6/22
ERCND  000341          5/58     5/59
ERCNI  000340          5/57     5/58
ERCNV  000264          5/12     5/13
ERCOF  000464          7/32     7/33
ERCON  000042          2/38
ERCPD  000237          4/47
ERCRC  000315          5/37     5/38
ERCSE  000404          6/33     6/35
ERCTF  000355          6/10     6/11
ERCTN  000272          5/18     5/19
ERCTY  000400          6/29     6/30
ERDAD  000245          4/53
ERDAI  000063          2/55
ERDCH  000015          2/17
ERDCT  000175          4/09
ERDCU  000263          5/11     5/12
ERDDE  000023          2/23
ERDDH  000235          4/45
ERDFN  000222          4/33
ERDID  000031          2/29
ERDIE  000417          6/45     6/46
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0058 PARU

ERDIO 000251          4/58
ERDIS 000260          5/08    5/09
ERDIU 000356          6/11    6/12
ERDNM 000045          2/41
ERDPE 000155          3/53
ERDSC 000401          6/30    6/31
ERDSE 000415          6/43    6/44
ERDSL 000277          5/23    5/24
ERDTO 000076          3/06
ERDVC 000226          4/38
EREAD 000242          4/50
EREMB 000366          6/19    6/20
ERENQ 000314          5/36    5/37
ERE01 000203          4/15
ERE02 000204          4/16
EREOF 000030          2/28
EREOT 000261          5/09    5/10
EREPE 000276          5/22    5/23
EREPL 000265          5/13    5/14
ERESO 000213          4/23
ERETX 000310          5/32    5/33
EREXC 000071          3/01
ERFAD 000244          4/52
ERFCE 000422          6/48    6/49
ERFCT 000312          5/34    5/35
ERFDE 000025          2/25
ERFFT 000232          4/42
ERFIL 000075          3/05
ERFIX 000243          4/51
ERFLF 000466          7/34    7/35
ERFNO 000002          2/06
ERFOE 000420          6/46    6/47
ERFPU 000172          4/06
ERFRE 000421          6/47    6/48
ERFRM 000462          7/30    7/31
ERFSI 000331          5/50    5/51
ERFSZ 000376          6/27    6/28
ERFTL 000100          3/08
ERFTM 000433          6/59    7/02
ERFUR 000230          4/40
ERFXF 000465          7/33    7/34
ERGBE 000416          6/44    6/45
ERGEN 000333          5/52    5/53
ERGES 000112          3/18
ERGFE 000223          4/35
ERGME 000423          6/49    6/50
ERGSG 000050          2/44
ERHAE 000362          6/15    6/16
ERHID 000363          6/16    6/17
ERHIP 000455          7/25    7/26
ERHIS 000160          3/56
ERHNE 000364          6/17    6/18
ERHTT 000454          7/24    7/25
ERIAD 000246          4/54
ERIBC 000437          7/05    7/06
ERIBM 000116          3/22
ERIBS 000044          2/40
ERICB 000114          3/20
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0059 PARU

ERICL 000143          3/43
ERICM 000001          2/05
ERICN 000147          3/47
ERIDD 000127          3/31
ERIDF 000210          4/20
ERIDI 000463          7/31    7/32
ERIDP 000057          2/51
ERIDT 000132          3/34
ERIDU 000131          3/33
ERIFC 000024          2/24
ERIFN 000217          4/30
ERIFT 000104          3/12
ERIGM 000142          3/42
ERIHN 000373          6/24    6/25
ERIIP 000456          7/26    7/27
ERILB 000137          3/39
ERILC 000435          7/03    7/04
ERILD 000130          3/32
ERILL 000434          7/02    7/03
ERILN 000154          3/52
ERILO 000106          3/14
ERIMM 000275          5/21    5/22
ERINC 000412          6/40    6/41
ERIND 000247          4/55
ERINE 000316          5/38    5/39
ERINF 000411          6/39    6/40
ERINP 000171          4/05
ERIOO 000124          3/28
ERIOP 000056          2/50
ERIOT 000077          3/07
ERIPD 000205          4/17
ERIPR 000176          4/10
ERIRB 000020          2/20
ERIRL 000061          2/53
ERIRP 000372          6/23    6/24
ERIRT 000266          5/14    5/15
ERIRV 000161          3/57
ERISE 000311          5/33    5/34
ERISO 000234          4/44
ERISQ 000436          7/04    7/05
ERISV 000216          4/26
ERITC 000066          2/58
ERITD 000432          6/58    6/59
ERITF 000323          5/44    5/45
ERITP 000343          5/60    6/01
ERIVP 000053          2/47
ERKAD 000414          6/42    6/43
ERLAB 000327          5/48    5/49
ERLDC 000405          6/35    6/36
ERLFN 000221          4/32
ERLIN 000270          5/16    5/17
ERLIS 000301          5/25    5/26
ERLNA 000300          5/24    5/25
ERLNK 000207          4/19
ERLNM 000320          5/40    5/41
ERLRF 000253          5/01
ERLTK 000252          4/60
ERLTL 000067          2/59
```

Figure D-1. PARU.LS Parameter Listing (concluded

```
0060 PARU

ERLVL 000200          4/12
ERMEM 000005          2/09
ERMIL 000146          3/46
ERMIM 000206          4/18
ERMIS 000126          3/30
ERMPR 000016          2/18
ERMRD 000144          3/44
ERMRL 000342          5/59    5/60
ERMUS 000457          7/27    7/28
ERNAC 000145          3/45
ERNAD 000027          2/27
ERNAE 000026          2/26
ERNAG 000344          6/01    6/02
ERNAI 000371          6/22    6/23
ERNAK 000256          5/06    5/07
ERNAS 000353          6/08    6/09
ERNCF 000345          6/02    6/03
ERNCS 000321          5/41    5/42
ERNDC 000426          6/52    6/53
ERNDR 000072          3/02
ERNDS 000471          7/37    7/38
ERNDV 000125          3/29
ERNEF 000052          2/46
ERNFC 000374          6/25    6/26
ERNHL 000413          6/41    6/42
ERNIN 000360          6/13    6/14
ERNM4 000351          6/06    6/07
ERNMC 000035          2/33
ERNMS 000054          2/48
ERNMW 000224          4/36
ERNNA 000361          6/14    6/15
ERNNL 000254          5/02    5/05
ERNOA 000335          5/54    5/55
ERNOR 000055          2/49
ERNOT 000011          2/13
ERNPC 000453          7/23    7/24
ERNPL 000322          5/42    5/44
ERNRD 000105          3/13
ERNRR 000150          3/48
ERNSA 000073          3/03
ERNSD 000240          4/48
ERNSL 000274          5/20    5/21
ERNSW 000043          2/39
ERNUD 000225          4/37
ERNUS 000472          7/38
EROFN 000220          4/31
EROOF 000451          7/19    7/20
EROPR 000003          2/07
EROTH 000262          5/10    5/11
EROTL 000441          7/07    7/08
EROVN 000007          2/11
EROVR 000236          4/46
ERPAR 000070          2/60
ERPDF 000133          3/35
ERPET 000163          3/59
ERPLS 000271          5/17    5/18
ERPNM 000173          4/07
ERPNU 000174          4/08
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0061 PARU

ERPRE 000111          3/17
ERPRH 000107          3/15
ERPRM 000324          5/45     5/46
ERPRN 000051          2/45
ERPRO 000250          4/56
ERPRP 000037          2/35
ERPRV 000102          3/10
ERPTY 000041          2/37
ERPUF 000121          3/25
ERPWL 000122          3/26
ERQTS 000013          2/15
ERRAD 000033          2/31
ERRBO 000214          4/24
ERRCA 000410          6/38     6/39
ERRCE 000403          6/32     6/33
ERRDL 000202          4/14
ERREB 000443          7/09     7/12
ERRES 000406          6/36     6/37
ERREV 000336          5/55     5/56
ERRFM 000140          3/40
ERRIE 000402          6/31     6/32
ERRMP 000047          2/43
ERROO 000201          4/13
ERRRR 000367          6/20     6/21
ERRST 000227          4/39
ERRVI 000267          5/15     5/16
ERSAL 000004          2/08
ERSCA 000325          5/46     5/47
ERSCS 000302          5/26     5/27
ERSDA 000377          6/28     6/29
ERSEC 000332          5/51     5/52
ERSEN 000060          2/52
ERSEQ 000273          5/19     5/20
ERSFT 000022          2/22
ERSHG 000357          6/12     6/13
ERSHP 000046          2/42
ERSHR 000460          7/28     7/29
ERSIM 000103          3/11
ERSMX 000350          6/05     6/06
ERSNI 000177          4/11
ERSNM 000074          3/04
ERSOC 000166          4/02
ERSPC 000021          2/21
ERSRE 000135          3/37
ERSRL 000036          2/34
ERSRR 000151          3/49
ERSTO 000115          3/21
ERSTR 000157          3/55
ERSTS 000164          3/60
ERTID 000014          2/16
ERTIM 000010          2/12
ERTIN 000152          3/50
ERTLM 000347          6/04     6/05
ERTMR 000470          7/36     7/37
ERTMT 000165          4/01
ERTOF 000257          5/07     5/08
ERTRC 000461          7/29     7/30
ERTRF 000317          5/39     5/40
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
  0062 PARU

ERTXT 000156          3/54
ERUAE 000233          4/43
ERUNI 000313          5/35    5/36
ERUNK 000424          6/50    6/51
ERUNM 000153          3/51
ERUOL 000123          3/27
ERUSY 000241          4/49
ERUTL 000442          7/08    7/09
ERVER 000334          5/53    5/54
ERVIU 000134          3/36
ERVNC 000430          6/54    6/57
ERVOL 000330          5/49    5/50
ERVTL 000440          7/06    7/07
ERWAB 000305          5/29    5/30
ERWAD 000032          2/30
ERWAK 000352          6/07    6/08
ERWMT 000215          4/25
ERWPB 000170          4/04
ERWSZ 000375          6/26    6/27
ERXMT 000012          2/14
ERXMZ 000120          3/24
ERXNA 000211          4/21
ERXUF 000212          4/22
UST   000400         43/04
USTAC 000015         43/19
USTBL 000017         43/21
USTCT 000014         43/18
USTDA 000006         43/12    43/30
USTEN 000022         43/26
USTES 000001         43/07
USTEZ 000000         43/06
USTFC 000016         43/20
USTFL 000007         43/13    43/50    43/51    43/52    43/53    43/54    43/55
                     43/56    43/57    43/58
USTGB 000006         43/30
USTGC 000002         43/31
USTGD 000012         43/32
USTIT 000011         43/15
USTOD 000020         43/22
USTRV 000012         43/16    43/32
USTS1 000004         43/10
USTS2 000005         43/11
USTSE 000003         43/09
USTSL 000010         43/14
USTSS 000002         43/08    43/31
USTST 000021         43/23
USTSZ 000022         43/24    43/26
USTTC 000013         43/17
?ABCD 000000         35/50    35/55
?ABLN 000004         35/55
?ABLP 000003         35/53    35/55
?ABPH 000001         35/51
?ABPL 000002         35/52
?AEPR 000005         22/02
?ALAU 000002         21/59
?AOPR 000006         22/03    22/04
?APBT 000010         19/34    20/08
?APND 000200         20/08
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0063 PARU

?ARBA 000001        48/17
?AREP 077700        48/22
?ARNO 000000        48/16
?ARNX 000003        48/23
?AROU 000002        48/18
?ASEB 000000        21/57
?AUAL 000003        21/60
?B2AP 000362        40/54
?B2CM 000360        40/52
?B2CP 000367        40/59
?B2DD 000364        40/56
?B2RC 000361        40/53
?B2RM 000365        40/57
?B2SD 000363        40/55
?B2TO 000366        40/58
?BAEA 000357        32/60
?BARA 000356        32/59
?BDLE 000346        32/57
?BFBI 000152        40/23
?BFDA 000145        40/21
?BFFO 000157        40/28
?BFNO 000154        40/25
?BFNR 000146        40/22
?BFOP 000153        40/24
?BFRA 000155        40/26
?BFSH 000140        40/20
?BFTI 000156        40/27
?BIBM 000121        39/23
?BMBI 000040        40/34
?BMDA 002000        40/32
?BMDB 000341        32/55
?BMFO 000001        40/39
?BMNO 000010        40/36
?BMNR 001000        40/33
?BMOP 000020        40/35
?BMRA 000004        40/37
?BMSH 100000        40/31
?BMSP 000002        55/53
?BMSU 000001        55/52
?BMTI 000002        40/38
?BPRM 000345        32/56
?BPSP 000076        55/58
?BPSU 000077        55/57
?BRDO 002120        39/24
?BSHB 000340        32/54
?BSSH 000033        45/01
?BSUN 000032        44/60
?BSWI 000035        45/03
?BSXR 000034        45/02
?BSYN 000031        44/59
?BTAB 000027        44/57
?BTGS 000026        44/56
?BTIG 000024        44/54
?BTIW 000025        44/55
?BTPN 000020        44/50
?BTRC 000023        44/53
?BTSG 000021        44/51
?BTSP 000022        44/52
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0064 PARU

?BTUF 000030        44/58
?BUDA 000347        32/58
?BUDB 000163        43/53
?BUDR 000162        43/52
?BUED 000167        43/57
?BUHD 000170        43/58
?BUHP 000166        43/56
?BUID 000164        43/54
?BUIN 000160        43/50
?BUIP 000161        43/51
?BUPH 000165        43/55
?C8BT 000003        13/14
?CACP 000002        23/09
?CCPS 000003        23/13
?CDEH 000005        23/17
?CDEL 000006        23/18
?CDMP 040000        20/01
?CDT0 000004        13/38
?CDT1 000005        13/39
?CDT2 000006        13/40
?CDT3 000007        13/41
?CEB0 000016        13/27   13/60
?CEB1 000017        13/28   13/59
?CEOC 000002        13/60
?CEOL 000012        13/22
?CEOS 000001        13/59
?CEPI 000002        13/08
?CESC 000013        13/45
?CFF  000015        13/25
?CFKT 000015        13/47
?CFTY 000000        23/07
?CHFS 000003        23/11
?CHID 000003        23/12
?CINT 000000        51/13
?CLAU 000004        23/14
?CLT  000014        13/24
?CLTH 000010        23/21
?CMIL 000007        23/19   23/21
?CMOD 000003        13/36
?CMSH 000005        23/15
?CMSL 000006        23/16
?CNAS 000010        13/20
?CNNL 000016        13/48
?CNRM 000002        13/35
?COBI 000001        54/11   54/17
?CONT 000001        51/14
?COTT 000011        13/21
?CPBN 000012        13/44
?CPID 000000        54/10   54/16
?CPM  000001        13/34
?CPOR 000003        23/10
?CRAC 000007        13/19
?CRAF 000005        13/17
?CRAT 000006        13/18
?CSFF 000001        13/07
?CSPO 000004        13/16
?CST  000000        13/06
?CTIM 000001        23/08
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0065 PARU

?CTO  000010        13/42
?CTSP 000011        13/43
?CUCO 000013        13/23
?CULC 000000        13/33
?CWRP 000014        13/46
?DAC2 000004        46/37
?DCC  000015        47/06
?DCI  000016        47/07    47/08
?DESC 000001        12/06
?DFBR 000201        47/13
?DFL0 100000        46/42
?DFL1 000001        46/44
?DFLG 000010        46/41    47/13
?DFLL 000210         8/13
?DFLR 040000        46/43
?DID  000002        46/35
?DIO  000000        18/51
?DLNK 000000        46/33
?DMAX 000177         8/39     8/41
?DMIN 000100         8/38    10/02
?DNUM 000012        46/46    46/47
?DPC  000003        46/36
?DPRI 000001        46/34
?DRES 000011        46/45
?DSFL 000007        46/40
?DSH  000013        47/04
?DSLT 000013        46/47
?DSMS 000014        47/05
?DSSZ 000006        46/39
?DSTB 000005        46/38
?DXLT 000017        47/08
?EBAS 000001        21/58
?ECLT 000004        47/22
?ECSS 000002        47/20
?ECSU 000000        47/18
?ECZS 000003        47/21    47/22
?ECZU 000001        47/19
?EFFP 000000        21/49
?EFLN 000001        21/50
?EFMA 000003        21/52
?EFNF 000000        21/44
?EFTY 000002        21/51    21/52
?ELAC 000012        22/21    22/23
?ELCR 000005        22/14
?ELCT 000007        22/16
?ELEP 100000        22/30
?ELFS 000013        22/23    22/24
?ELGN 000003        22/12
?ELL1 000400        22/31
?ELL2 001000        22/32
?ELL3 001400        22/33
?ELLN 000016        22/26
?ELR1 000014        22/24    22/25
?ELR2 000015        22/25    22/26
?ELRE 000006        22/15
?ELUH 000010        22/19
?ELUT 000011        22/20    22/21
?ELVL 000000        22/11
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0066 PARU

?ELVR 000004          22/13
?ENET 000017          21/19    21/21
?ENOR 000001          18/40    18/46    18/52
?ENOV 040000          18/46
?ENVR 000001          18/52
?EOFC 000004          14/21
?ESCP 004000          21/37
?ESCR 000002          21/29
?ESDD 002000          21/38
?ESED 010000          21/36
?ESEP 000001          21/28
?ESFC 000000          21/27
?ESNE 000400          21/40
?ESNR 020000          21/35
?ESRD 040000          21/34
?ESRP 001000          21/39
?ESSE 100000          21/33
?ETBB 000000          51/07
?ETFT 000015          21/17
?ETLT 000016          21/18
?ETMX 000034          21/23
?ETSP 000014          21/16
?ETSZ 000004          21/21
?ETXB 020000          51/08    51/23
?EURT 000000    MC    46/03#
?FAAB 000015          15/09    15/17
?FACA 000004          15/17
?FACE 000001          15/19
?FACO 000020          15/15
?FACR 000002          15/18
?FACW 000010          15/16
?FAEA 000001          32/48
?FAEB 000017          15/11    15/19
?FAFI 000114          10/14    10/15
?FAOB 000013          15/07    15/15
?FARA 000002          32/47
?FARB 000016          15/10    15/18
?FAWB 000014          15/08    15/16
?FAWS 000115          10/15    10/16
?FBCI 000116          10/16
?FCON 000061           9/41     9/42
?FCPD 000014           8/59     8/60     9/06
?FCRA 000045           9/33     9/34
?FDCF 000004           8/52     8/53
?FDIR 000012           8/57     8/58     9/02
?FDKU 000024           9/10     9/11     9/19
?FDLE 001000          32/45
?FFCC 000110          10/10    10/11
?FGFN 000003           8/51     8/52
?FGLT 000042           9/28
?FHST 000064           9/49     9/50
?FIPC 000036           9/24     9/26     9/44
?FLCC 000107          10/09    10/10
?FLCF 000005           8/53
?FLDU 000013           8/58     8/59     9/05
?FLNK 000000           8/48     8/49
?FLOG 000105          10/07    10/08
?FLPA 000054           9/38     9/39
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0067 PARU

?FLPC 000055          9/39     9/40
?FLPD 000030          9/14     9/20
?FLPU 000027          9/13     9/14     9/16
?FLTU 000030          9/16
?FLUG 000076          9/58
?FMCU 000025          9/11     9/12
?FMDB 040000         32/43
?FMTF 000002          8/50     8/51
?FMTU 000026          9/12     9/13
?FMTV 000015          8/60     9/03
?FNCC 000106         10/08    10/09
?FNGF 002010         11/39    11/40
?FNHI 000067          9/52
?FNLO 000063          9/47     9/48
?FNPN 000065          9/50     9/51
?FOCC 000111         10/11    10/12
?FPHI 000062          9/42     9/45
?FPIH 000051          9/34     9/35
?FPIL 000044          9/31     9/32
?FPLA 000053          9/37     9/38
?FPLO 000044          9/30     9/31
?FPOH 000060          9/40     9/41
?FPOL 000052          9/35     9/36
?FPRG 000101         10/03    10/04
?FPRM 002000         32/44
?FPRV 000112         10/12    10/13
?FPVC 000066          9/51     9/52
?FQUE 000041          9/27     9/28     9/30
?FREM 000063          9/48     9/49
?FSDF 000001          8/49     8/50
?FSGF 000010         11/20    11/21
?FSHB 100000         32/42
?FSPR 000040          9/26     9/27
?FSTF 000103         10/05    10/06
?FSYN 000074          9/55
?FTPA 000052          9/36     9/37
?FTRA 000044          9/32     9/33
?FTXT 000104         10/06    10/07
?FUDA 000400         32/46
?FUDF 000100         10/02    10/03
?FUPF 000102         10/04    10/05
?FWRD 000113         10/13    10/14
?GAC0 000003         55/41    55/42
?GAC1 000005         55/42    55/43
?GAC2 000007         55/43    55/44
?GARG 000003         34/18    34/19
?GCMD 000001         34/16    34/17
?GCNT 000002         34/17    34/18
?GDAT 000004         16/55
?GFCF 100000         34/25
?GFEX 040000         34/30
?GFXB 020000         34/34
?GHID 000000         55/39    55/40    55/47
?GIN  000001         16/52
?GLST 000003         16/54
?GMES 000000         34/15    34/16
?GNUL 000005         16/56
?GNUM 000001         34/07    34/08
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0068 PARU

?GOUT 000002        16/53
?GRCH 000002        25/19
?GRCL 000003        25/20
?GREQ 000000        34/06   34/07
?GRES 000003        34/09   34/11
?GRIH 000004        25/21
?GRIL 000005        25/22
?GRLN 000014        55/47
?GRLT 000010        25/26
?GRPH 000006        25/23
?GRPL 000007        25/24   25/26
?GRRH 000000        25/17
?GRRL 000001        25/18
?GSW  000002        34/08   34/09
?GSWS 000005        34/20
?GSYS 000001        55/40   55/41
?GTLN 000004        34/11
?GTSW 000004        34/19   34/20
?GXT2 000013        55/45   55/47
?GXTN 000011        55/44   55/45
?HAOH 000016        29/20   29/22
?HAOL 000017        29/22   29/24
?HAPH 000004        25/56
?HAPL 000005        25/57
?HAPS 000630        29/58
?HARA 000012        26/04
?HATH 000010        29/14   29/15
?HATL 000011        29/15   29/16
?HCMH 000024        30/04   30/05
?HCML 000025        30/05   30/06
?HCNT 000024        29/30   29/56
?HCPD 000014         9/06
?HDAT 000000        27/29   27/30
?HDIR 000015         9/03
?HGHH 000006        29/12   29/13
?HGHL 000007.       29/13   29/14
?HIBU 000003        25/43   25/45
?HICH 000077        17/03   17/04
?HIDH 000002        29/08   29/09
?HIDL 000003        29/09   29/10
?HIDS 177770         8/29
?HIEN 000001        25/41
?HILT 000004        25/45
?HIPC 000062         9/45    9/47
?HIST 000000        25/40
?HIWD 000002        25/42
?HIWH 000026        30/06   30/07
?HIWL 000027        30/07
?HMSK 077400         8/30
?HMXR 000377        28/39
?HMXS 000177        28/40
?HOGH 000014        29/18   29/19
?HOGL 000015        29/19   29/20
?HOPH 000012        29/16   29/17
?HOPL 000013        29/17   29/18
?HPID 000030        29/56   29/58
?HPMH 000004        29/10   29/11
?HPML 000005        29/11   29/12
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0069 PARU

?HPRH 000002        25/53
?HPRL 000003        25/55
?HRD  000004        27/36   28/39   28/40
?HRDC 000003        27/35   27/36
?HRE1 000001        27/30   27/31
?HRE2 000002        27/31   27/35
?HREN 000001        27/43   27/44
?HRIS 000002        27/44   27/45
?HRNH 000022        29/26   29/28
?HRNL 000023        29/28   29/30   30/04
?HRRE 000003        27/45   27/47
?HRST 000000        27/42   27/43
?HRSZ 000004        27/47   28/39
?HSAH 000002        28/12
?HSAL 000003        28/13
?HSBH 000006        25/58
?HSBL 000007        25/60
?HSBT 000003        27/56
?HSIH 000010        26/01
?HSIL 000011        26/02
?HSIS 000006        28/09
?HSKY 000005        28/15
?HSMN 000004        28/21
?HSMS 000004        28/14
?HSMX 000005        28/22
?HSOV 000002        28/05
?HSPC 000000        27/53   27/54
?HSPE 000004        28/07
?HSPS 000003        28/06
?HSR1 000001        28/03
?HSR4 000004        28/28
?HSR5 000005        28/08
?HSR6 000006        28/16
?HSR7 000007        28/10   28/33
?HSSZ 000010        28/33   28/40
?HSTY 000000        28/02
?HSVL 000002        27/55   27/56
?HSVM 000001        27/54   27/55
?HSYH 000020        29/24   29/25
?HSYL 000021        29/25   29/26
?HTAP 000002        27/17   27/18
?HTAS 000004        27/19   27/20
?HTMX 000005        27/22
?HTOH 000000        29/06   29/07
?HTOL 000001        29/07   29/08
?HTSY 000005        27/20   27/22
?HTTG 000003        27/18   27/19
?HTTH 000000        25/51
?HTTL 000001        25/52
?HTTP 000001        27/16   27/17
?HTXX 000000        27/15
?HUNT 000030         9/20
?IAC0 000004        55/06   55/07
?IAC1 000006        55/07   55/08
?IAC2 000010        55/08   55/09
?IBAD 000003        19/10   19/11
?IBEP 000005        19/54
?IBIB 000003        19/29   20/03
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
  0070 PARU

?IBIN 010000        20/03
?IBLT 000014        19/21   21/16   21/17   21/18   21/19   21/21   21/23
?IBNB 000002        41/50
?IBNS 000003        41/51
?IBPO 000006        19/55
?IBRF 000000        41/48
?IBSO 000001        41/49
?IBST 000000        41/47
?ICDM 000001        19/27   20/01
?ICFB 000001        19/26   19/60
?ICH  000000        19/07   19/08
?ICRF 040000        19/60
?ID16 010000        20/23
?ID8  004000        20/22
?IDAM 014000        20/21
?IDD0 000002        19/51
?IDD1 000003        19/52
?IDD2 000004        19/53
?IDEL 000013        19/19   19/21
?IDIO 000000        18/39   18/45   18/51
?IDPH 000002        41/18
?IDPL 000003        41/19
?IDPN 000004        41/25
?IERR 000011        55/09   55/13
?IEXO 002000        20/05
?IFLG 000004        19/12
?IFNB 020000        41/42
?IFNP 000011        19/17   19/18
?IFNS 010000        41/43
?IFOB 000004        19/30   20/04
?IFOP 004000        20/04
?IFRF 100000        41/40
?IFSO 040000        41/41
?IFST 100000        41/39
?IIPC 001000        20/06
?IIPS 000006        19/32   20/06
?ILTH 000005        41/27
?IMEP 002000        20/24
?IMIO 100000        18/45
?IMPO 001000        20/25
?IMPR 000177        41/08   41/59
?IMRS 000012        19/18   19/19
?IOEX 000005        19/31   20/05
?IOPH 000002        41/23
?IOPL 000003        41/24
?IOPN 000004        41/20
?IPID 000000        55/04   55/05   55/13
?IPKL 100000        19/59
?IPLB 000000        19/25   19/59
?IPLT 000007        41/30
?IPRL 000011        41/35
?IPST 020000        20/02
?IPTB 000002        19/28   20/02
?IPTR 000006        41/28   41/30
?IRCL 000005        19/13   19/14
?IRES 000004        19/11   19/12   19/13
?IRLR 000006        19/14   19/15
?IRLT 000007        41/32
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
  0071 PARU

?IRNH 000007        19/15    19/16
?IRNL 000010        19/16    19/17
?IRPT 000010        41/33    41/35
?ISFL 000000        41/14    41/47    41/48    41/49    41/50    41/51
?ISTI 000001        19/08    19/09
?ISTO 000002        19/09    19/10
?ITCB 000002        55/05    55/06
?IUFL 000001        41/15
?IWLN 000012        55/13
?LACT 000003        11/15    11/16
?LB16 010000        53/50
?LB8  004000        53/49
?LBAC 000006        53/31#   53/33
?LBAM 014000        53/51
?LBB0 000002        53/41
?LBB1 000003        53/42
?LBB2 000004        53/43
?LBDV 000001        53/25#   53/26
?LBFG 000000        53/24#   53/25
?LBIB 000000        53/39    53/47
?LBIM 100000        53/47
?LBLN 000011        53/35#
?LBOI 000005        53/30#   53/31
?LBR1 000007        53/33#   53/34
?LBR2 000010        53/34#   53/35
?LBSB 000001        53/40    53/48
?LBSC 040000        53/48
?LBST 000003        53/27#   53/29
?LBUV 000004        53/29#   53/30
?LBVD 000002        53/26#   53/27
?LCNT 002000        11/25    11/26
?LCOM 002051        11/48    11/52
?LCPD 000013         9/05
?LDEL 000025        14/20
?LDER 000004        11/16    11/17
?LDIR 000012         9/02
?LDMA 003777        11/07
?LDMI 002000        11/06    11/25
?LEND 000002        11/14    11/15
?LERC 000006        11/18    11/19
?LFER 002007        11/35    11/39
?LFTA 002050        11/44    11/48
?LFUC 002006        11/34    11/35
?LIPC 000036         9/44
?LMNT 002001        11/26    11/27
?LNCC 002004        11/32    11/33
?LNER 002005        11/33    11/34
?LNGF 002047        11/40    11/44
?LNUD 000200        23/39
?LOCH 000000        17/02
?LPAD 000005        11/17    11/18
?LPFL 000007        11/19    11/20
?LPPT 002003        11/28    11/32
?LSGF 000047        11/21
?LSMA 001777        11/05
?LSMI 000000        11/04
?LSNA 002052        11/52
?LSTR 000001        11/13    11/14
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
 0072 PARU

?LSUP 002002        11/27   11/28
?LUMA 010000        11/09
?LUMI 004000        11/08
?LUNT 000024         9/19
?MBLN 000004        42/12
?MBMA 000001        42/08
?MBNB 000003        42/10   42/12
?MBOA 000000        42/07   42/12
?MBPD 000002        42/09
?MCOB 040000        54/17
?MCPI 100000        54/16
?MFWT 100000        37/05
?MXAC 000377         8/28
?MXFN 000040         8/21
?MXFP 000100         8/20
?MXHN 000040         8/17    8/20
?MXIO 002000         8/27
?MXIP 002000         8/26
?MXLP 000377        41/09
?MXPI 000100         8/25
?MXPL 000200         8/14
?MXPN 000020         8/15    8/20
?MXUN 000020         8/16    8/20
?NDAR 000000        48/06
?NDAS 000004        48/11
?NDFA 000005        48/13
?NDFH 000002        48/09
?NDFL 000003        48/10
?NDNU 177777        48/03
?NDOV 000001        48/08
?NFKY 000000        33/05
?NFLN 000003        33/10
?NFNM 000001        33/07
?NFTP 000002        33/08
?NTRN 000000        51/01
?NULL 000100        17/04
?OAC0 177774        12/11
?OAC1 177775        12/12
?OAC2 177776        12/13
?OANS 040000        19/50
?OF1B 000011        19/35   20/09
?OF2B 000012        19/36   20/10
?OFCE 000040        20/10
?OFCR 000100        20/09
?OFIN 000020        20/11   20/13
?OFIO 000030        20/13
?OFOT 000010        20/12   20/13
?OFP  177777        12/14
?OIBM 100000        19/49
?OPAM 014000        16/46   20/21   53/51
?OPBE 000000        16/32   16/42
?OPBP 000006        16/38   16/48   19/55
?OPCH 000000        16/18
?OPD0 000002        16/34   19/51   40/14   53/41
?OPD1 000003        16/35   19/52   40/15   53/42
?OPD2 000004        16/36   16/44   16/45   16/46   19/53   40/16   53/43
?OPDF 000001        16/33   16/43
?OPDL 004000        16/44   20/22   53/49
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0073 PARU

?OPDM 010000          16/45    20/23    53/50
?OPEH 000003          16/21
?OPEL 000004          16/22    16/27
?OPEP 002000          16/47    20/24
?OPFC 000002          16/20
?OPFL 000000          16/17    16/18
?OPIB 000013          19/37    20/11
?OPLT 000005          16/27
?OPMD 040000          16/43
?OPME 100000          16/42
?OPMP 001000          16/48    20/25
?OPOB 000014          19/38    20/12
?OPPC 000005          16/37    16/47    19/54
?OPPH 000002          16/24
?OPPL 000003          16/25
?OPTY 000001          16/19
?ORDS 000002          16/06    21/06
?ORDY 000001          16/05    21/05
?ORFX 000003          16/07    21/07
?ORTN 000000          12/15
?ORUN 000005          16/09    21/09
?ORVB 000006          16/10    21/10
?ORVR 000004          16/08    21/08
?OVED 000000          49/32
?OVEO 000001          49/33
?PBLT 000007          18/18
?PCAD 000002          18/08    18/09
?PCAL 000010          24/15
?PCON 000007          24/14
?PCS1 000010          18/24    18/25
?PCS2 000011          18/25    18/26
?PCS3 000012          18/26    18/27
?PCS4 000013          18/27    18/28
?PCS5 000014          18/28    18/29
?PCS6 000015          18/29    18/30
?PCS7 000016          18/30    18/31
?PCS8 000017          18/31    18/33
?PDEL 000400          20/07
?PDFP 000017          24/23    24/27
?PDIR 000006          24/13
?PDLM 000007          19/33    20/07
?PFBS 000400          24/39
?PFDA 002000          24/37
?PFDB 040000          24/33
?PFEX 020000          24/34
?PFLG 000000          24/07
?PFPM 010000          24/35
?PFPP 100000          24/32
?PFPX 004000          24/36
?PFRP 000001          24/42
?PFRS 000002          24/41
?PIFP 000014          24/20
?PIPC 000002          24/09
?PLFP 000016          24/22
?PLTH 000020          24/27    24/28
?PMEM 000004          24/11
?PMSK 000377          35/15
?PNM  000003          24/10
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
  0074 PARU

?POFP 000015        24/21
?PPBL 000020        18/33
?PPCR 000013        24/18
?PPRI 000005        24/12
?PPRV 000012        24/17
?PRBB 000007        18/23   18/24
?PRCL 000006        18/14   18/18   18/23
?PRES 000003        18/09   18/10
?PRNH 000004        18/10   18/12
?PRNL 000005        18/12   18/14
?PSAL 000020        31/16   31/17
?PSBK 000031        31/25   31/26
?PSCH 000050        31/43   31/44
?PSCL 000051        31/44   31/45
?PSEX 000041        31/34   31/35
?PSF2 000026        31/22   31/23
?PSF3 000027        31/23   31/24
?PSF4 000042        31/35   31/36
?PSFL 000025        31/21   31/22
?PSFP 000000        31/12   31/13
?PSIH 000052        31/45   31/46
?PSIL 000053        31/46   31/47
?PSLT 000056        31/50
?PSMB 000034        31/28   31/29
?PSMX 000044        31/37   31/38
?PSNM 000001        24/08
?PSNR 000021        31/17   31/18
?PSNS 000022        31/18   31/19
?PSPD 000043        31/36   31/37
?PSPH 000054        31/47   31/48
?PSPL 000055        31/48   31/50
?PSPR 000030        31/24   31/25
?PSPS 000032        31/26   31/27
?PSPV 000040        31/33   31/34
?PSQF 000024        31/20   31/21
?PSRH 000046        31/41   31/42
?PSRL 000047        31/42   31/43
?PSSF 000033        31/27   31/28
?PSSL 000045        31/38   31/41
?PSSN 000001        31/13   31/17
?PSST 000023        31/19   31/20
?PSSW 000037        31/31   31/33
?PSTI 000000        18/06   18/07   18/51   18/52   18/53
?PSTO 000001        18/07   18/08
?PSYS 000035        31/29   31/30
?PSYT 000036        31/30   31/31
?PUNM 000011        24/16
?PVDV 000200        24/54
?PVEX 000040        24/52
?PVIF 001000        24/56
?PVIP 000400        24/55
?PVPC 000001        24/47
?PVPP 000020        24/51
?PVPR 000010        24/50
?PVSP 002000        24/57
?PVSU 000004        24/49
?PVTY 000002        24/48
?PVUI 000100        24/53
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
  0075 PARU

?PXLT 000040        24/28
?RAC0 000006        55/25   55/26
?RAC1 000010        55/26   55/27
?RAC2 000012        55/27   55/28
?RCFA 177765        35/21
?RCMK 003400        51/42
?RF1B 000015        19/39   21/01
?RF2B 000016        19/40   21/01
?RF3B 000017        19/41   21/01   21/05   21/06   21/07   21/08   21/09
                    21/10
?RFAB 060000        36/08
?RFBS 000007        21/01
?RFCF 100000        36/05
?RFEC 010000        36/09
?RFER 040000        36/07
?RFWA 020000        36/06
?RHID 000003        55/23   55/24   55/57   55/58
?RMSK 174000        35/13
?RPID 000000        55/19   55/20   55/31
?RPLN 000017        55/31
?RSAV 000052  MC    46/24#
?RSYS 000004        55/24   55/25
?RTCB 000002        55/20   55/23
?RTDS 000002        21/06
?RTDY 000001        21/05
?RTFX 000003        21/07
?RTMK 000007        51/45
?RTUN 000005        21/09
?RTVB 000006        21/10
?RTVR 000004        21/08
?RXT2 000016        55/29   55/31
?RXTN 000014        55/28   55/29
?SACK 000400        51/31   51/42
?SACP 000002        32/08
?SAFB 000002        18/53
?SAFE 000002        18/41   18/47   18/53
?SAFM 020000        18/47
?SAS8 000100        50/29
?SASC 020000        50/28
?SBER 000002        51/51
?SBRH 000010        52/12
?SBRL 000011        52/13
?SBSC 000000        50/36
?SBSH 000004        52/08
?SBSL 000005        52/09
?SBUP 000001        50/44   50/45
?SBYC 000002        50/45   50/46
?SBYM 000004        50/48   50/49
?SCAC 000011        47/36
?SCEG 000000        47/27
?SCET 000013        47/38
?SCGS 000012        47/37
?SCGX 000004        47/31
?SCHN 000000        50/06   50/07
?SCIT 000400        50/34
?SCLB 107510        47/43
?SCLD 000007        47/34
?SCLE 000006        47/33
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0076 PARU

?SCLG 113510        47/44
?SCLS 000010        47/35
?SCMK 174000        51/23
?SCON 004000        51/11   51/23
?SCPS 000003        32/12
?SCRC 000000        50/33
?SCRM 000001        47/28
?SCSH 000025        32/33
?SCSL 000026        32/34   32/35
?SCSM 000005        47/32
?SCST 000003        47/30
?SCXS 000002        47/29
?SDAC 000040        51/28
?SDAD 000003        50/46   50/47
?SDCN 100000        36/28
?SDCU 000003        32/11
?SDDN 000000        36/27
?SDEH 000005        32/17
?SDEL 000006        32/18
?SDET 000004        51/17
?SDIS 000007        51/20
?SDMD 100000        50/24
?SDOR 000023        52/23
?SDPP 000000        50/23
?SDPR 040000        50/26
?SDSC 000000        50/25
?SDTI 000100        36/30
?SDTO 000200        36/29
?SDTP 000040        36/31
?SEBC 000000        50/27
?SEFH 000017        32/27
?SEFL 000020        32/28
?SELN 000004        50/10
?SEOT 000003        51/16
?SEPR 004000        50/32
?SEXT 000006        50/54   50/55
?SFAH 000021        32/29
?SFAL 000022        32/30
?SFCS 000022        52/22
?SFSZ 000005        50/15   50/16
?SFTM 000006        50/16   50/17
?SGHL 000024        52/24
?SGLN 000003        51/52
?SHAP 000007        19/45   20/17
?SHFS 000003        32/10
?SHLN 000012        50/20
?SHLT 000010        50/56
?SHOP 000400        20/17
?SHRD 000002        52/34
?SHRR 000001        52/33
?SHRS 000000        52/32
?SHSP 000001        50/37
?SIDX 000023        32/31
?SIGP 000200        41/59
?SIHI 000006        52/10
?SIID 000003        36/19
?SILN 000002        36/18
?SILO 000007        52/11
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0077 PARU

?SIMM 000001        36/17
?SIN  000001         8/04
?SIND 000007        50/55   50/56
?SINT 000006        51/19
?SIPL 000010        36/21
?SIRL 000003        50/47   50/48
?SIRN 000000        36/16
?SIRX 000012        52/14
?SITB 010000        51/10   51/23
?SIXH 000002        52/06
?SIXL 000003        52/07
?SLAU 000004        32/13
?SLBA 000000        49/42
?SLBB 000004        49/48
?SLBC 000007        49/52
?SLBF 000003        49/46
?SLBH 000005        49/50
?SLBL 000006        49/51
?SLBS 000002        49/45
?SLBU 000001        49/44
?SLDC 000000        49/18
?SLDE 000004        49/25
?SLDH 000003        49/22
?SLDL 000002        49/21
?SLDN 000001        49/20
?SLED 000000        49/37
?SLEO 000001        49/38
?SLLD 000000        49/08
?SLME 000100        49/10
?SLRC 001000        50/35
?SLTH 000027        32/35
?SMAX 000077         8/36    8/38
?SMCH 000036        24/24
?SMCL 000037        24/25
?SMDI 000003        50/09   50/10   50/14
?SMIL 000007        32/19
?SMIN 000000         8/35    8/48
?SMSH 000005        32/15
?SMSL 000006        32/16
?SNAK 001000        51/32   51/42
?SNKC 000000        51/49
?SNPR 000000        50/30
?SOHB 040000        51/05   51/23
?SOPN 000024        32/32
?SOPR 002000        50/31
?SPAR 000004        22/01
?SPET 000005        51/37   51/45
?SPLR 000200        51/30
?SPNH 000003        32/09
?SPNK 000007        51/39   51/45
?SPNL 000004        32/14
?SPPT 000012        41/60
?SPRA 000010        50/18   50/19
?SPRM 000011        41/58
?SPRV 000006        51/38   51/45
?SPTM 000010        41/57
?SPWK 000002        51/36   51/45
?SRIM 000013        42/01
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
  0078 PARU

?SRJR 000016          52/18
?SRJT 000017          52/19
?SRNR 000014          52/16
?SRNT 000015          52/17
?SRSR 000020          52/20
?SRST 000021          52/21
?SRTY 000007          50/17    50/18
?SRVI 002000          51/33    51/42
?SSAB 000003          50/39
?SSAR 000002          50/38
?SSDA 000011          50/19    50/20
?SSIS 000000          50/43    50/44
?SSLR 000100          51/29
?SSNL 000006          50/50
?SSTI 000001          50/07    50/08
?SSTS 000016          32/26    32/54    32/55    32/56    32/57    32/58
                      32/60
?STAH 000012          32/22
?STAL 000013          32/23
?STCH 000010          32/20
?STCL 000011          32/21
?STHI 000000          52/04
?STIM 000001          32/07
?STLO 000001          52/05
?STMH 000014          32/24
?STML 000015          32/25
?STOC 000002          50/08    50/09
?STOR 000013          52/15
?STOV 000005          50/49    50/50    50/54
?STTD 000005          51/18
?STTO 000001          51/50
?STXB 000000          51/04
?STYP 000000          32/06
?SURT 000025  MC      46/14#
?SWAK 000002          51/15
?SWSZ 000004          50/14    50/15
?T32T 000100          35/43
?TABR 000010          35/42
?TAC0 000006          44/13
?TAC1 000007          44/14
?TAC2 000010          44/15
?TAC3 000011          44/16
?TAOS 002000          35/32
?TATH 000002          23/29
?TATL 000003          23/30
?TBCX 002400          35/33
?TBLT 000006          23/34
?TCCX 003000          35/34
?TCFA 177771          35/22
?TCIN 001000          35/30
?TCTH 000000          23/27
?TCTL 000001          23/28
?TCUD 000016          44/21
?TEFH 000001          22/39    22/40
?TEFL 000003          22/41    22/42
?TEFM 000002          22/40    22/41
?TEFW 000000          22/38    22/39
?TELN 000014          44/19
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0079 PARU

?TEXT 003400        35/35
?TFP  000003        44/10
?TFPS 000015        44/20
?TGEX 000023        44/26   44/28
?TGXL 000016        44/29
?TIDP 000020        44/23
?TKAD 000022        44/25
?TLN  000024        44/28
?TLNK 000000        44/07   44/28
?TLTH 000004        22/42
?TMP  000003        46/29
?TMSK 003400        35/14
?TMTH 000004        23/31
?TMTL 000005        23/32   23/34
?TPC  000012        44/17
?TR32 000101        35/44
?TRAN 100000        51/02   51/23
?TRAP 000400        35/29
?TRMA 000007        22/04
?TSAB 000400        44/40
?TSEL 000000        35/28
?TSGS 001000        44/39
?TSIG 004000        44/37
?TSIW 002000        44/38
?TSL  000004        44/11
?TSLK 000021        44/24
?TSO  000005        44/12
?TSP  000002        44/09   44/29
?TSPN 100000        44/33
?TSRC 010000        44/36
?TSSG 040000        44/34
?TSSH 000020        44/44
?TSSP 020000        44/35
?TSTA 000001        44/08   44/50   44/51   44/52   44/53   44/54   44/55
                    44/56   44/57   44/58   44/59   44/60   45/01   45/02
                    45/03
?TSUF 000200        44/41
?TSUN 000040        44/43
?TSUP 001400        35/31
?TSWI 000004        44/46
?TSXR 000010        44/45
?TSYN 000100        44/42
?TSYS 000017        44/22   44/29
?TUSP 000013        44/18
?UDBM 000000        37/17
?UDDA 002000        37/18
?UDDB 002040        37/19
?UDDC 002100        37/20
?UDDD 002140        37/21
?UDDR 000004        37/15
?UDVB 000003        37/14
?UDVI 000002        37/13
?UDVM 000001        37/12
?UDVX 000000        37/11
?UFDB 010000        43/41
?UFDR 020000        43/40
?UFED 000400        43/45
?UFHD 000200        43/46
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
0080 PARU

?UFHP 001000        43/44
?UFID 004000        43/42
?UFIN 100000        43/38
?UFIP 040000        43/39
?UFPH 002000        43/43
?UMAX 000377         8/42
?UMIN 000200         8/41
?VF01 000017        53/13   53/16
?VFBO 000005        53/09   53/10
?VFCP 000002        53/06   53/07
?VFLE 000174        53/16
?VFLP 000003        53/07   53/08
?VFNO 000000        53/19
?VFRE 000001        53/05   53/06
?VFTB 000006        53/10   53/13
?VFTO 000004        53/08   53/09
?VFTY 000000        53/03   53/05   53/16
?VFWC 000011        53/12   53/13   53/16
?VRTN 000002        12/05
?X2AP 000002        40/44   40/54
?X2CM 000000        40/42   40/52
?X2CP 000007        40/49   40/59
?X2DD 000004        40/46   40/56
?X2RC 000001        40/43   40/53
?X2RM 000005        40/47   40/57
?X2SD 000003        40/45   40/55
?X2TO 000006        40/48   40/58
?XAFD 000013        39/58
?XAFT 000014        39/59
?XDAT 000002        39/32
?XDUL 000001        39/27
?XDUT 000002        39/28
?XFBI 000012        39/48   40/23   40/34
?XFBP 000011        39/56
?XFCA 000021        38/23
?XFDA 000005        39/43   40/21   40/32
?XFDU 000003        38/09
?XFFO 000017        39/53   40/28   40/39
?XFFT 000014        38/18
?XFGS 000006        39/37   40/20   40/21   40/22   40/23   40/24   40/25
                    40/26   40/27   40/28
?XFHA 000012        38/16
?XFHO 000017        38/21
?XFLC 000025        38/29
?XFLO 000024        38/28
?XFLP 000006        38/12
?XFMA 000031        38/37
?XFMI 000001        38/36
?XFML 000002        38/08
?XFMU 000001        38/07   38/36
?XFNO 000014        39/50   40/25   40/36
?XFNR 000006        39/44   40/22   40/33
?XFNV 000027        38/31
?XFOP 000013        39/49   40/24   40/35
?XFP1 000001        40/07
?XFP2 000002        40/08
?XFP3 000003        40/09   40/12   40/13   40/14   40/15   40/16   40/17
?XFPL 000011        38/15
```

Figure D-1. PARU.LS Parameter Listing (continued)

```
  0081 PARU

?XFPT 000007        38/13
?XFRA 000015        39/51   40/26   40/37
?XFRJ 000013        38/17
?XFSH 000000        39/38   40/20   40/31
?XFST 000022        38/24
?XFSU 000005        38/11
?XFSV 000031        38/33   38/37
?XFTI 000016        39/52   40/27   40/38
?XFUN 000020        38/22
?XFWV 000026        38/30
?XFXM 000016        38/20
?XFXU 000015        38/19
?XLMT 000004        39/34
?XLTH 000021        40/04
?XMFC 000000        39/11   40/12
?XMFI 000001        39/12   39/19   39/23   40/13
?XMFR 000005        39/16   39/20   40/17
?XMIB 040000        39/19
?XMLE 000007        39/07   39/08
?XMLF 000005        39/05
?XMLL 000001        38/59
?XMLR 000004        39/04
?XMLS 000006        39/06
?XMLT 000002        38/60
?XMLV 000003        39/01
?XMLX 000010        39/08
?XMRO 002000        39/20   39/24
?XMUE 000007        38/55   38/56
?XMUF 000005        38/53   39/05   39/23   39/24
?XMUL 000001        38/47   38/59
?XMUQ 000003        38/51
?XMUR 000004        38/52
?XMUS 000006        38/54
?XMUT 000002        38/48   38/60   39/28
?XMUX 000010        38/56
?XPBP 000012        39/57
?XPRI 000005        39/35
?XRES 000010        39/55
?XRFN 000000        38/44
?XSCV 000060        40/12
?XSD1 000063        40/15
?XSD2 000064        40/16
?XSDE 000062        40/14
?XSEQ 000007        39/54
?XSIB 000061        40/13
?XSRO 000065        40/17
?XTIM 000003        39/33
?XTYP 000001        39/31
?XXW0 000015        39/60
?XXW1 000016        40/01
?XXW2 000017        40/02   40/52   40/53   40/54   40/55   40/56   40/57
                    40/58   40/59
?XXW3 000020        40/03
```

Figure D-1. PARU.LS Parameter Listing (concluded)

End of Appendix

Appendix E
Powers of 2 Table

2^n	n	$1/2^n$
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125

End of Appendix

Appendix F
Selecting Multiprogramming or Multitasking

Many applications can be implemented using either multiprogramming or multitasking. For some of these applications, you may have no compelling reasons for choosing one method over the other, and you may decide largely according to your personal taste. However, for other applications, your choice may make the difference between a highly successful implementation and one which is merely adequate. Therefore, when you analyze an application and reduce it into specific tasks or processes, you must be aware of the unique advantages, and relative disadvantages, of each of these methods.

Advantages of Multiprogramming

Dividing an application into several different processes has four advantages:

- Processes are autonomous and cannot be subverted by other processes.
- Each process has a unique and secure address space.
- Each process can have a full 64K-byte logical address space.
- You can initiate and terminate processes from the console.

If a portion of an application can fail without totally degrading the entire system, then it is to your advantage to implement different parts of the application in different processes. In this case, if one process is abnormally terminated, other processes in the application can continue toward their successful conclusion.

Further, in a multiprogramming implementation each process has its own logical address space of up to 64K bytes, where a maximum 64K address space may be a severe limitation for many tasks co-existing in one process. While tasks within a process can effectively extend their address space by calling overlays, shared routines, and shared libraries, the tasks must compete for use of shared partitions and overlay areas. Allocating different tasks to different processes decreases this competition.

Finally, from a console you can initiate a process in the application when you need it to perform some function. Afterward, the process can be terminated, releasing up to 64K bytes of main memory.

Advantages of Multitasking

There are four advantages inherent to multitasking:

- Tasks can share the same physical address space, thus occupying less memory.
- Tasks can synchronize and communicate rapidly.
- Passing data between tasks is efficient.
- Multitasking forces structured programming.

Tasks using common re-entrant code paths in the same physical address space are ideal candidates to reside in the same process. The system requires very little memory to manage task activity, but much more memory to manage each process.

Intertask synchronization and communication (via ?XMT/?REC calls) is very fast, since all tasks reside in the same address space.

Finally, there are many functions, such as defining support of a user device (?IDEF) which you can implement only through multitasking. Clearly, functions which require rapid asynchronous service, such as device interrupt routines, time-out routines, and alarm routines to name a few, demand the quick response which only multitasking can offer. Only multitasking provides a structured control of event-driven processes.

Application Example

In this example application, we combine the advantages of both methods in a multiprogramming-multitasking implementation of an on-line Management Information System (MIS). This application has the following discrete functions:

Function	Description
Accept new orders	A highly interactive function requiring 8K bytes of shareable code, 8K bytes of data, 8 consoles, and a timeout capability on input.
On-line data query/update	A function which is interactive at one time and compute-bound at others. It requires 16K bytes of shareable code, 8K bytes of data, and 8 consoles.
Report generation	This low-priority function is not interactive, but performs I/O frequently. It requires 16K bytes of shareable code and 32K bytes of data.
Badge reader and time-clock	These functions require a very fast response; they need 4K bytes of data, 4K bytes of shareable code, and 16 PARANOIA Badge Readers. This program uses ?IDEF to interface these user devices to the system.

We can implement this application using the following strategy. We dedicate an order-entry process to each console. These processes contain several tasks: a task to read console input, a separate asynchronous task to terminate the process if the console is inactive for 10 minutes, an asynchronous task to write complete orders to disk concurrently with console input, and an asynchronous task to issue an ?IREC for global MIS status messages. These processes are swappable since any given terminal may be inactive; they are separate because we want each process's data separate and protected, and because any given console's process could fail without affecting other console activity.

We dedicate a swappable query process to each console. These processes will contain three tasks: one to read console input, another asynchronous task to write updates to disk, and a third asynchronous task to wait for MIS status messages. It is highly interactive at times and is assigned a high priority. However, when querying the database, AOS will heuristically decrease its priority in proportion to how compute-bound it becomes. Thus, one console's process will not degrade the response time of the other consoles.

Reports are run in a swappable batch process at the lowest priority, so that they will not interfere with very compute-bound query programs. Each report process creates multiple tasks to decrease I/O latency and increase efficiency by concurrently reading and writing I/O buffers.

The badge-reader/time-clock function runs in a resident process since it provides user-device interrupt service. It performs badge number validation, accesses the employee file to determine whether the employee is authorized access on this day and at this time, and enters the employee's entrance and exit in a log file. This process uses multitasking to overlap disk I/O with reader service. There will be one task per badge reader (16 readers). We could have used 16 separate processes, but since the routines are very short and totally re-entrant, we chose multitasking.

End of Appendix

Index

Within this index, "f" (or "ff") after a page number means "and the following page" (or "pages"). Commands, calls, and acronyms are in uppercase letters (e.g., ?CREATE); all others are lowercase.

A

B

C

D

E

F

G

H

I

M

P

Q

R

S

U

V

W

X

Z

Data General

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA® 4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		

Other _____
(Specify) _____

3. Software

- ☐ AOS
- ☐ AOS/VS
- ☐ AOS/RT32
- ☐ MP/OS
- ☐ MP/AOS
- ☐ RDOS
- ☐ DOS
- ☐ RTOS
- ☐ Other

Specify _____

4. Languages

- ☐ ALGOL
- ☐ DG/L
- ☐ COBOL
- ☐ Interactive COBOL
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ FORTRAN 77
- ☐ FORTRAN 5
- ☐ RPG II
- ☐ PL/1
- ☐ APL
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

- ☐ HASP
- ☐ HASP II
- ☐ RJE80
- ☐ RCX 70
- ☐ RSTCP
- ☐ 4025
- ☐ X.25
- ☐ SAM
- ☐ CAM
- ☐ XODIAC™
- ☐ DG/SNA
- ☐ 3270
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

CUT ALONG DOTTED LINE

Data General

Data General Corporation, Westboro, Massachusetts 01580, (617) 366-8911

FOLD

TAPE

FOLD

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

Data General

TP_____

TIPS ORDER FORM
Technical Information & Publications Service

BILL TO:
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

SHIP TO: (if different)
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form)

[Minimum order is $50.00]

Tax Exempt #_____
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No._____ Expiration Date_____

☐ Purchase Order Number:_____

SHIP VIA

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other _____
_____

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order _____ Phone _____ Extension _____

Mail Orders to:

Data General Corporation
Attn: Educational Services/TIPS F019
4400 Computer Drive
Westboro, MA 01580
Tel. (617) 366-8911 ext. 4032

Buyer's Authorized Signature Date
(agrees to terms & conditions on reverse side)

Title

DGC Sales Representative (If Known) Badge #

DISCOUNTS APPLY TO MAIL ORDERS ONLY

012-1780

educational services

CUT ALONG DOTTED LINE

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. **PRICES**
   Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

2. **PAYMENT**
   Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

3. **SHIPMENT**
   Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

4. **TERM**
   Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

5. **CUSTOMER CERTIFICATION**
   Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

6. **DATA AND PROPRIETARY RIGHTS**
   Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

7. **DISCLAIMER OF WARRANTY**
   DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

8. **LIMITATIONS OF LIABILITY**
   IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYRIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

9. **GENERAL**
   A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

DISCOUNT SCHEDULES

DISCOUNTS APPLY TO MAIL ORDERS ONLY.

LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.

DataGeneral

TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   TIPS
   Educational Services – M.S. F019
   Data General Corporation
   4400 Computer Drive
   Westboro, MA 01580
8. We'll take care of the rest!

educational services

User Documentation Remarks Form

Your Name _____ Your Title _____

Company _____

Street _____

City _____ State _____ Zip _____

We wrote this book for you, and we made certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve the manual. Please take a few minutes to respond. Thank you.

Manual Title _____ Manual No. _____

Who are you? ☐EDP Manager ☐Analyst/Programmer ☐Other _____
☐Senior Systems Analyst ☐Operator _____

What programming language(s) do you use? _____

How do you use this manual? *(List in order: 1 = Primary Use)* _____

___ Introduction to the product ___ Tutorial Text ___ Other
___ Reference ___ Operating Guide _____

About the manual:	Yes	Somewhat	No
Is it easy to read?	☐	☐	☐
Is it easy to understand?	☐	☐	☐
Are the topics logically organized?	☐	☐	☐
Is the technical information accurate?	☐	☐	☐
Can you easily find what you want?	☐	☐	☐
Does it tell you everything you need to know	☐	☐	☐
Do the illustrations help you?	☐	☐	☐

If you have any comments on the software itself, please contact Data General Systems Engineering.
If you wish to order manuals, use the enclosed TIPS Order Form (USA only).

Remarks:

Date

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

POSTAGE WILL BE PAID BY ADDRESSEE

DataGeneral

User Documentation, M.S. E-111
4400 Computer Drive
Westborough, Massachusetts 01581

How to Insert These Dividers

These dividers will make your manual easier to use. We suggest inserting them as follows.

Divider Name	Insert
Introduction to AOS	Before Chapter 1
Process Creation and Management	Before Chapter 2
Memory Management	Before Chapter 3
Interprocess Communications	Before Chapter 4
File Creation and Management	Before Chapter 5
File Input/Output	Before Chapter 6
Multitasking	Before Chapter 7
User Device Support	Before Chapter 8
Miscellaneous Calls	Before Chapter 9
Binary Synchronous Communications	Before Chapter 10
Connection Management	Before Chapter 11
Array Processing	Before Chapter 12
Block I/O	Before Chapter 13
Appendixes	Before Appendix A

Data General Corporation, Westboro, MA 01580

093-000120-05