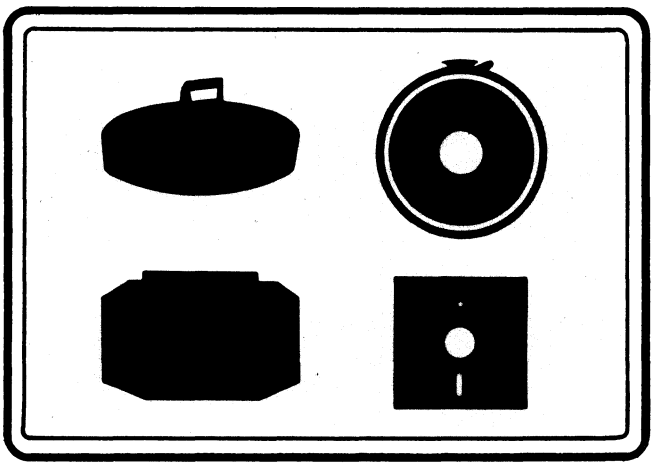
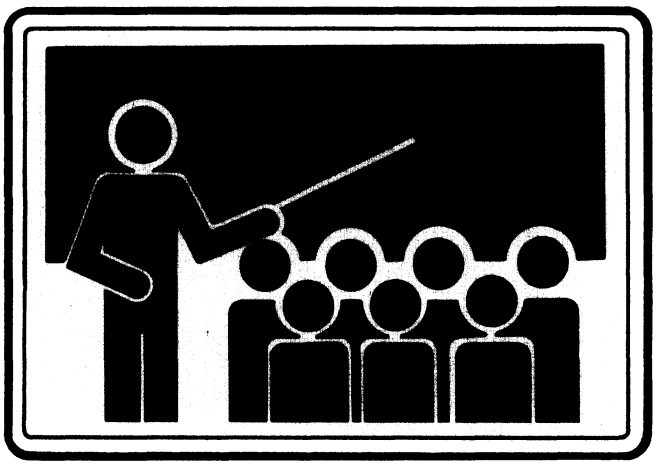
DataGeneral

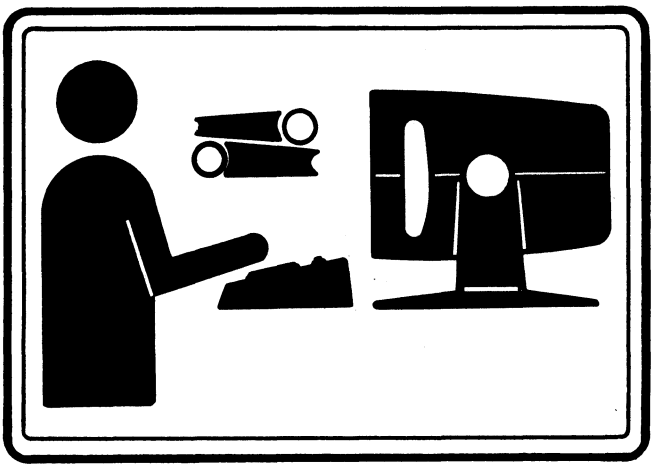
Educational Services

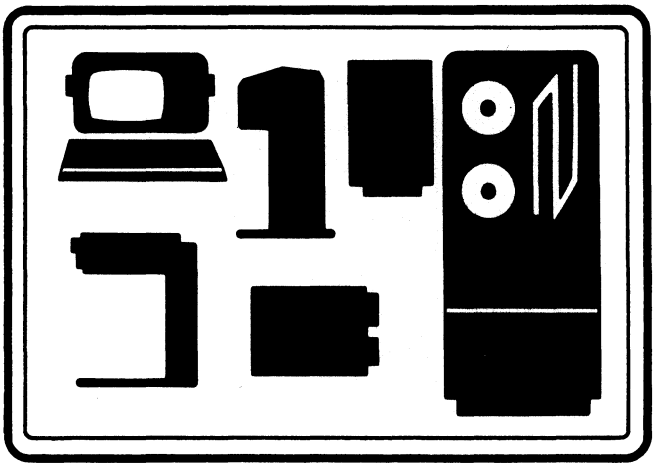
S309/VS
AOS/VS SYSTEM PROGRAMMING

COURSE MATERIALS









019-000153-01

NOTICE

DATA GENERAL CORPORATION (DGC) HAS PREPARED THIS DOCUMENT FOR USE BY DGC PERSONNEL, LICENSEES, AND CUSTOMERS. THE INFORMATION CONTAINED HEREIN IS THE PROPERTY OF DGC AND SHALL NOT BE REPRODUCED IN WHOLE OR IN PART WITHOUT DGC PRIOR WRITTEN APPROVAL.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

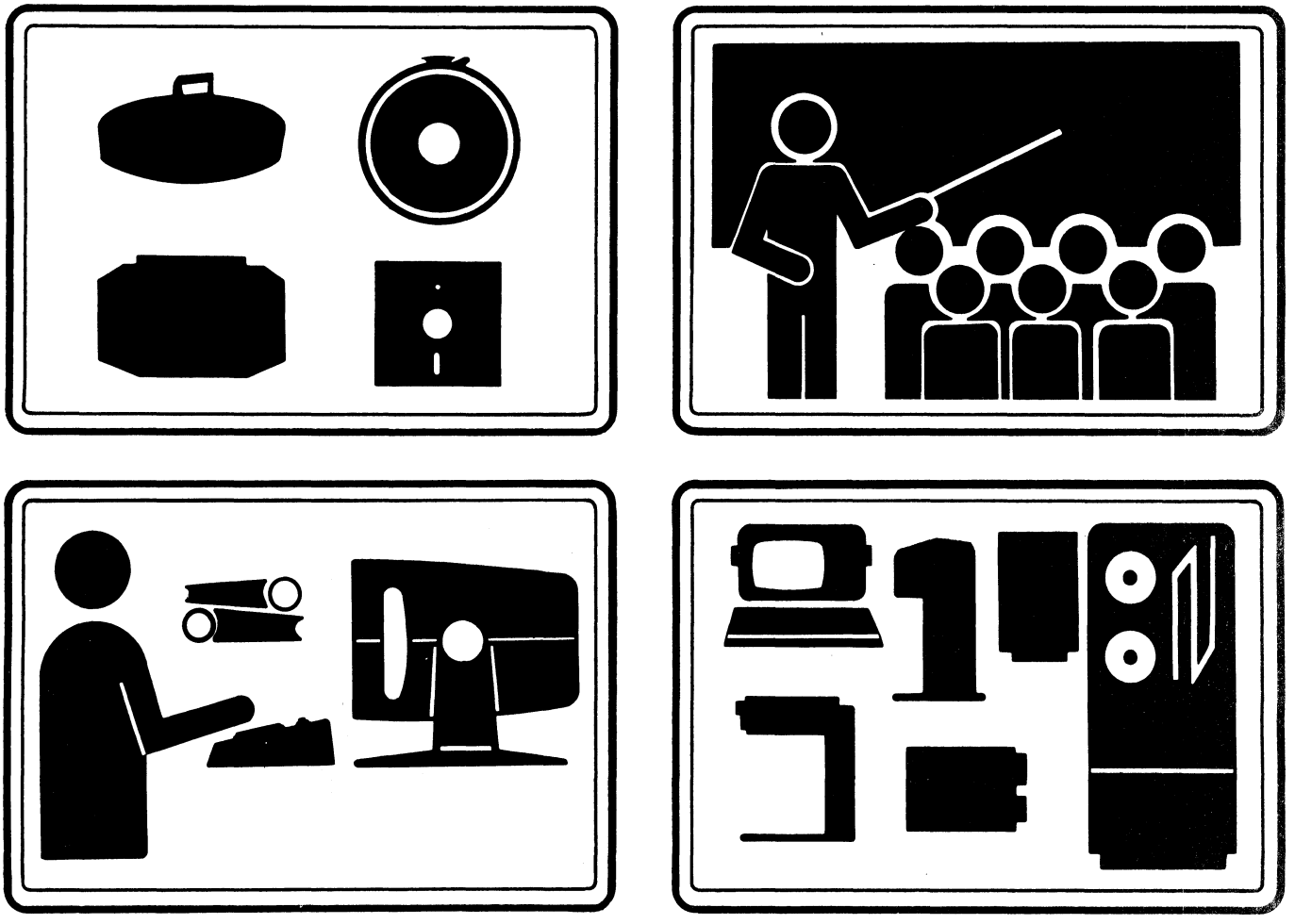
THE TERMS AND CONDITIONS GOVERNING THE SALE OF DGC HARDWARE PRODUCTS AND THE LICENSING OF DGC SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN DGC AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY DGC FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY OF DGC WHATSOEVER.

DataGeneral

Educational Services

S309/VS
AOS/VS SYSTEM PROGRAMMING

COURSE MATERIALS



019-000153-01

AOS/VS SYSTEM PROGRAMMING

Summary

S309VS teaches the student how to use AOS/VS System Calls in user written programs.

Prerequisites

S209 (AOS/VS & AOS USER) or equivalent experience: The student is expected to be thoroughly familiar with the Command Line Interpreter (CLI), especially files and directories, and to be fluent with one of the standard text editors (SED or SPEED).

S105VS (MV FAMILY ASSEMBLY LANGUAGE PROGRAMMING) or high level language programming experience: Although the use of system calls from high level languages is covered, the student is expected to be able to understand the AOS/VS System Programmer's Reference Manual, which is directed toward the assembly language programmer.

Objectives

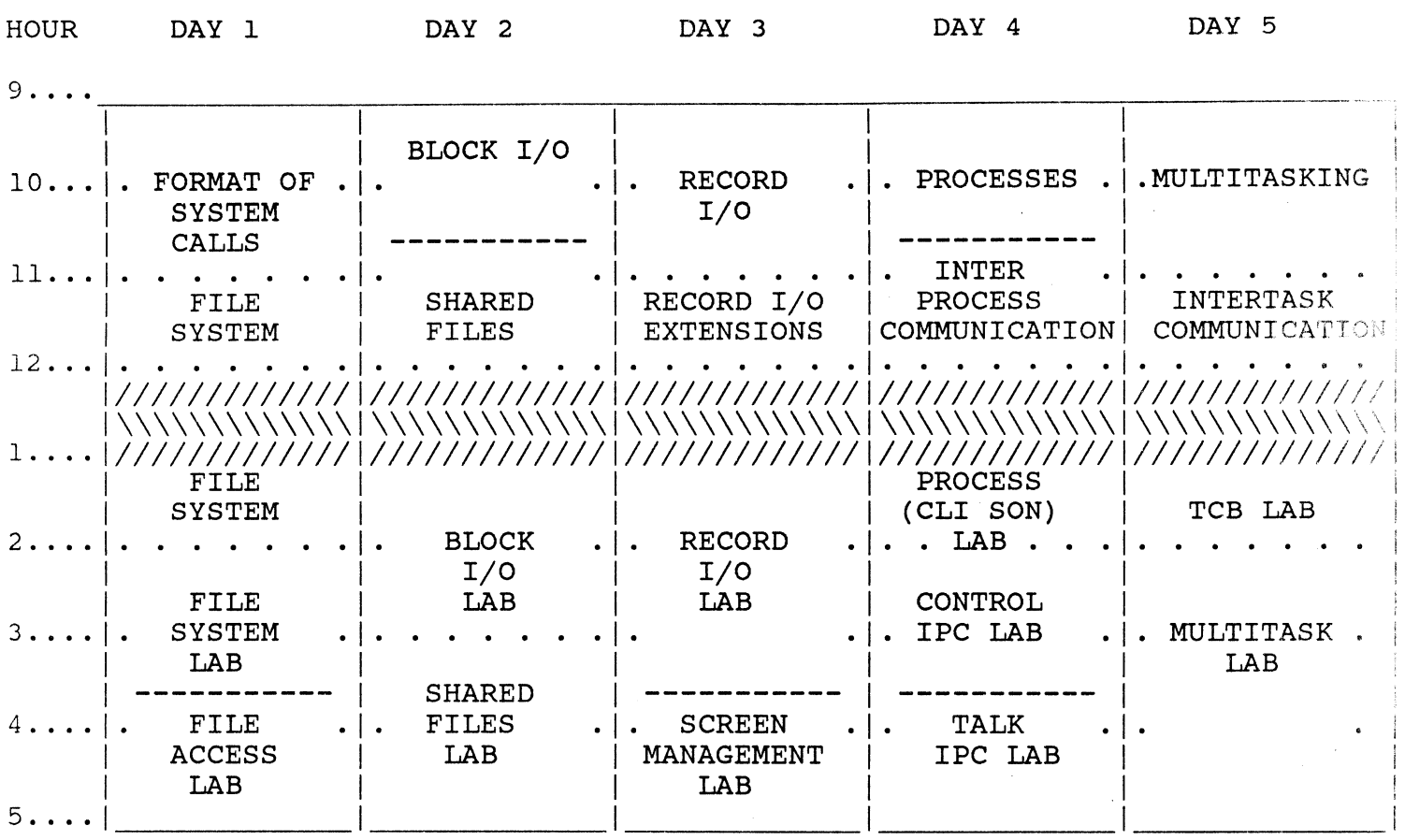
Upon completion of this course, the student should be able to:

1. Describe the correct format for system calls in assembler and high level languages.
2. Select appropriate system calls for use in a user written program.
3. Create and manage files and directories through the use of system calls.
4. Read and write disk and tape files using Block I/O system calls.
5. Share memory and data among processes using Shared Files.
6. Access files using Record I/O system calls.
7. Create, manage, and terminate processes using system calls.
8. Send and receive inter-process communications.
9. Implement multitasking programs.
10. Synchronize tasks using intertask communication.

COURSE TOPICS

- Format of system calls
- File System
- Block I/O
- Shared Files
- Record I/O
- Processes
- Inter-Process Communication
- Multitasking

S309VS AOS/VS SYSTEM PROGRAMMING



exception condition.

7. Describe how the programmer can handle expected errors (exceptions).

8. State the purpose of system call Packets.

9. Show the correct format, using parametric encoding, for system call packets in assembler language.

10. Describe when to use system calls instead of statements in high level language programs.

11. List the system call support features available for selected high level languages.

12. Associate available support features with their functions.

13. Show the correct format for system calls in selected high level languages.

14. Describe how the programmer can handle expected errors (exceptions) in high level language system calls.

15. Show the correct format for system call packets in selected high level languages.

Outline

- Introduction
  - What is a system call?
- Format of system calls
  - Assembler language format
    - Components of system call format
      - system call macro
      - normal return
      - exception return
    - Origin of system calls
      - SYSID
      - PARU
      - MASM.PS
    - Accumulator usage
    - System call exceptions (errors)
      - Handling expected errors
    - System call packets
      - Handling double word items
  - System calls from high level languages
    - When to use system calls instead of statements
    - System call support features and their functions
      - System call symbol support
      - SYS function
      - BYTEADDR, WORDADDR functions
    - System call exceptions (errors)
      - Handling expected errors
    - System call packets

Purpose of System Calls

A system call is a request to the operating system to perform some action on behalf of a user program.

Types of System Calls

- File Management
- Block IO
- Shared File IO
- Record IO
- Memory Management
- Multitasking
- Inter-process Communication
- Connection Management
- Process Management
- User Device
- Miscellaneous

Format of System Calls

Assembler language format example:

```
?CALL       [packet-address]    ; System call
WBR         error-address       ; Error return
instruction                     ; Normal return
```

System call format components:

- System call
  - Invocation of a macro.
  - Packet address, when used, may be an argument to the call, or placed in AC2.
- Error return
  - Must be a ONE-WORD instruction.
  - Executed if system call encounters an error (exception condition).
- Normal return
  - Next instruction to be executed if system call succeeds.

Origin of system calls:

- MASM.PS
  - Assembler's permanent symbol table.
  - Built from SYSID and PARU.
- SYSID.32.SR
  - Defines system call macros.
- PARU.32.SR
  - Defines symbols for constants used with system calls.

Result of assembling a system call:

```
LLEF        2, packet-address
XJSR        @6                  ; call to SCALL
nnn                             ; system call word
```

- SCALL
  - System call handler routine.
  - Loaded by LINK from URT32.LB.
  - LINK stores SCALL address in location 6.
- System call word
  - Number identifying specific system call.
  - Passed as in-line argument to SCALL.

Accumulator Usage

Input to system call:

- AC0 As defined for specific call.
- AC1 As defined for specific call.
- AC2 Packet address, when used, or as defined.
- AC3 XJSR @6 places return address in AC3.

Output from system call:

- AC0 On success, as defined by specific call.
  On error return, AC0 contains error code.
- AC1 As defined by specific call.
- AC2 Packet address, when used, or as defined.
- AC3 Returns frame pointer value.

Unexpected errors:

Error return must be ONE-WORD instruction.

WBR one-word, but has limited range.

DERR n 'Detected Error' instruction: Pushes address of DERR instruction onto stack, then pushes 'n' (0 <= n <= 31). Jumps through the one word pointer that the programmer has placed in location 47 to his error handler routine.

'ERRXT' Error exit routine written for this course, to be linked with all lab programs. Designed to be invoked with DERR instruction.

The ERRXT module includes a one word pointer in location 47. Since a one word pointer can only address the first 32 KW, a bridge in the ZREL partition is used to LJMP to the actual error handler routine.

```
            .LOC    47
            .WORD   ERRBRG

            .ZREL
ERRBRG:     LJMP    ERRXT
```

CLI output resulting from ERRXT:

```
** ABORT **
error text from error code
AT LOCATION symbol+offset
DERR CODE: n
AC0: nnnnnn
AC1: nnnnnn
AC2: nnnnnn
AC3: nnnnnn
Error: from program
X, program ...
```

Expected errors:

```
ERRCK:      WSEQI   error-code,0
            DERR    n
            WBR     OK
```

System Call Packets

- Used with some system calls.
- Array containing some single and some double words.
- Offsets have symbolic names.
- On input to call, specifies how call is to be performed.
- Reserved entries MUST be zero.
- On output from call, receives results.

Parametric encoding method:

```
PKT:        .LOC PKT+offset      ; offset purpose
            .WORD value          ; value description
            .LOC PKT+offset      ; offset purpose
            .DWORD value         ; value description
              ..
              ..
              ..
            .LOC PKT+pkt_length
```

- Symbols for packet offsets, lengths, and value options are defined in PARU. They start with '?'.
- Use .WORD for single word values.
- Use .DWORD for double word values.
- Some value items are used to select options:

```
.LOC PKT+offset
.WORD option1+option3+option7
```

- Enhances readability.
- Ensures compatibility with future revisions.
- Save retyping by copying packets from old programs with editor.

System Calls in High Level Languages

When to use them ?

- Compilers produce system calls needed to implement built-in statement features such as I/O.
- Features provided by system calls may be available as runtime routines.
- Use system calls directly when no statement or runtime routine has the features you need.

FORTRAN 77 System Call Support

SUPPORT FEATURE	SUPPORT FUNCTION
• F77BUILD_SYM utility	Converts system call symbol definitions into F77 INCLUDE files.
• ISYS function	Can be used to make any system call.
• WORDADDR function	Returns address of any item needed for system call.
• BYTEADDR function	Returns bytepointer to any text item needed for system call.

F77BUILD_SYM Utility

Using the F77BUILD_SYM utility:

- Make assembled listing files from SYSID and PARU.

  X, MASM/NOPS/L=(SYSID,PARU).32.LS, (SYSID,PARU).32

- Execute F77BUILD_SYM utility.

  X, F77BUILD_SYM

- Reads SYSID.32.LS and PARU.32.LS.

- Creates QSYM.F77.IN.

- Can make shorter include files using symbol list file:

  - Reduces program compilation time.
  - Ordinary text file.
  - List all sysmbols required:
    - System calls.
    - Packet offsets.
    - Option values.
    - Expected error codes.
  - List symbols in assembler format, one symbol per line.

  - **X, F77BUILD_SYM, symbol_list_file**

  - Rename QSYM.F77.IN to reflect contents.

F77BUILD_SYM utility results:

SYMBOL	BECOMES
?XYZ	ISYS_XYZ
?XY.Z	ISYS_XY_Z
ERXYZ	ISYS_ERXYZ

F77 System Call Format

- Include statement(s) for symbol file(s)
- Packets declared as arrays of single word (2 byte) integers.
- Packet array boundaries must be zero to packet length minus one.
- Declare EQUIVALENCE names as 4 byte integers for each double word packet item.
- Initialize packet values with assignment statements or data statements.
- Address and bytepointer packet items must be initialized with assignment statements using WORDADDR or BYTEADDR functions.
- Reserved packet items MUST be zero.
- Packets may be placed in COMMON and data statements may be in BLOCK DATA SUBPROGRAM if desired.
- Choose double word (4 byte) integer variable names to take the roles of AC0, AC1, and AC2.
- Choose double word integer variable to receive result code.
- Make system call using ISYS function.
- Check result code for success (zero) or exception (error code).
- To test for expected errors, compare result code with error code symbolic constant.
- Can use CALL EXIT(error_code) for unexpected errors.

F77 System Call Format

```
      PROGRAM SYSTEM_CALLER

%LIST(OFF)
%INCLUDE "XYZ.QSYM.F77.IN"
%LIST(ON)

      INTEGER*2       PKT(0:ISYS_pkt_length-1)
      INTEGER*4       PKT_DBLITEM
      EQUIVALENCE (PKT_DBLITEM,PKT(ISYS_DBLITEM))

      PKT(ISYS_ABC) = ISYS_option7+ISYS_option3
          ..
          ..
      PKT_DBLITEM   = WORDADDR(buffer)
          ..
          ..
      IAC0 = BYTEADDR('text_string<0>')
      IAC1 = 0
      IAC2 = WORDADDR(PKT)
      ICODE = ISYS(ISYS_XYZ,IAC0,IAC1,IAC2)
      IF (ICODE.EQ.ISYS_ERXYZ) GOTO 123
      IF (ICODE.NE.0) CALL EXIT(ICODE)
          ..
          ..
```

C Example of initializing packet using data statements:

```
      DATA PKT(ISYS_offset) /value/
```

PASCAL SYSTEM CALL SUPPORT

SUPPORT FEATURE	SUPPORT FUNCTION
• BUILD_SYM utility	Converts system call symbol definitions into PASCAL or F77 INCLUDE files.
• SYS function	Can be used to make any system call.
• WORDADDR function	Returns address of any item needed for system call.
• BYTEADDR function	Returns bytepointer to any text item needed for system call.

Pascal BUILD_SYM Utility

Using the BUILD_SYM utility:

- Make assembled listing files from SYSID and PARU.

  X, MASM/NOPS/L=(SYSID,PARU).32.LS, (SYSID,PARU).32

- Execute BUILD_SYM utility.

  PASBUILD

  or X, BUILD_SYM/LANG=PASCAL

- Reads SYSID.32.LS and PARU.32.LS.

- Creates QSYM.PAS.IN.

- Can make shorter include files using symbol list file:

  - Reduces program compilation time.
  - Ordinary text file.
  - List all sysmbols required:
    - System calls.
    - Option values.
    - Expected error codes.
  - List symbols in assembler format, one symbol per line.
  - NOTE that packet offset symbols are not used.

  - **PASBUILD, symbol_list_file**

  - Rename QSYM.PAS.IN to reflect contents.

BUILD_SYM utility results:

SYMBOL	BECOMES
?XYZ	SYS_XYZ
?XY.Z	SYS_XY_Z
ERXYZ	SYS_ERXYZ

Pascal System Call Format

- Use INCLUDE statement(s) for symbol file(s). Surround with list off directives to keep listing brief.
- Declare packets as structures with members in exact order defined in PARU.32.SR.
- Member names are not offsets, so this method is non-parametric.
- Single word system call items may be declared as SHORT_INTEGER.
- Double word system call items may be declared as LONG_INTEGER.
- Address and bytepointer packet items must be initialized with assignment statements using WORDADDR or BYTEADDR functions.
- Reserved packet items MUST be zero.
- Make system call using SYS function in IF statement. SYS returns TRUE on exception conditon.

Pascal System Call Format

```
PROGRAM system_caller(input,output);

{$C+}           { allow ASCII codes in angle brackets }
{$I-}
%INCLUDE 'QSYM.PAS.IN';
{$I+}

VAR
    buff: PACKED ARRAY[1..512] OF CHAR;
    fname:PACKED ARRAY[1..6]   OF CHAR;

    ac0,ac1,ac2 : LONG_INTEGER;

    pkt: RECORD
        item0:  SHORT_INTEGER;
        item1:  SHORT_INTEGER;
        item2:  LONG_INTEGER;
        item3:  LONG_INTEGER
    END;

    pkt.item0:= 512;
    pkt.item1:= SYS_option7+SYS_option3;
    pkt.item2:= 0;
    pkt.item3:= WORDADDR(buff);


    ..
    ..
fname: = 'FILE<0>';
    ..
ac0:= BYTEADDR(fname);
ac1:= 0;
ac2:= WORDADDR(pkt);
IF SYS(SYS_XYZ,ac0,ac1,ac2) THEN
    IF ac0 <> SYS_ERXYZ THEN
        exit_routine(ac0);

    ..
    ..
```

C System Call Support

SUPPORT FEATURE	SUPPORT FUNCTION
• packets directory	Contains INCLUDE files which declare packets as structures.
• SYS function	Can be used to make any system call.
• SYS_MNEMONIC functions	Provides individual function for each system call.
• & operator	Returns address of any item needed for system call.
• char * declaration	Declares bytepointer to any text item needed for system call.

C System Call Format

- Use INCLUDE statement(s) for packet file(s). Surround with NOLIST and LIST statements to keep the listing brief.

- Packet INCLUDE files automatically cause inclusion of SYSID.H and PARU.H. Use INCLUDE statements for SYSID.H and PARU.H only if your program needs no packet include files.

- Declare packet variables using type name found in include file (upper case letters).

- For constants such as system call names, packet items, and error codes, use the same spelling as the assembly language form of the name (upper case), but with $ instead of the question mark, if any. Note the error codes have no question mark and thus no $.

- Declare text string items as char *.

- Address and bytepointer packet items must be initialized with assignment statements using the & operator or a char * variable.

- Reserved packet items MUST be zero.

- Make system call using SYS function. SYS requires as arguments the system call symbol and pointers to integer variables to represent the accumulators. SYS returns a non-zero error code on exception conditon.

- Or, make the system call using one of the SYS_MNEMONIC functions. They require as arguments integer variables for the accumulator input values and pointers to integer variables to receive the accumulator output values. When the SYS_MNEMONIC functions are used, the program must be linked with CSYSCALLS.LB.

C System Call Format

```
#nolist
#include <stdio.h>
#include <packets:xyz.h>
#list

main()
{
    char buff[512];
    char *fname;

    int ac0,ac1,ac2;

    P_XYZ pkt;

    pkt.item0 = 512;
    pkt.item1 = $OPTION7+$OPTION3;
    pkt.item2 = 0;
    pkt.item3 = &buff;


    ..
    ..
fname = "FILE";
    ..
ac0 = fname;
ac1 = 0;
ac2 = &pkt;
if ((error = sys ($XYZ,&ac0,&ac1,&ac2)) != 0 &&
     error != ERXYZ) exit_routine(error);

    ..
    ..
```

PL/1 System Call Support

SUPPORT FEATURE	SUPPORT FUNCTION
• PL1SYSID.IN, PL1PARU.IN	System call symbol definition INCLUDE files.
• SYS function	Can be used to make any system call.
• ADDR function	Returns address of any item needed for system call.
• BYTEADDR function	Returns bytepointer to any text item needed for system call.

PL/1 System Call Format

- Use INCLUDE statements for PLISYSID.IN and PLIPARU.IN. Compile with /NOINCLUDES to keep listing brief.
- Declare packets as structures with members in exact order defined in PARU.32.SR.
- Member names are not offsets, so this method is non-parametric.
- Single word system call items may be declared as FIXED BINARY (15) ALIGNED BIT (16), or UNIONS of these types.
- Double word system call items may be declared as FIXED BINARY (31), POINTER, ALIGNED BIT (32), structures of two single word items, or UNIONS of these types.
- Address and bytepointer packet items must be initialized with assignment statements using ADDR, (not WORDADDR), or BYTEADDR functions.
- Reserved packet items MUST be zero.
- Make system call using SYS function in IF statement. SYS returns TRUE on exception conditon.
- To test for expected errors, compare low order 16 bits of AC0 variable with error code symbolic constant.
- Can improve compilation time by making /SAVEDECLARES file and using it via /DECLARES= or /SYMLIB switch during compilation.

PL/1 System Call Format

```
%INCLUDE "PL1PARU.IN";
%INCLUDE "PL1SYSID.IN";

system_caller: PROCEDURE;

    DECLARE buff    ALIGNED CHARACTER(512);
    DECLARE fname   CHARACTER(32);

    DECLARE 1 ac0           UNION,
              2 ac0_ptr           POINTER,
              2 ac0_abit      ALIGNED BIT (32),
              2 ac0_words,
                3 ac0_left        ALIGNED BIT (16),
                3 ac0_right       ALIGNED BIT (16);
    DECLARE ac1         FIXED BINARY (31);
    DECLARE ac2         POINTER;

    DECLARE 1 pkt,
              2 item0     FIXED BINARY (15),
              2 item1     ALIGNED BIT (16),
              2 item2     FIXED BINARY (31),
              2 item3     UNION,
                3 item3_ptr       POINTER,
                3 item3_bin       FIXED BINARY (31);

    item0 = 512;
    item1 = SYS_option7+SYS_option3;
    item2 = 0;
    item3_ptr = ADDR(buff);

/* At some other point in the program, item3 can be set to -1, if
needed, by using the name item3_bin. */
    ..
    ..
fname = "FILE<NUL>"C;
    ..
ac0_ptr = BYTEADDR(fname);
ac1 = 0;
ac2 = ADDR(pkt);
IF SYS(SYS_XYZ,ac0_ptr,ac1,ac2) THEN
    IF ac0_right ^= SYS_ERXYZ THEN
        CALL exit_routine(ac0_abit);

    ..
    ..
```

MODULE 2

FILE SYSTEM

Objectives

Upon completion of this module, the student should be able to:

1. Define the following terms related to File Structure: DISK BLOCK, DATA ELEMENT, INDEX BLOCK, and INDEX LEVEL.
2. Describe the structure of disk files.
3. Define DIRECTORY.
4. List and contrast the two types of directories.
5. List important items of file information contained in a DIRECTORY ENTRY.
6. Identify the items of the ?CREATE System Call Packet.
7. Use ?CREATE to create a file.
8. Delete a file using ?DELETE.
9. Describe the various mechanisms used in referencing files.
10. Describe the File Access Control features.
11. List System Calls related to File Access Control, inlcuding their functions.
12. Obtain filestatus information with system calls.
13. Expand filename templates with system calls.

Outline

- File System
    - File Structure
        - Disk Blocks
        - Data Elements
        - Index Blocks
        - Indexing Levels
    - Directories
        - File Entry Information
            - Filename
            - Access Control List
            - Dates
            - File Type
            - File Length
            - File Disk Address
            - Current and Maximum Index Levels
        - Control Point Directories
    - Referencing Files
        - Working Directory
        - Searchlist
        - Pathnames
            - Pathname Prefixes
    - Creating Files
    - Deleting Files
    - File Access Control
    - Obtaining Filestatus Information
    - Expanding Filename Templates

- DIRECTORY A file whose contents describes other files

- DIRECTORY ENTRY A collection of information about a file:

  - filename
  - access control list
  - permanence attribute
  - element size
  - maximum and current index levels
  - file first logical disk address
  - logical byte length

- CONTROL POINT DIRECTORY Same as regular directory except has maximum space limit plus current space used, in disk blocks

File Referencing

- Current Working Directory
- Searchlist
- Pathnames
  - Pathname Prefixes
    : (root)
    @ (peripheral)
    ^ (superior directory)
    = (current directory)

System Calls for Controlling File Referencing

- ?DIR Set current working directory
  - AC0 - B.P. to directory name or 0 for DIR/I
  - AC1 - reserved
  - AC2 - reserved
- ?GLIST Returns current searchlist
  - AC0 - reserved
  - AC1 - B.P. to buffer
  - AC2 - buffer byte length
- ?SLIST Set searchlist
  - AC0 - reserved
  - AC1 - reserved
  - AC2 - B.P. to searchlist string

Deleting a File

- ?DELETE

  - AC0 - B.P. to file name or 0 if packet used
  - AC1 - Reserved
  - AC2 - 0 if no packet used else packet address

Use ?DELETE with file name and no packet or, for a file that you have currently open, with packet containing channel number and no file name.

- DISK BLOCK 512 bytes

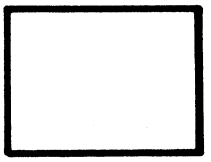
- DATA ELEMENT One or more contiguous disk blocks

- ELEMENT SIZE Number of disk blocks per data element in a file

- INDEX BLOCK One disk block - can point to up to 128 data elements or next level index blocks

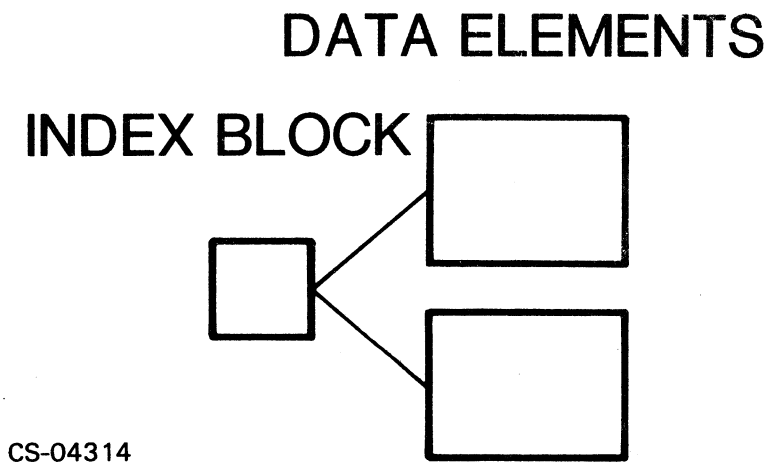
- INDEX LEVEL A file can have from zero to three levels of index blocks

DATA ELEMENT



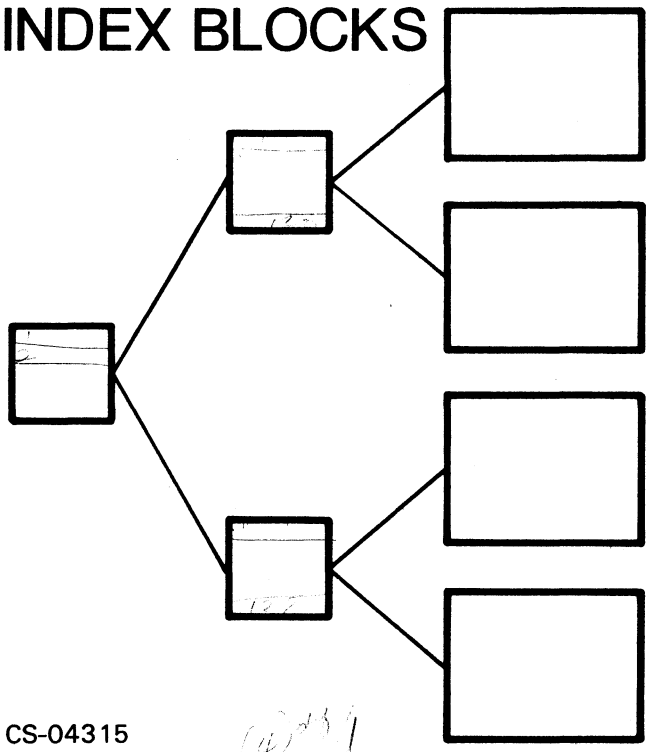
CS-04313

1-LEVEL INDEXING



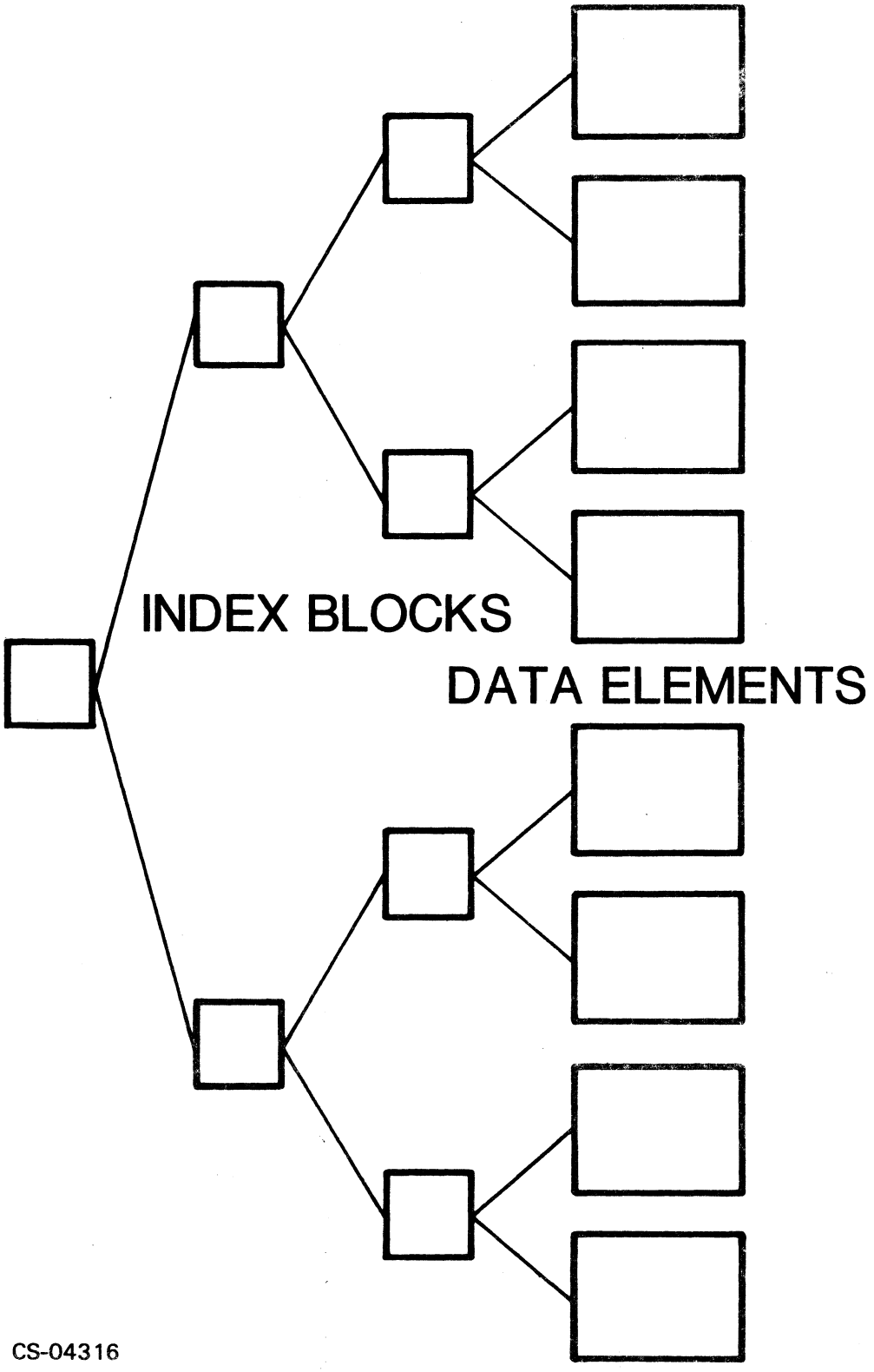
2-LEVEL INDEXING

DATA ELEMENTS



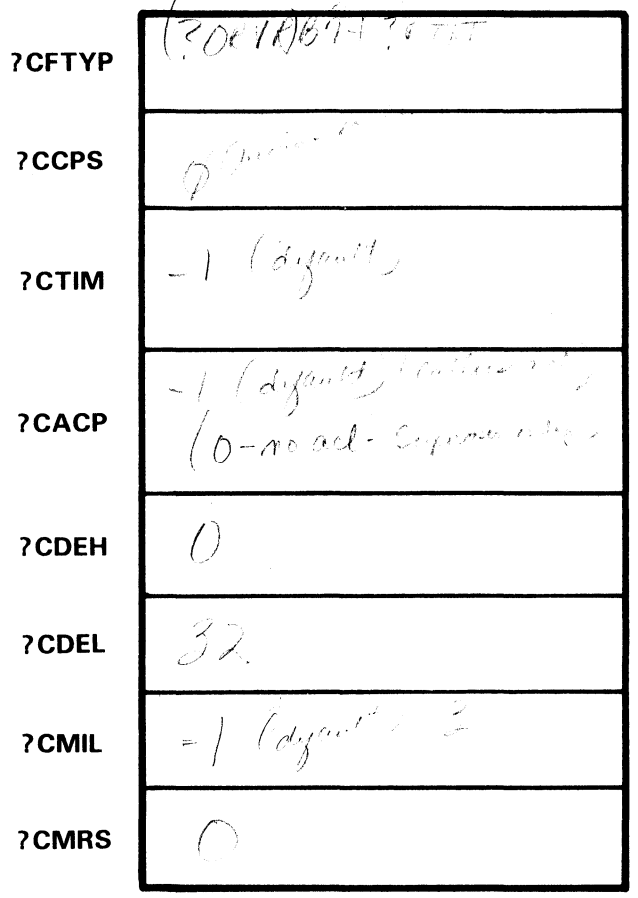
CS-04315

3-LEVEL INDEXING



CS-04316

?CREATE PACKET



CS-04317

```
          .TITL   CREATE
          .EXTL   ERRXT

          .NREL   1

START:    LLEFB   0,FILE*2      ; NAME OF FILE
          WSUB    1,1
          WSUB    2,2
          ?DELETE               ; ATTEMPT TO DELETE THE FILE
          WBR     DLERR

CREATE:   LLEFB   0,FILE*2      ; NAME OF FILE
          WSUB    1,1
          ?CREATE CREPK         ; THEN CREATE THE FILE
          DERR    2

DONE:     WSUB    2,2
          ?RETURN
          DERR    3


;  ROUTINE TO CHECK ?DELETE ERROR

DLERR:                          ; GOT A ?DELETE ERROR
          WSNEI   ERFDE,0       ; WAS IT FILE DOES NOT EXIST ?
          WBR     CREATE        ; YES. IGNORE THE ERROR
          DERR    1             ; NO. TERMINATE WITH ERROR.

FILE:     .TXT    'FILE'

          .NREL   0

          ; PARAMETER PACKET FOR FILE CREATION

CREPK:    .LOC    CREPK+?CFTYP; RECORD FORMAT AND FILE TYPE
          .WORD   (?ORFX)B7+?FTXT ; FIXED, TEXT
          .LOC    CREPK+?CCPS     ; RECORD SIZE FOR FIXED
          .WORD   80
          .LOC    CREPK+?CTIM     ; ADDR OF TIME BLOCK
          .DWORD  -1              ; CURRENT TIME
          .LOC    CREPK+?CACP     ; ACL BYTE PTR
          .DWORD  -1              ; USE DEFACL
          .LOC    CREPK+?CDEH     ; (RESERVED)
          .WORD   0
          .LOC    CREPK+?CDEL     ; ELEMENT SIZE
          .WORD   32.
          .LOC    CREPK+?CMIL     ; MAX INDEX LEVEL
          .WORD   -1              ; USE SYSTEM DEFAULT
          .LOC    CREPK+?CMRS     ; (RESERVED)
          .WORD   0
          .LOC    CREPK+?CLTH     ; ?CLTH = PKT LENGTH

          .END    START
```

```
        PROGRAM CREATE

%LIST(OFF)
%INCLUDE 'CREATE_QSYM.F77.IN'
%LIST(ON)

    INTEGER*2 CREPK(0:ISYS_CLTH-1)
    INTEGER*4 CREPK_CTIM,CREPK_CACP
    EQUIVALENCE (CREPK(ISYS_CTIM),CREPK_CTIM)
    EQUIVALENCE (CREPK(ISYS_CACP),CREPK_CACP)

C       FILE CREATION PACKET

    CREPK(ISYS_CFTYP) = ISYS_ORFX*256+ISYS_FTXT     ! FIXED, TEXT
    CREPK(ISYS_CCPS)  = 80       ! FILE CONTROL PARAMS FIXED SZ
    CREPK_CTIM = -1              ! DEFAULT TIME
    CREPK_CACP = -1              ! DEFAULT ACL
    CREPK(ISYS_CDEH)  = 0        ! RESERVED
    CREPK(ISYS_CDEL)  = 32       ! ELEMENTSIZE
    CREPK(ISYS_CMIL)  = -1       ! DEFAULT MAXIMUM INDEX LEVELS
    CREPK(ISYS_CMRS)  = 0        ! RESERVED

    IAC0 = BYTEADDR('FILE<0>')   ! DELETE 'FILE' IF IT EXISTS
    IAC1 = 0
    IAC2 = 0
    IER = ISYS(ISYS_DELETE,IAC0,IAC1,IAC2)
    IF (IER.NE.0.AND.IER.NE.ISYS_ERFDE) CALL EXIT(IER)

    IAC0 = BYTEADDR('FILE<0>')       ! CREATE 'FILE'
    IAC1 = 0
    IAC2 = WORDADDR(CREPK)
    IER = ISYS(ISYS_CREATE,IAC0,IAC1,IAC2)
    IF (IER.NE.0) CALL EXIT(IER)

    END
```

```
PROGRAM create(input,output);

{$C+}
{$I-}
%INCLUDE 'CREATE_QSYM.PAS.IN'
{$I+}

VAR
        ac0,ac1,ac2 : LONG_INTEGER;
        fname : PACKED ARRAY [1..6] OF CHAR;
        crepk : RECORD
                cftyp : SHORT_INTEGER;
                ccps :  SHORT_INTEGER;
                ctim :  LONG_INTEGER;
                cacp :  LONG_INTEGER;
                cdeh :  SHORT_INTEGER;
                cdel :  SHORT_INTEGER;
                cmil :  SHORT_INTEGER;
                cmrs :  SHORT_INTEGER
        END;

        PROCEDURE exit(code:LONG_INTEGER);
                VAR ac1,ac2 : LONG_INTEGER;
        BEGIN
        ac1 := 0;
        ac2 := SYS_RFCF+ SYS_RFEC+ SYS_RFAB;
        IF SYS(SYS_RETURN,code,ac1,ac2) THEN
        END;    { of procedure exit }

BEGIN
        crepk.cftyp := SYS_ORFX*256+SYS_FTXT;   { FIXED, TEXT }
        crepk.ccps  := 80;      { FILE CONTROL PARAMS FIXED SZ }
        crepk.ctim  := -1;      { DEFAULT TIME }
        crepk.cacp  := -1;      { DEFAULT ACL }
        crepk.cdeh  := 0;       { RESERVED }
        crepk.cdel  := 32;      { ELEMENTSIZE }
        crepk.cmil  := -1;      { DEFAULT MAXIMUM INDEX LEVELS }
        crepk.cmrs  := 0;       { RESERVED }

        fname := 'FILE<0>';
        ac0 := BYTEADDR(fname); { DELETE 'FILE' IF IT EXISTS }
        ac1 := 0;
        ac2 := 0;
        IF SYS(SYS_DELETE,ac0,ac1,ac2) THEN
                IF ac0 <> SYS_ERFDE THEN exit(ac0);
        ac0 := BYTEADDR(fname);
        ac1 := 0;
        ac2 := WORDADDR(crepk);
        IF SYS(SYS_CREATE,ac0,ac1,ac2) THEN exit(ac0);
END.
```

```
#nolist
#include <stdio.h>
#include <packets:create.h>
#list

main()
{
    P_CREATE     filepkt;          /* packet for file creation   */

    int ac0, ac1, ac2, error;      /* accumulators and error var */
    char *fname;                   /* filename byte pointer */

/* FILE CREATION PACKET */

    filepkt.cftyp_format = $ORFX; /* record format fixed */
    filepkt.cftyp_entry  = $FTXT; /* text type file */
    filepkt.ccps         = 80;     /* fixed record length */
    filepkt.ctim         = -1;     /* default times */
    filepkt.cacp         = -1;     /* default ACL */
    filepkt.cdeh         = 0;      /* reserved */
    filepkt.cdel         = 32;     /* element size */
    filepkt.cmil         = -1;     /* default max index lvls */
    filepkt.cmrs         = 0;      /* reserved */

/* ATTEMPT TO DELETE THE FILE, */
/* IGNORE THE ERROR IF IT DOESN'T EXIST */

    fname   = "FILE";         /* B.P. to filename */
    ac0     = fname;          /* ac0 <- B.P. */
    ac1     = 0;              /* reserved */
    ac2     = 0;              /* reserved */
    if ((error = sys ($DELETE,&ac0,&ac1,&ac2)) != 0 &&
         error != ERFDE) errxt(error);

/* CREATE THE FILE */

    fname   = "FILE";           /* B.P. to filename */
    ac0     = fname;            /* ac0 <- B.P. */
    ac1     = 0;                /* reserved */
    ac2    = &filepkt;          /* ac2 <- W.P. to create pkt */
    if (error = sys($CREATE,&ac0,&ac1,&ac2)) errxt(error);
} /* end main */


errxt(e)                  /* FUNCTION ERRXT */
  int e;
{
  int a0, a1, a2;
    a0      = e;
    a1      = 0;
    a2      = $RFCF + $RFEC + $RFAB;
    sys ($RETURN, &a0, &a1, &a2);
} /* end errxt */
```

```
create: PROCEDURE;
%INCLUDE "PL1SYSID.IN";
%INCLUDE "PL1PARU.IN";
    DECLARE 1 ac0             UNION,
              2 ac0_ptr       POINTER,
              2 ac0_abit      ALIGNED BIT (32),
              2 ac0_words,
                3 ac0_left    ALIGNED BIT (16),
                3 ac0_right   ALIGNED BIT (16);
    DECLARE 1 ac2             UNION,
              2 ac2_ptr       POINTER,
              2 ac2_bin       FIXED BINARY(31);
    DECLARE ac1               FIXED BINARY (31);
    DECLARE fname             CHARACTER(6);
    DECLARE 1 crepk,
              2 cftyp         FIXED BINARY (15),
              2 ccps          FIXED BINARY (15),
              2 ctim          UNION,
                3 ctim_ptr    POINTER,
                3 ctim_bin    FIXED BINARY (31),
              2 cacp          UNION,
                3 cacp_ptr    POINTER,
                3 cacp_bin    FIXED BINARY (31),
              2 cdeh          FIXED BINARY (15),
              2 cdel          FIXED BINARY (15),
              2 cmil          FIXED BINARY (15),
              2 cmrs          FIXED BINARY (15);
    cftyp = SHIFT(SYS_ORFX,8) ! SYS_FTXT; /* FIXED, TEXT */
    ccps  = 80;               /* FILE CONTROL PARAMS FIXED SIZE */
    ctim_bin = -1;            /* DEFAULT TIME */
    cacp_bin = -1;            /* DEFAULT ACL */
    cdeh   = 0;               /* RESERVED */
    cdel   = 32;              /* ELEMENTSIZE */
    cmil   = -1;              /* DEFAULT MAXIMUM INDEX LEVELS */
    cmrs   = 0;               /* RESERVED */

    fname = "FILE<NUL>"C;
    ac0_ptr = BYTEADDR(fname); /* DELETE 'FILE' IF IT EXISTS */
    ac1 = 0;
    ac2_bin = 0;
    IF SYS(SYS_DELETE,ac0_ptr,ac1,ac2_bin) THEN
            IF ac0_right^=SYS_ERFDE THEN CALL exit(ac0_abit);

    ac0_ptr = BYTEADDR(fname);      /* CREATE 'FILE' */
    ac1 = 0;
    ac2_ptr = ADDR(crepk);
    IF SYS(SYS_CREATE,ac0_ptr,ac1,ac2_ptr) THEN
                              CALL exit(ac0_abit);

    exit:   PROCEDURE(code);
            DECLARE (code,ac0,ac1,ac2)      ALIGNED BIT (32);
            ac1 = "0"B;
            ac2 = SYS_RFCF ! SYS_RFEC ! SYS_RFAB;
            IF SYS(SYS_RETURN,code,ac1,ac2) THEN;
    END     exit;
END create;
```

File Access Control

Types of Access

- OWNER - Can delete file
- WRITE - Can modify file contents
- APPEND - Can extend file contents
- READ - Can read file contents
- EXECUTE - Can run (program file), or reference (directory)

ACL Format

Text string containing username - access pairs

- Username may be a template
- Username must end with null byte
- Access byte follows username
  - ?FACO Owner
  - ?FACW Write
  - ?FACA Append
  - ?FACR Read
  - ?FACE Execute
- ACL is terminated with null byte after last access byte

```
ACL: .TXT "USER<0><?FACO!?FACW!?FACA!?FACR!?FACE>
          +<0><?FACR!?FACE>"
```

ACL System Calls

- ?GACL - Get a file's ACL
- ?SACL - Set a file's ACL
- ?GTACP - Get caller's access privileges to a file
- ?DACL - Set or get process' current DEFACL

Obtaining Filestatus Information

- ?FSTAT

  - AC0 - Filename B.P. or channel number
  - AC1 - Flags
    Bit 0 - AC0 contains channel number
    Bit 1 - Ignore links
  - AC2 - Packet address

Expanding Filename Templates

- ?GNFN Get next filename

  - AC0 - Reserved
  - AC1 - Directory channel number (from ?GOPEN)
  - AC2 - Packet address

  packet contents

  - ?NFKY - Set to zero for first call (single word)
  - ?NFRS - Reserved (single word)
  - ?NFNM - B.P. to buffer to receive filename
  - ?NFTP - B.P. to filename template to expand

  packet length = ?NFLN

MODULE 3

BLOCK I/O

Objectives

Upon completion of this module, the student should be able to:

1. Select appropriate applications for Block I/O.
2. Use ?GOPEN, ?RDB, ?WRB, and ?GCLOSE to perform Block I/O.
3. Describe a method for correctly handling the last block of an input file.
4. Access the next allocated block of a sparse file.
5. Manage file space using ?ALLOCATE.

Outline

- Block I/O
  - When to Use Block I/O
  - ?GOPEN
  - ?RDB
  - ?WRB
  - ?GCLOSE
  - Handling the last block correctly
  - Sparse files
    - ?BLOCKIO
  - File space management
    - ?ALLOCATE

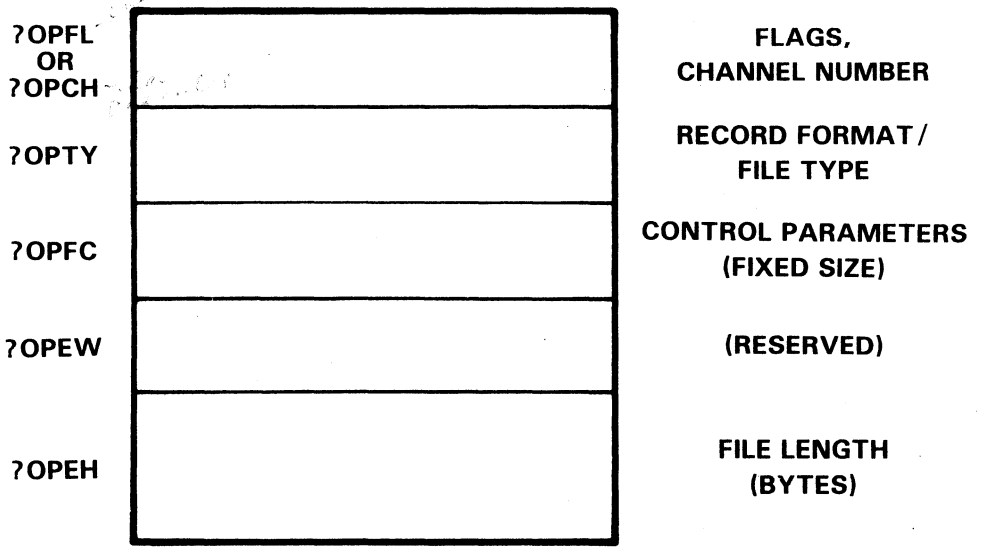
Block I/O	Record I/O
File must be on block-oriented device	Device independent
Requires actual filename	Can use generic filename
File must already exist	Has delete/create options on open
Requires separate packets for open, read/write	Uses same packet for all calls
Access must be on block boundary, length in 512 byte blocks	Offers four record formats to choose from
Program must specify file position on each access	Has internal pointer to next sequential record – can use relative or absolute positioning
Program must close file before terminating	System closes file automatically on process termination

CS-04318

?GOPEN PACKET

AC0 – B.P. to filename

AC1 – -1 = System selects channel



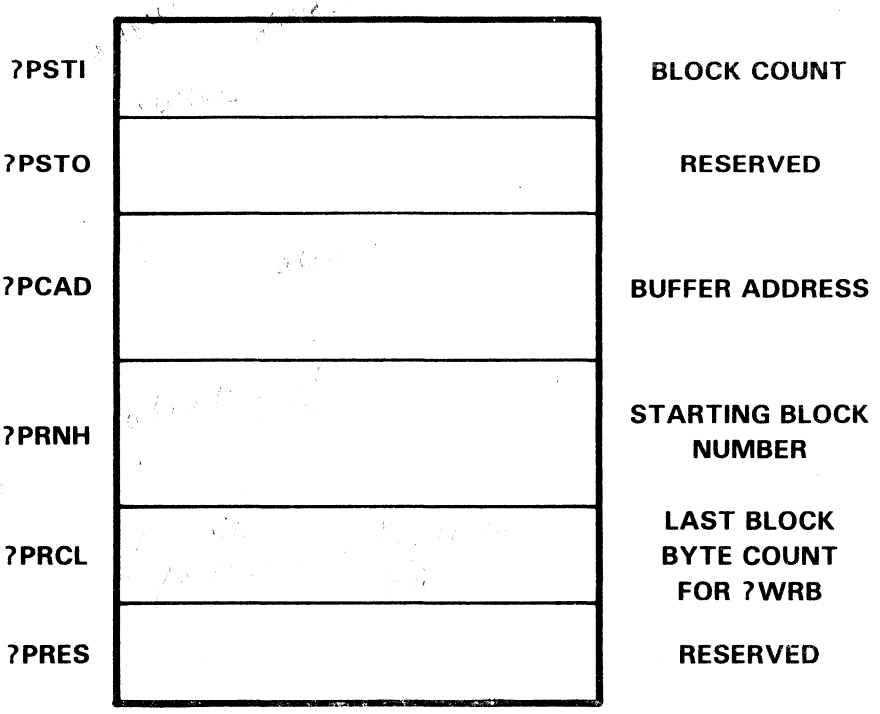
CS-04319

?RDB / ?WRB PACKET

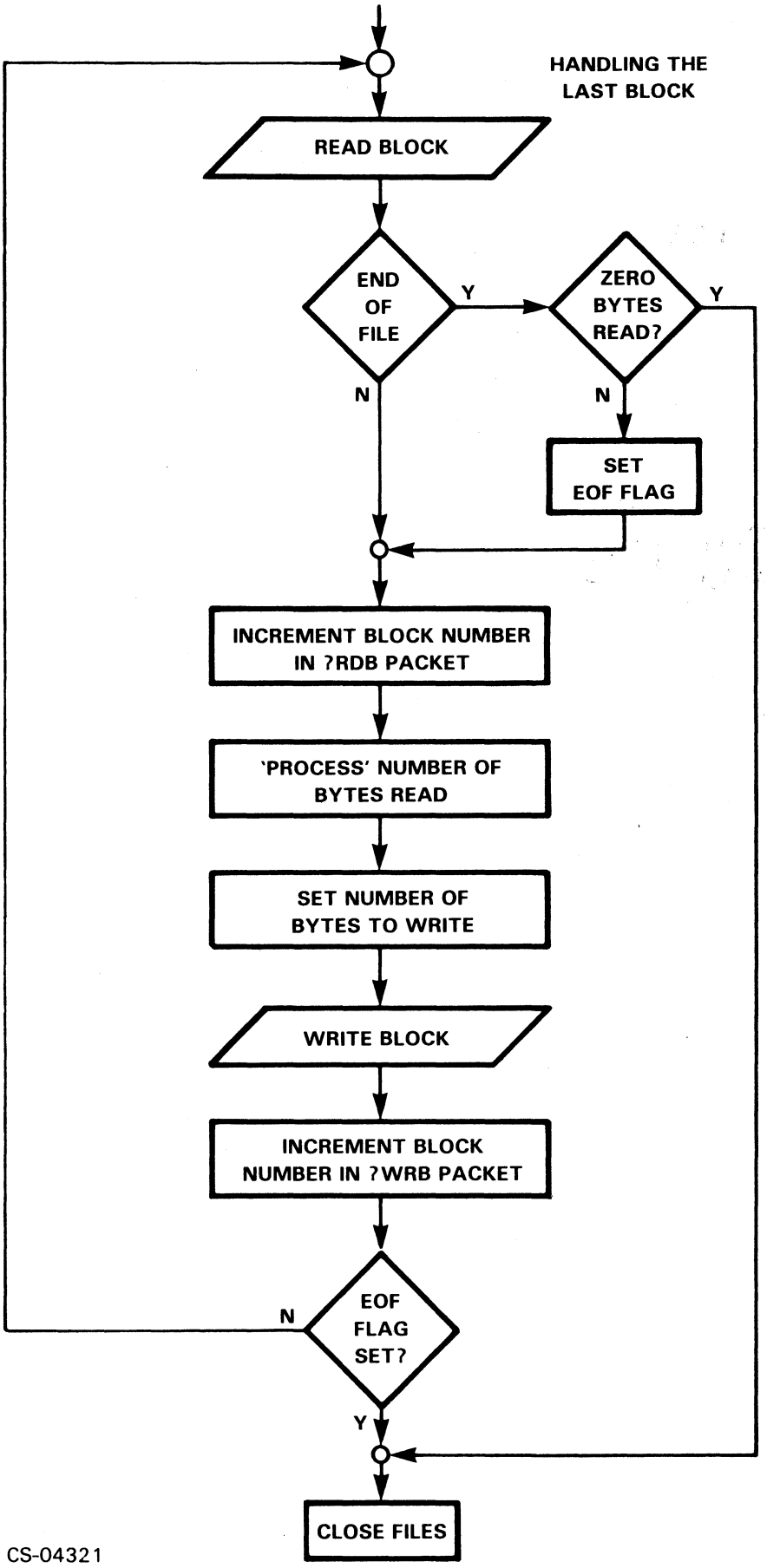
AC0 – Reserved

AC1 – Channel number

AC1 –?RDB returns number of bytes read in AC1

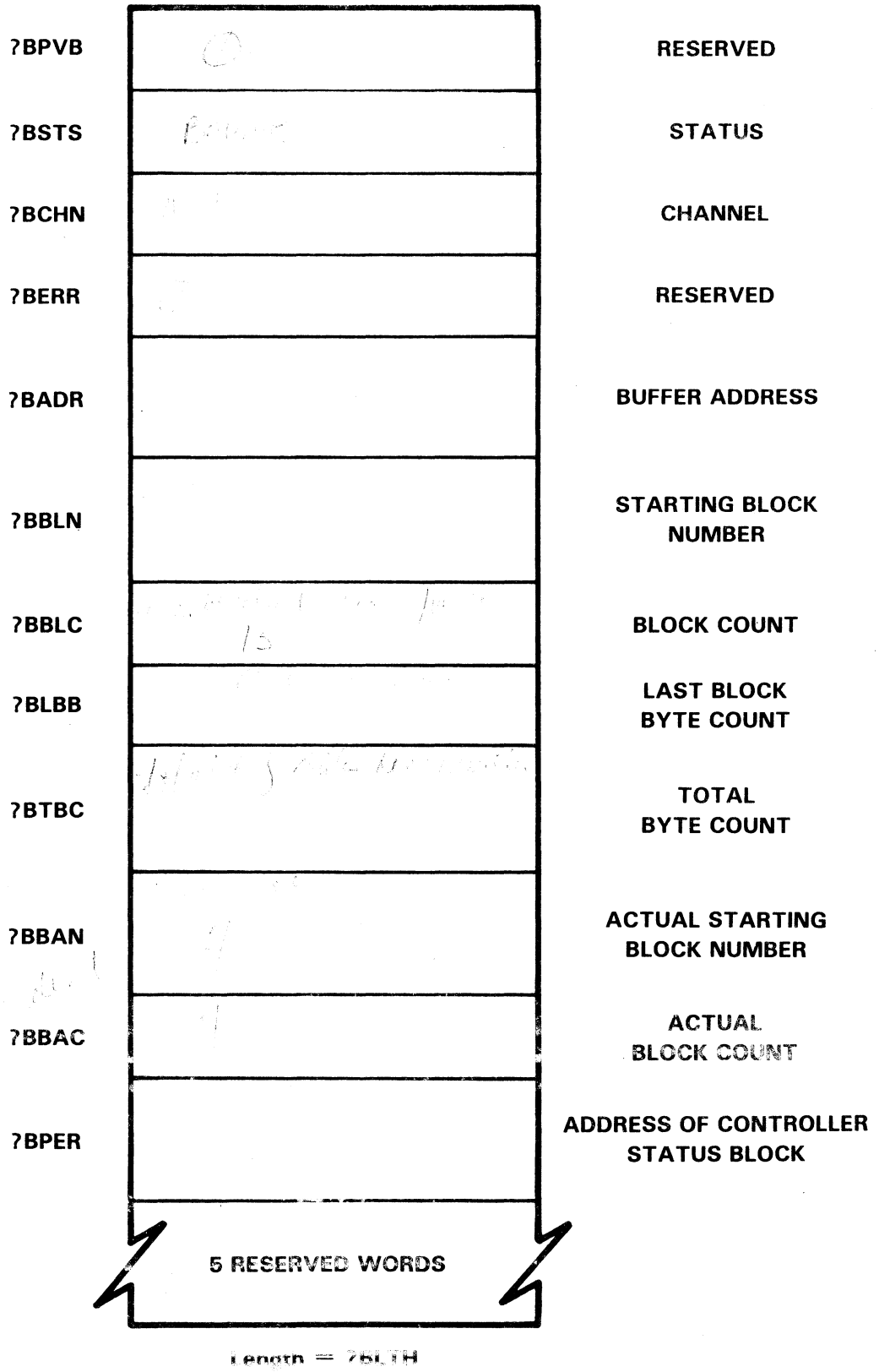


CS-04320


HANDLING THE
LAST BLOCK
READ BLOCK
END
OF
FILE
Y
ZERO
BYTES
READ?
Y
N
N
SET
EOF FLAG
INCREMENT BLOCK NUMBER
IN ?RDB PACKET
'PROCESS' NUMBER OF
BYTES READ
SET NUMBER OF
BYTES TO WRITE
WRITE BLOCK
INCREMENT BLOCK
NUMBER IN ?WRB PACKET
N
EOF
FLAG
SET?
Y
CLOSE FILES

CS-04321

?BLKIO PACKET



CS-04322

Managing File Space

- ?ALLOCATE - Allocate disk blocks to a file

  - AC0 - Reserved
  - AC1 - Channel number
  - AC2 - Address of ?RDB/?WRB packet

?ALLOCATES allocates disk blocks for the file elements specified by the block count (?PSTI) and starting block number (?PRNH) in the ?RDB/?WRB packet. It fills newly allocated blocks with zeros but leaves any already existing blocks unchanged. By using ?ALLOCATE, a program can ensure that sufficient space exists before writing to a file.

MODULE 4

SHARED FILES

Objectives

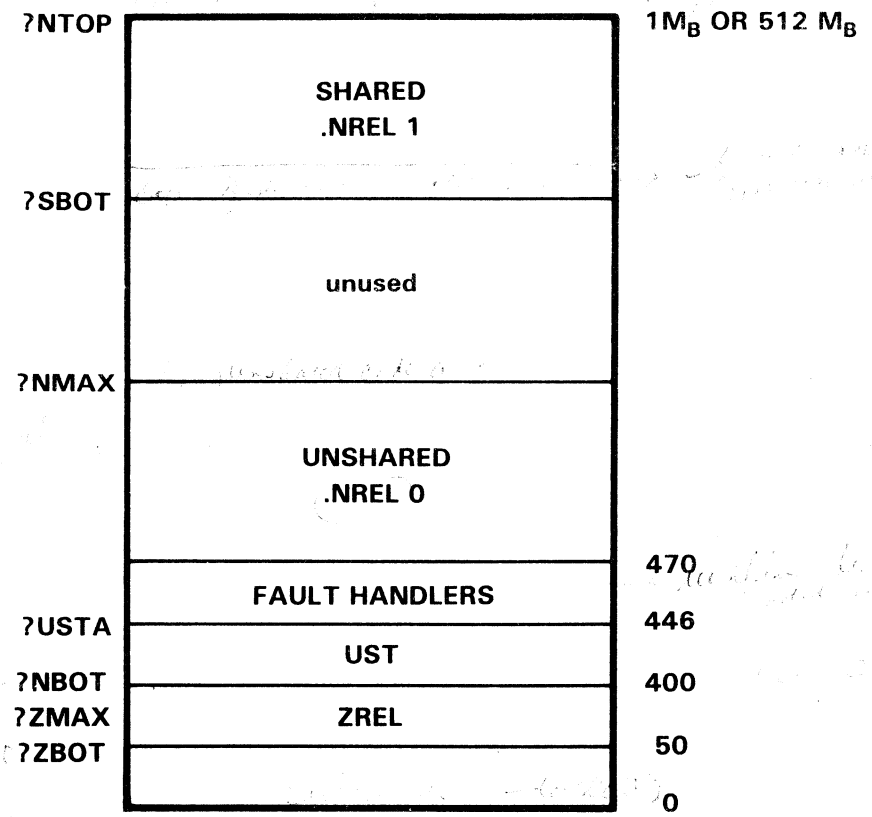
Upon completion of this module, the student should be able to:

1. List the types of Memory Partitions.
2. List the assembler directives used to specify memory partitions.
3. Describe the system's management of shared pages.
4. State the element size and block count requirements for Shared File IO.
5. Use system calls to perform Shared File IO.

Outline

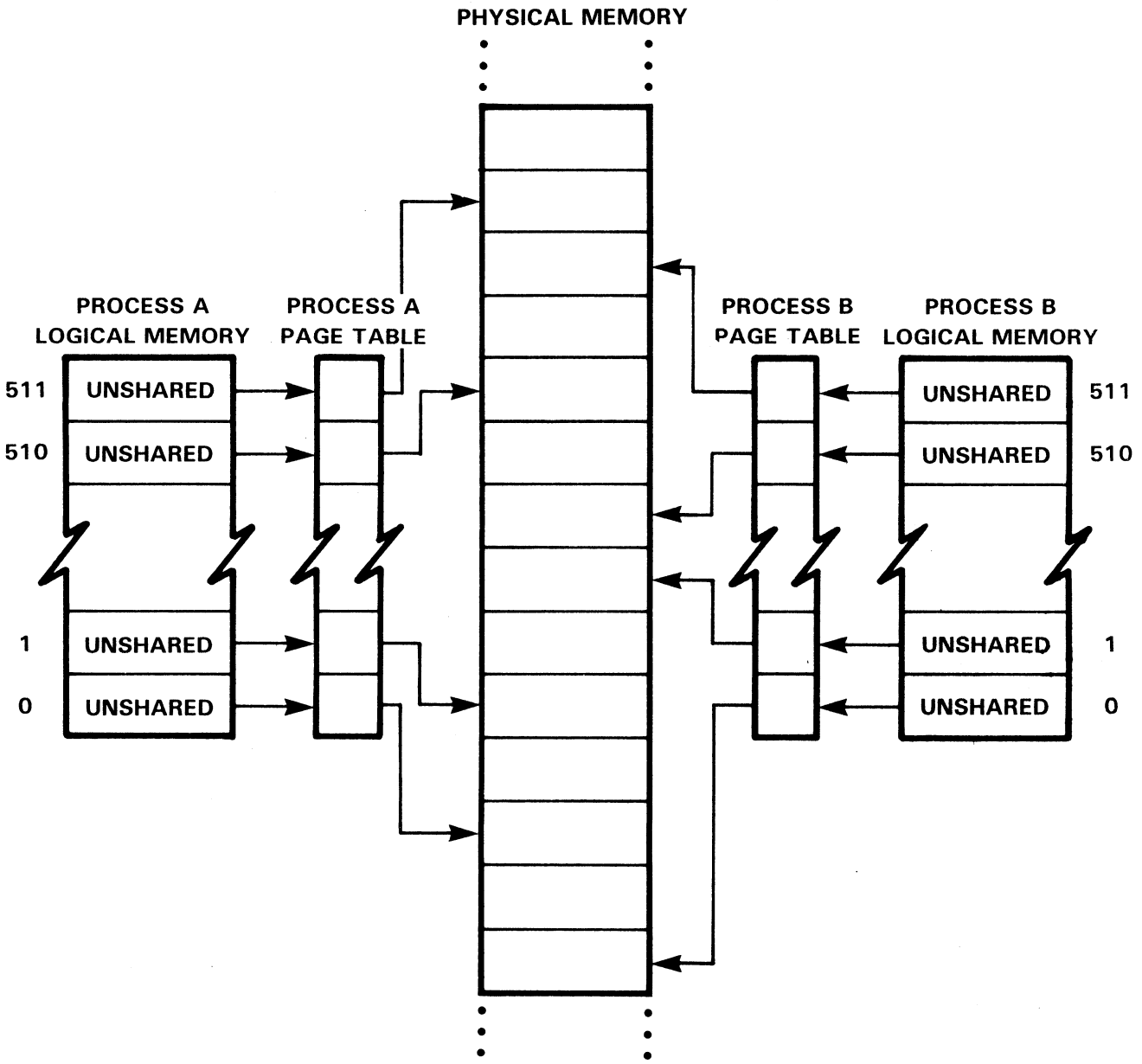
- Shared Files
  - Memory Partitions
    - ZREL
    - UNSHARED
    - SHARED CODE
    - SHARED DATA
  - Shared File I/O
    - ?SOPEN
    - ?SPAGE
      - ?SCLOSE
      - ?RPAGE
    - ?FLUSH

MEMORY PARTITIONS



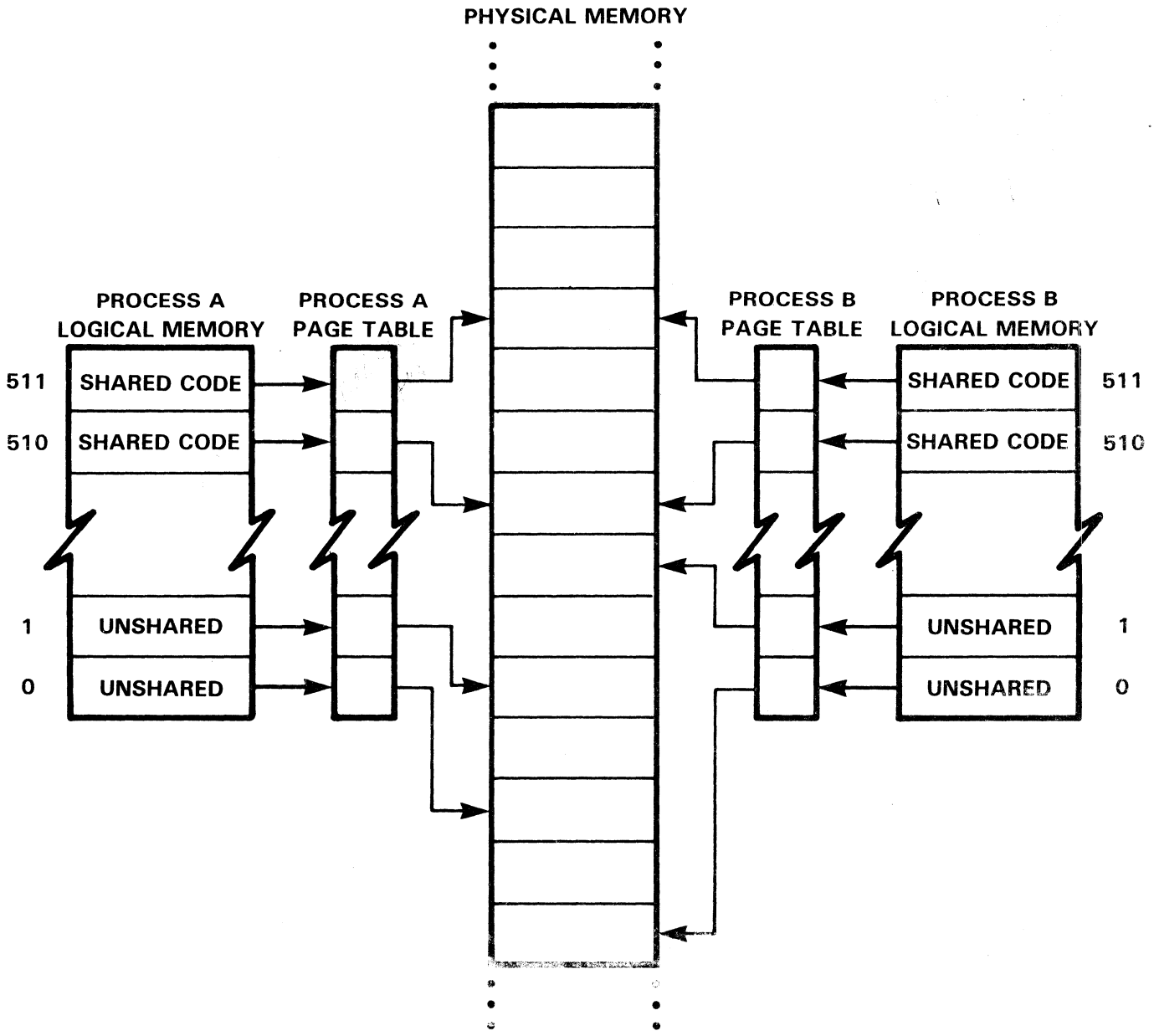
CS-04323

MEMORY WITHOUT SHARING



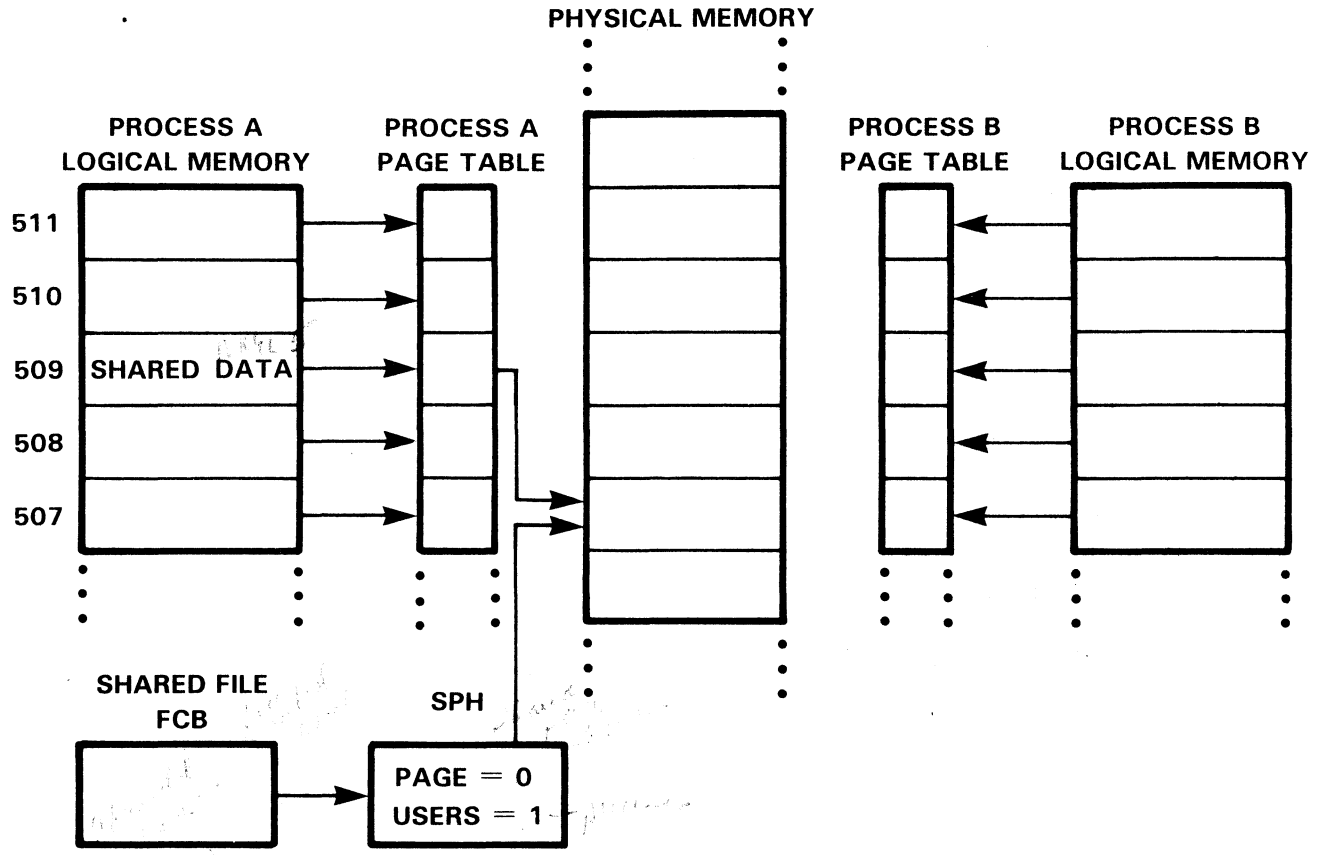
CS-04324

SHARED CODE



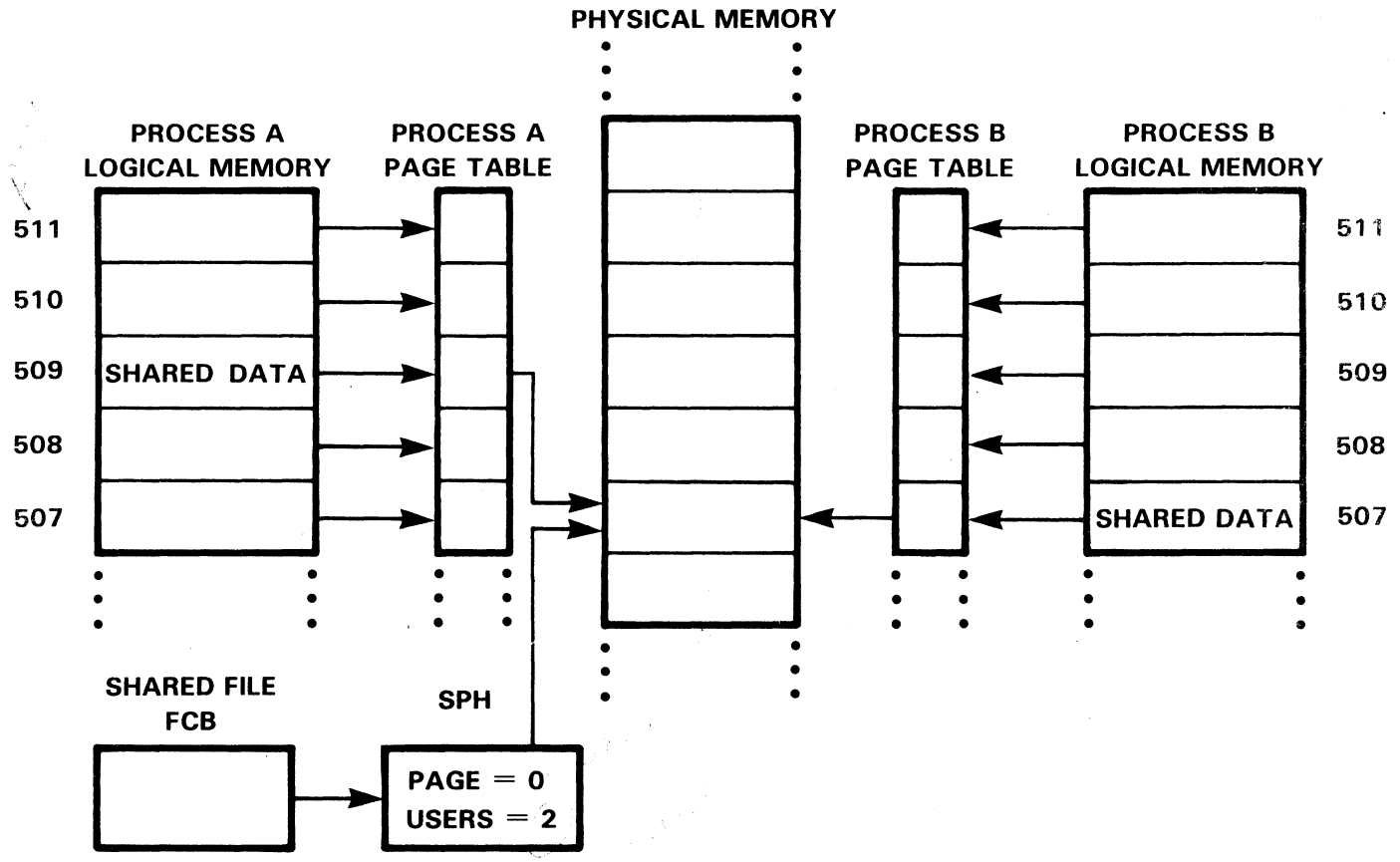
CS-04325

SHARED DATA
one user



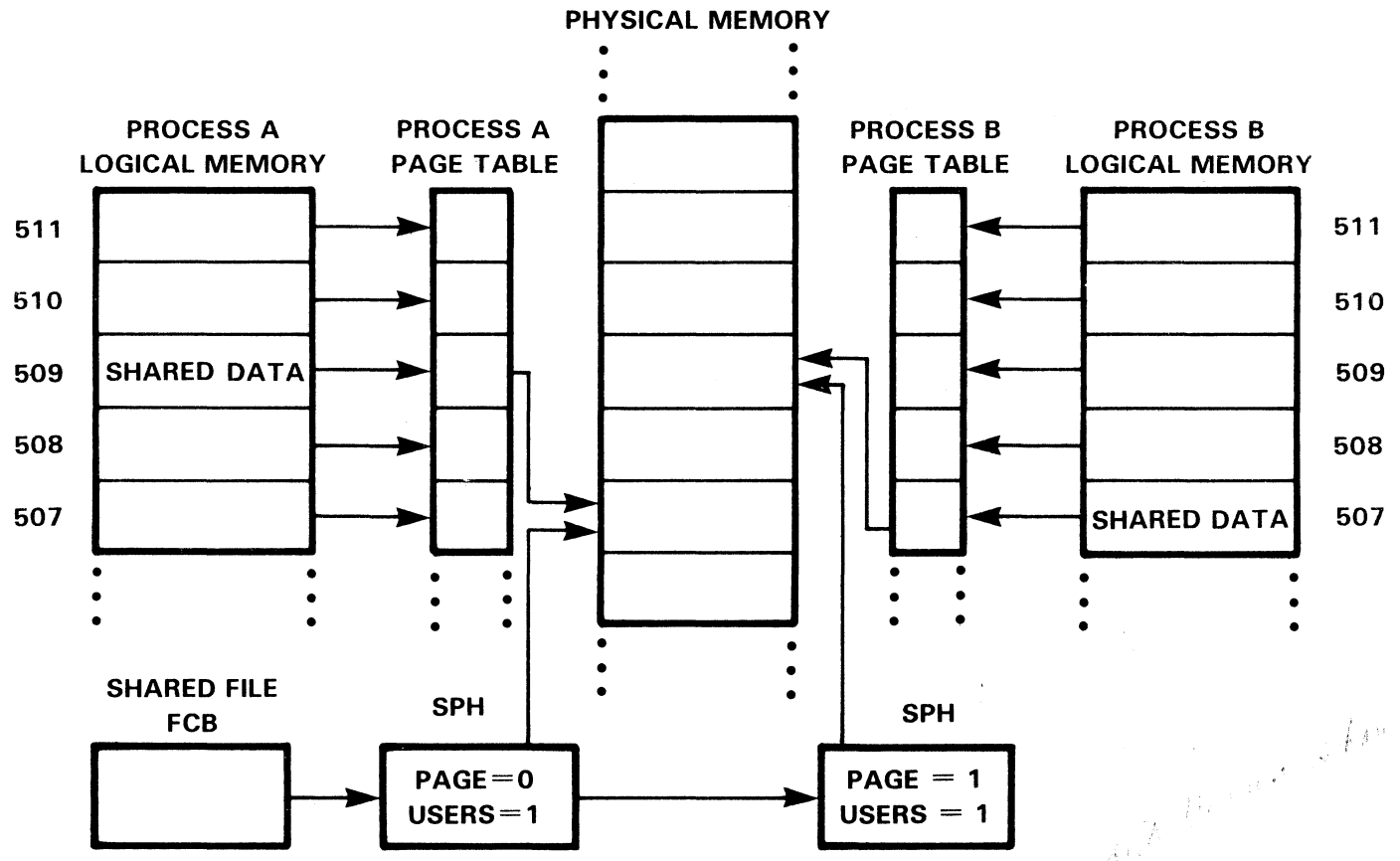
CS-04326

SHARED DATA
two users



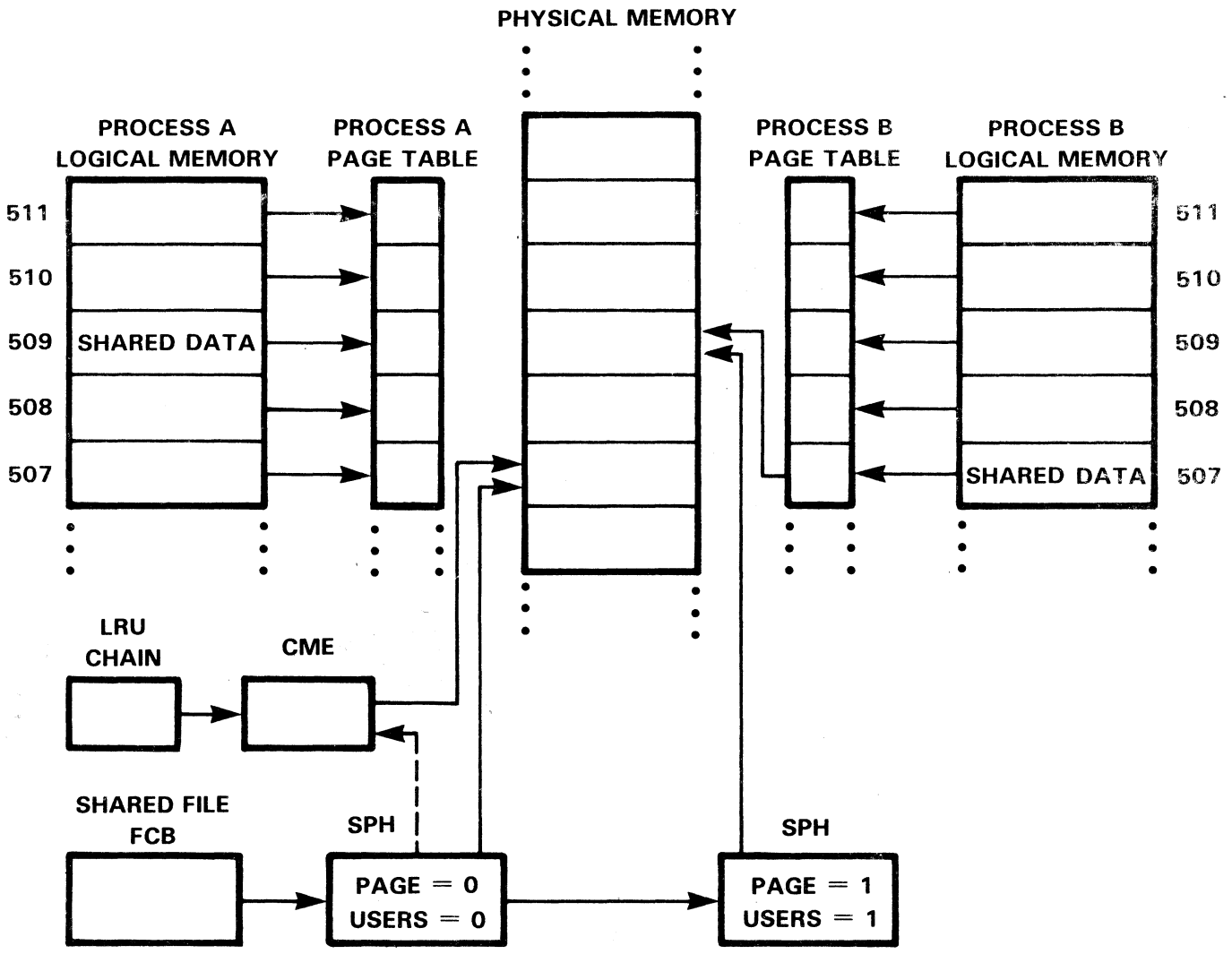
CS-04327

SHARED DATA
different pages



CS-04328

SHARED DATA on the LRU



CS-04329

SHARED FILES

- File must already exist
- File must have element size = multiple of four
- ?SOPEN

  AC0 - B.P. to filename
  AC1 - -1 = System assigns / returns channel
  AC2 - 0 = readonly

- ?SPAGE

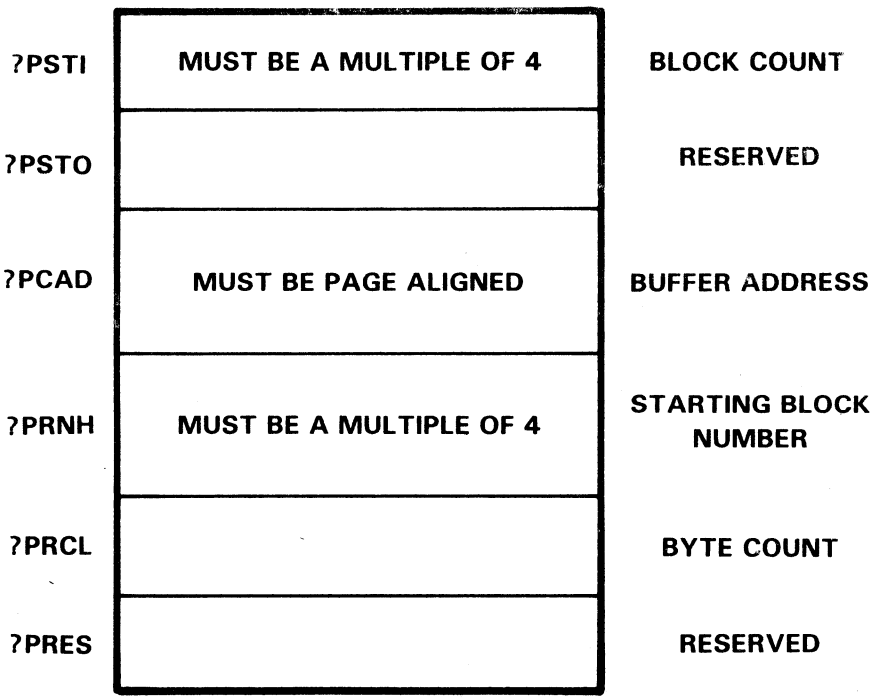
  AC0 - reserved
  AC1 - channel number
  AC2 - packet address

- Setting Up Shared Buffer

  - .NREL 5
  - .ALIGN = 10
  - .PART partname SHARED DATA ALIGN=10

  - Declare Buffer in Named Common Block
  - X LINK program common_name/SHARED/ALIGN=10

?SPAGE PACKET



length –?PBLT

CS-04331

UPDATING SHARED FILES

- Reallocation of page on LRU
  - Page has use count = 0
  - System selects this page to allocate to new use
  - If page has been modified, system writes page to shared file on disk
- User program issues ?FLUSH
  - AC0 - buffer address
  - AC1 - reserved
  - AC2 - reserved
- User program issues ?RPAGE with flush flag
  - AC0 - bit 0 - 1 = flush
    bits 1-31 buffer address
  - AC1 - reserved
  - AC2 - reserved
- Last closer closes shared file
  - ?SCLOSE
    - AC0 - reserved
    - AC1 - bit 0 = 1 - release pages
      - bits 1-31 channel number
    - AC2 - reserved
  - System writes all modified pages to shared file on disk
  - System removes any shared file page CMEs from LRU
  - System releases Shared Page Headers and File Control Block

MODULE 5

RECORD I/O

Objectives

Upon completion of this module, the student should be able to:

1. List and describe the four RECORD FORMATS.
2. List the Record I/O System Calls.
3. Identify the items of the Record I/O Packet.
4. Select appropriate ?ISTI OPTIONS in the Record I/O Packet.
5. Describe the Generic File mechanism.
6. List the five Generic File names, including their special features and restrictions.
7. Describe the relationship betweem the CLI DATAFILE and LISTFILE, and the Generic Files.
8. List the purposes of the @NULL Generic File.
9. Describe the role of the AOS/VS AGENT in Record I/O.
10. List the types of Record I/O Extensions.
11. List the items and options of the Screen Management extension.

Outline

- Record I/O
  - Record Formats
    - Datasensitive
    - Fixed
    - Variable
    - Dynamic
  - ?OPEN
  - ?READ
  - ?WRITE
  - ?CLOSE
  - Record I/O Packet
  - Generic Files
    - @INPUT
    - @OUTPUT
    - @CONSOLE
    - @DATA
    - @LIST
      - Relationship to CLI 'DATAFILE' and 'LISTFILE'
    - @NULL - a Different Kind of Generic File
  - The Role of the AOS/VS AGENT
  - Record I/O Extensions
    - Screen Management
      - Auto-terminating Reads
    - Field Translation
    - Labeled Tape Special Headers
    - Networking

Record I/O System Calls

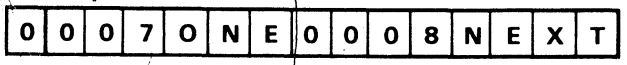
- All use same packet
- Device independent
- System AGENT hides details
- ?OPEN
  - Has delete / create options
- ?READ / ?WRITE
  - Offer choice of four record formats
- ?CLOSE
  - Or, system will close file on process termination

RECORD FORMATS

DATASENSITIVE

FIXED

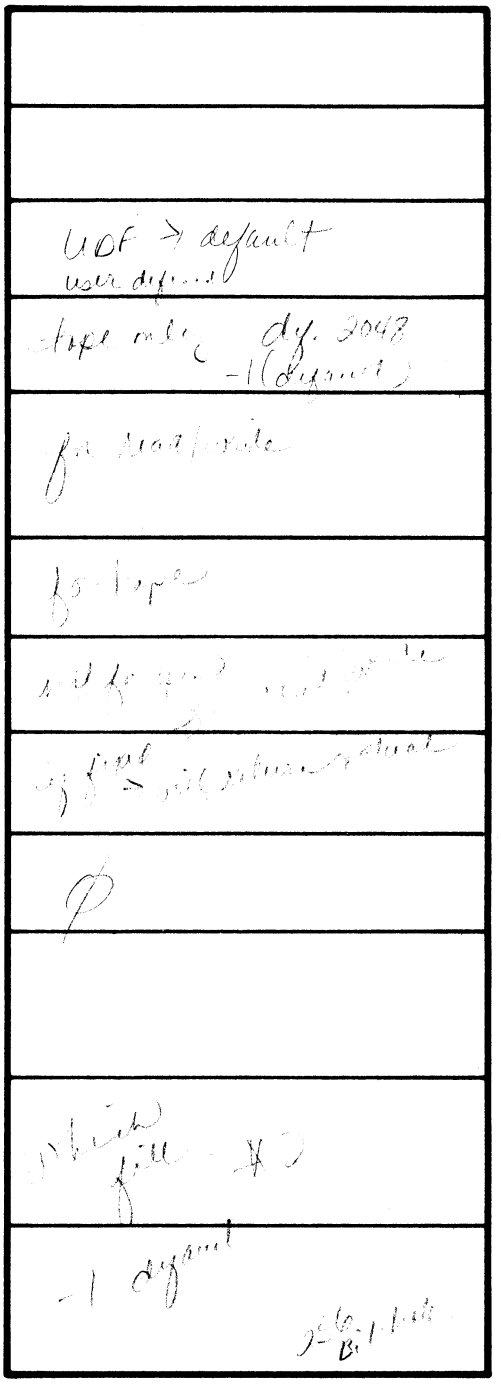
VARIABLE



DYNAMIC

CS-04332

RECORD I/O PACKET



length –?IOSZ

CS-04333

?ISTI OPTIONS

?ICRF	CHANGE RECORD FORMAT
?RTDS	DATA SENSITIVE
?RTFX	FIXED
?RTVR	VARIABLE
?RTDY	DYNAMIC
?OFIN	INPUT
?OFOT	OUTPUT
?OFIO	BOTH
?OFCE	IF FILE DOES NOT EXIST, CREATE IT, THEN OPEN
?OFCR	CREATE FILE THEN OPEN
?OFCE + ?OFCR	DELETE THE FILE IF IT EXISTS, CREATE NEW FILE AND OPEN

CS-04334

?ISTI OPTIONS

?IEXO	EXCLUSIVE OPEN
?APND	APPEND
?IPST	ABSOLUTE RECORD POSITIONING
?IFOP	FORCE OUTPUT
?IBIN	BINARY INPUT

CS-04335

MEANING OF ?IRCL
record length

OPERATION / RECORD TYPE	DATA SENSITIVE	VARIABLE	DYNAMIC	FIXED
READ	maximum	maximum	actual	actual
WRITE	maximum	actual	actual	actual

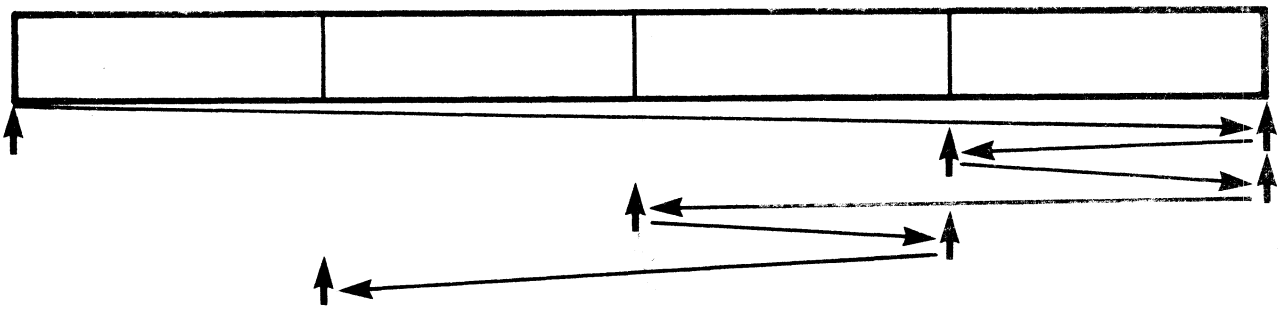
CS-04336

FILE POSITIONING

POSITIONING	RECORD NUMBER	RESULT
RELATIVE	0	NEXT RECORD
RELATIVE	N	N RECORDS (FIXED) OR N BYTES (OTHERS) FROM NEXT RECORD
ABSOLUTE	0	BEGINNING OF FILE
ABSOLUTE	-1	END OF FILE
ABSOLUTE	N	N RECORDS (FIXED) OR N BYTES (OTHERS) FROM BEGINNING OF FILE

CS-04337

RECORD POSITIONING



POSITIONING	RECORD NUMBER	RESULT
Absolute	-1	EREOF
Relative	-1	Last Record
Relative	-2	Previous Record
•	•	•
•	•	•
•	•	ERBOF

CS-04338

```
; BITBL - MACRO TO BUILD A BIT TABLE

;               THE FIRST ARGUMENT TO THE MACRO IS THE LENGTH
;       OF THE TABLE TO BUILD (IN WORDS). SUCCEEDING ARGUMENTS
;       ARE THE BIT NUMBERS OF THE BITS TO SET IN THE TABLE.

        .MACRO  BITBL

**      .NOLOC  1

        R=1                             ; INITIALIZE POINTERS
        I=0

        .DO     ^1                      ; CLEAR THE TABLE REGISTERS
                W\I=0
                I=I+1
        .ENDC

        .DO     .ARGCT-1                ; DO ONCE FOR EACH REMAINING
                                        ; ARGUMENT
                R=R+1                   ; NEXT ARG
                I=^R                    ; I GETS NUMBER OF BIT TO SET
                K=I/16.                 ; K IS WORD WITHIN THE TABLE
                L=I-(K*16.)             ; L IS BIT WITHIN THE WORD
                I=1BL                   ; GENERATE THE BIT
                W\K=W\K!I               ; OR IT INTO THE REGISTER
        .ENDC

        I=0
        .DO     ^1                      ; OUTPUT THE TABLE
        .WORD   W\I
                I=I+1
        .ENDC

**      .NOLOC  0
%
```

GENERIC FILES

	LOG ON AT CONSOLE	BATCH JOB COMMENCES	CLI XEQ COMMAND
@INPUT / ECHO / @OUTPUT	@CONn	batch job command file	Same as CLI s @INPUT
	@CONn	:QUEUE:user.OUTPUT.n	Same as CLI s @OUTPUT
@CONSOLE MUST BE TYPE "CON"	@CONn	———	Same as CLI s @CONSOLE
@DATA	———	———	CLI s DATAFILE
@LIST	———	:QUEUE:user.LIST.n	CLI s LISTFILE

CS-04339

Using Generic Files

- CLI user selects actual file for program's @DATA
  - **) DATAFILE, MYDATA**
- CLI user selects actual file for program's @LIST
  - **) LISTFILE, MYRESULTS**
  - CLI creates MYRESULTS if it does not exist
- CLI user executes program
  - **) X, PROGRAM**
- Program ?OPENs @DATA
  - System opens corresponding actual file, MYDATA
- Program ?OPENs @LIST
  - System opens corresponding actual file, MYRESULTS
  - Creation options are ignored
  - File is opened with append option
- Program reads from @DATA
  - System reads from MYDATA
- Program writes to @LIST
  - System writes to MYRESULTS

- @NULL
  - A different kind of generic file
  - Used for testing program
  - ?READ from @NULL gives End Of File error
  - ?WRITE to @NULL discards output
  - @NULL may be assigned as actual file for another generic file

Role of the Agent

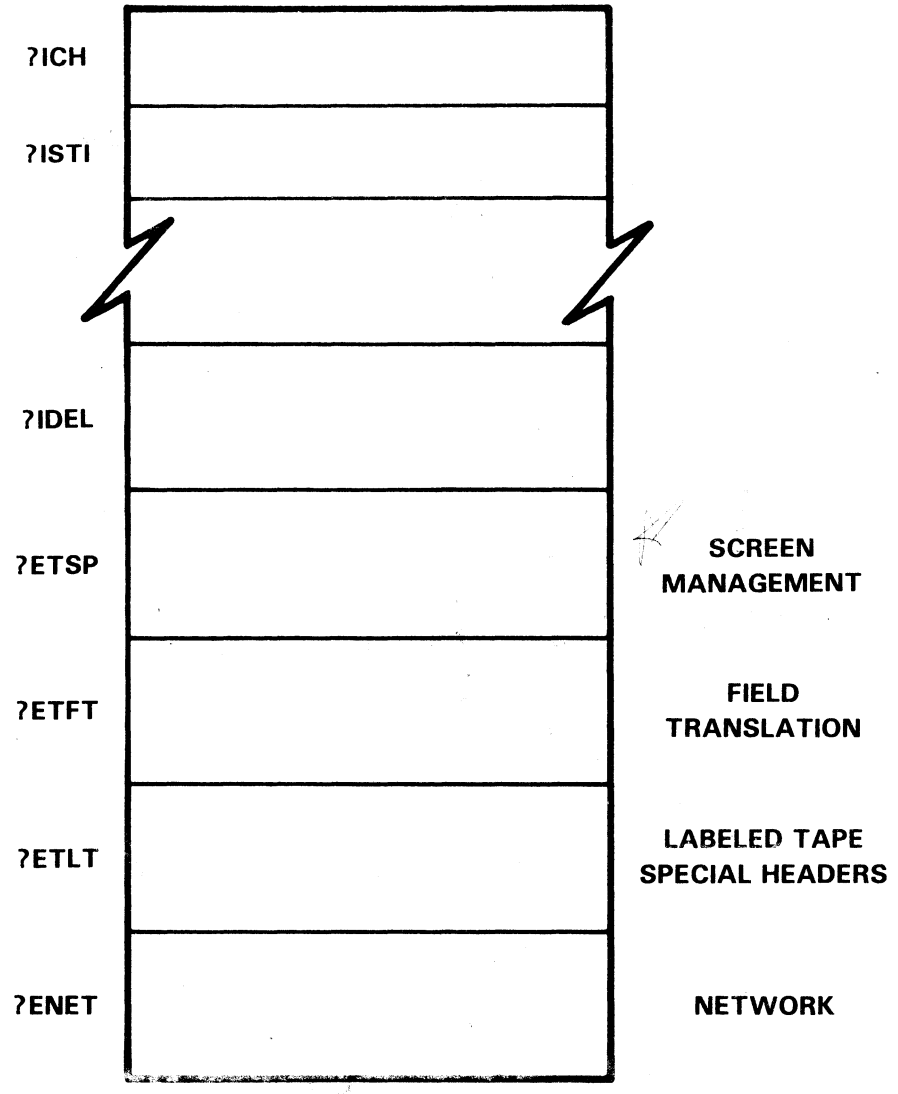
- For ?OPEN, Agent does ?GOPEN
  - Discovers file type
  - If generic, Agent opens actual file instead
  - If queue, Agent creates and opens temporary file
    - When file is closed, Agent will QPRINT/DEL the temporary file
  - If console, Agent communicates with PMGR process to open console
- For ?READ / ?WRITE, Agent does:
  - Block I/O, if file is on block oriented device
    - Uses block I/O buffer in Agent space
    - Does ?RDB when record specified by ?READ is not in current buffer
    - Copies record from Agent buffer to user record I/O buffer
    - On ?WRITE, Agent copies from user record I/O buffer to Agent buffer
    - Does ?WRB when ?WRITE fills current buffer
  - PMGR I/O routines in Agent, if file is a console device
  - Primitive IPCs, if file is type IPC
- On process termination, Agent automatically closes files

Record I/O Extensions

- Screen Management
- Field Translation
- Labelled Tape Special Headers
- Networking

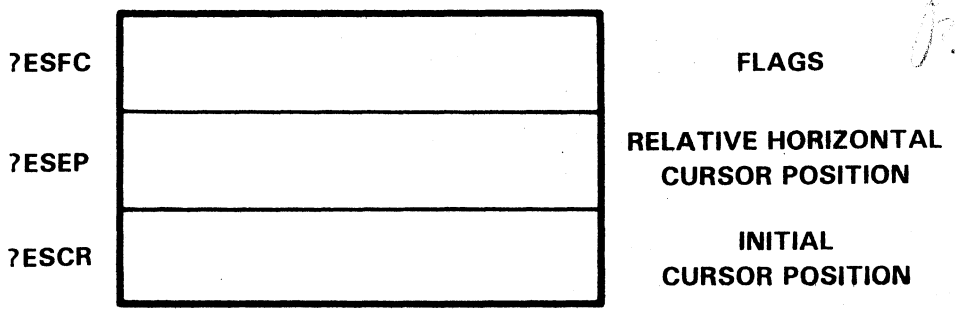
- Include ?IPKL flag in offset ?ISTI
- Record I/O packet contains four additonal double words
  - Bits 1-31 contain extension packet address
    - Non-zero means extended feature is desired
  - Bit 0 is flag bit
    - Non-zero means examine extension packet
    - System zeros flag bit after examining extension
    - Zero means do not examine extension, use previous values
- Packet length is ?IBLT

EXTENDED RECORD I/O PACKET



CS-04340

SCREEN MANAGEMENT



Flags:

?ESSE Enable Screen Edit

?ESRD Display Before Read

?ESCP Initial Cursor Position Specified

?ESNE Read with no Echo

?ESED Do Not Echo Delimiters

?ESRP Return Cursor Position

?ESNR Drop Type Ahead

?ESGT End Read if Buffer Empty

?ESBE (returned) Read Ended due to Buffer Empty

?ESDD (returned) Delimiter was Function Key

CS-04341

Auto Terminating Reads

- Program stores two consecutive delete characters (<177>) at end of ?READ buffer
- Buffer length (?IRCL) should include delete characters
- Length of auto-entry field will be number of byte positions preceeding first delete in buffer
- Use enable screen edit flag ?ESSE
- ?READ terminates when:
  - User enters delimiter
  - User fills auto-entry field without entering delimiter

MODULE 6

PROCESSES

Objectives

Upon completion of this module, the student should be able to:

1. Identify the attributes of a process.
2. List the Process Types.
3. List the range of assignable priorities for each type.
4. Show the relationship between assigned priorities and priority enqueue factor (PNQF).
5. State the process scheduling algorithm.
6. Create processes, assigning the appropriate attributes.
7. Manage processes using the process informational and control system calls.
8. Send and receive initial IPC messages.
9. Terminate processes.
10. Send and receive process termination messages.

Outline

- Processes
  - Attributes
  - Scheduling
  - Creation
  - Management
  - Initial Messages
  - Termination

PROCESS ATTRIBUTES

PID

USERNAME

PROCESS NAME

* PROGRAM

INITIAL DIRECTORY

* CURRENT DIRECTORY

* SEARCHLIST

* DEFAULT ACL

GENERIC FILES

WORKING SET LIMITS

* PROCESS TYPE

* PROCESS PRIORITY

PRIVILEGES

PROCESS PRIVILEGES

USE IPC

ACCESS DEVICES

SUPERUSER

SUPERPROCESS

CREATE WITHOUT BLOCKING

UNLIMITED SONS OR SONS QUOTA

CHANGE LOGICAL ADDRESS SPACE TYPE

CHANGE WORKING SET LIMITS

CHANGE USERNAME

CHANGE TYPE

CHANGE PRIORITY

CS-04343

PROCESS TYPES

RESIDENT	Never swapped.
PREEMPTIBLE	Normally swapped only when blocked.
SWAPPABLE	Can be swapped even when not blocked.

CS-04359

PROCESS GROUPS

Group 1	Group 2	Group 3
1-255	256,257,258	259-511

CS-04360

SCHEDULER ALGORITHM

The scheduler always selects the highest priority, ready, eligible process to run next.

CS-04361

GROUPS AND PNQF

1-255

Group 1: PNQF = assigned priority

Group 2: Scheduled heuristically

Group 3: PNQF = assigned priority plus constant

CS-04362

HEURISTIC SCHEDULING

- Group 2 processes only.
- Considers process behavior.
- Interactive – receive high priority (low PNQF).
- Compute-bound – receive low priority (high PNQF)

CS-04363

TIME SLICES

- Prediction of amount of CPU time process will be able to use without blocking itself.
- Time Slice = $S_S * 2^{TSE}$
  - Where S_S = Subslice = 32 msec.
  - And TSE = Time Slice Exponent, also known as Behavior Rating.

1-6

CS-04364

UPDATING BEHAVIOR RATINGS

- Process uses entire time slice without blocking itself.
  - New B.R. = old B.R. + 1
- Process blocks itself before using up its time slice.
  - New B.R. based on actual CPU time used.

CS-04365

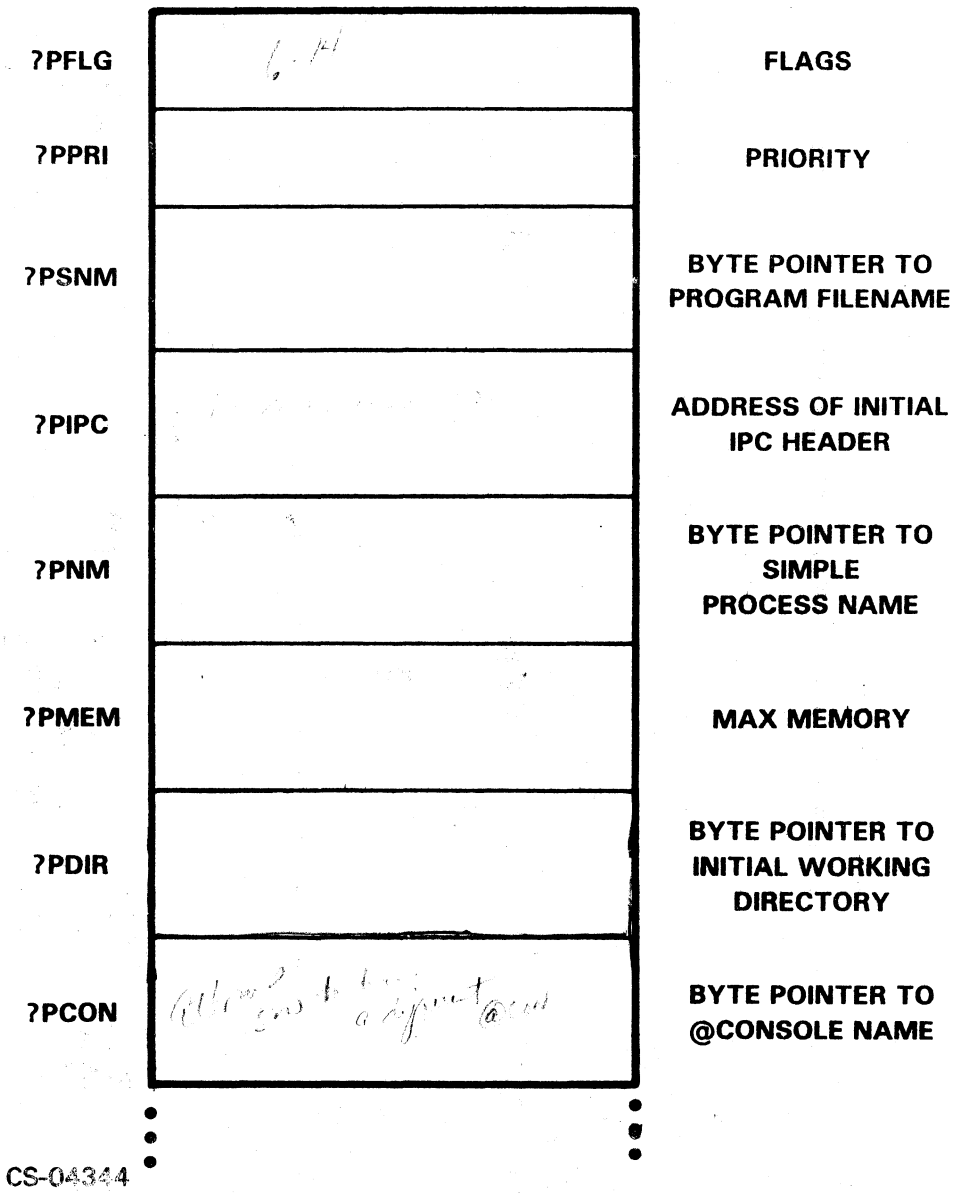
SPECIAL CASES

- Swappable 1, 2, 3 = Group 2.
- Group boundaries set at VSGEN.

Group 1	Group 2	Group 3
1 to G_1	(G_1 + 1) to G_2 Swappable 1, 2, 3	(G_2 + 1) to 511

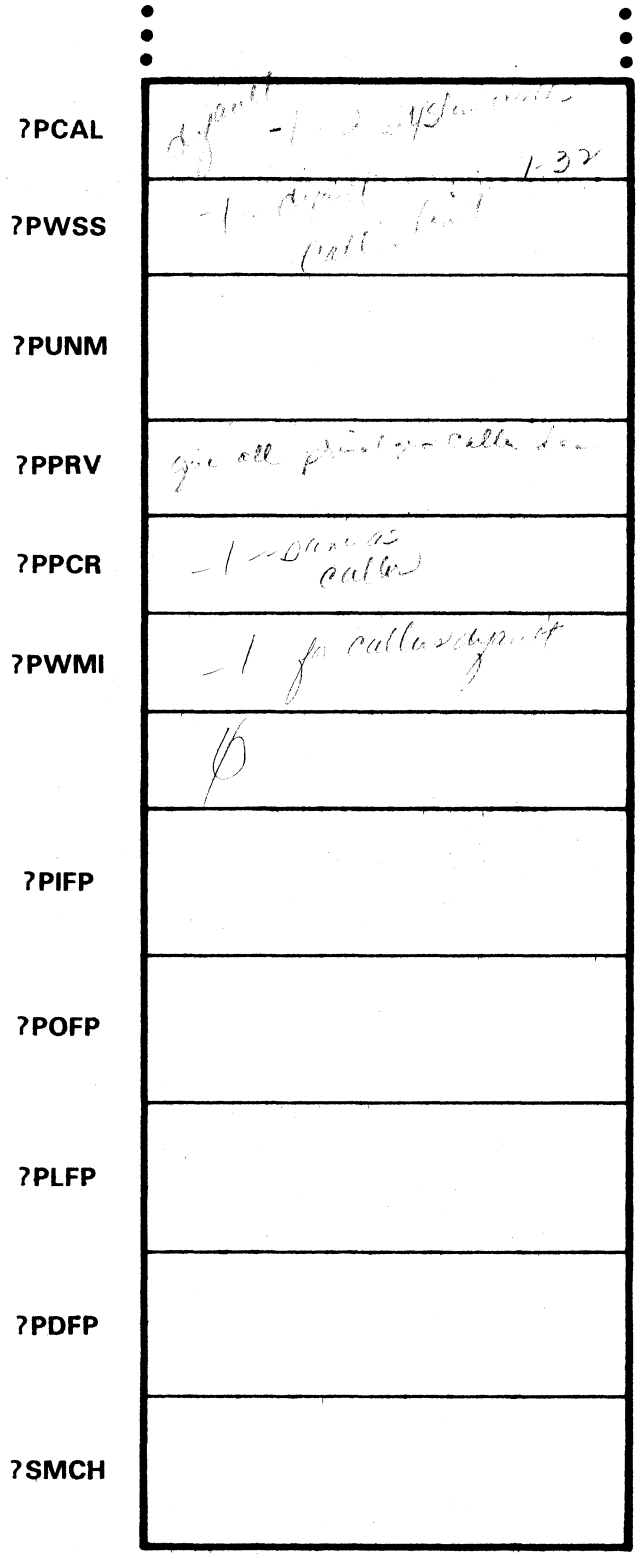
CS-04366

?PROC PACKET


?PFLG
FLAGS
?PPRI
PRIORITY
?PSNM
BYTE POINTER TO PROGRAM FILENAME
?PIPC
ADDRESS OF INITIAL IPC HEADER
?PNM
BYTE POINTER TO SIMPLE PROCESS NAME
?PMEM
MAX MEMORY
?PDIR
BYTE POINTER TO INITIAL WORKING DIRECTORY
?PCON
BYTE POINTER TO @CONSOLE NAME
CS-04344

?PROC PACKET

continued



?PCAL	MAX SIMULTANEOUS SYSTEM CALLS
?PWSS	MAX WORKING SET SIZE
?PUNM	BYTE POINTER TO USERNAME
?PPRV	PRIVILEGES
?PPCR	SONS QUOTA
?PWMI	MIN WORKING SET SIZE
	RESERVED
?PIFP	BYTE POINTER TO @INPUT FILENAME
?POFP	BYTE POINTER TO @OUTPUT FILENAME
?PLFP	BYTE POINTER TO @LIST FILENAME
?PDFP	BYTE POINTER TO @DATA FILENAME
?SMCH	CPU TIME LIMIT

length –?PLTH

CS-04345

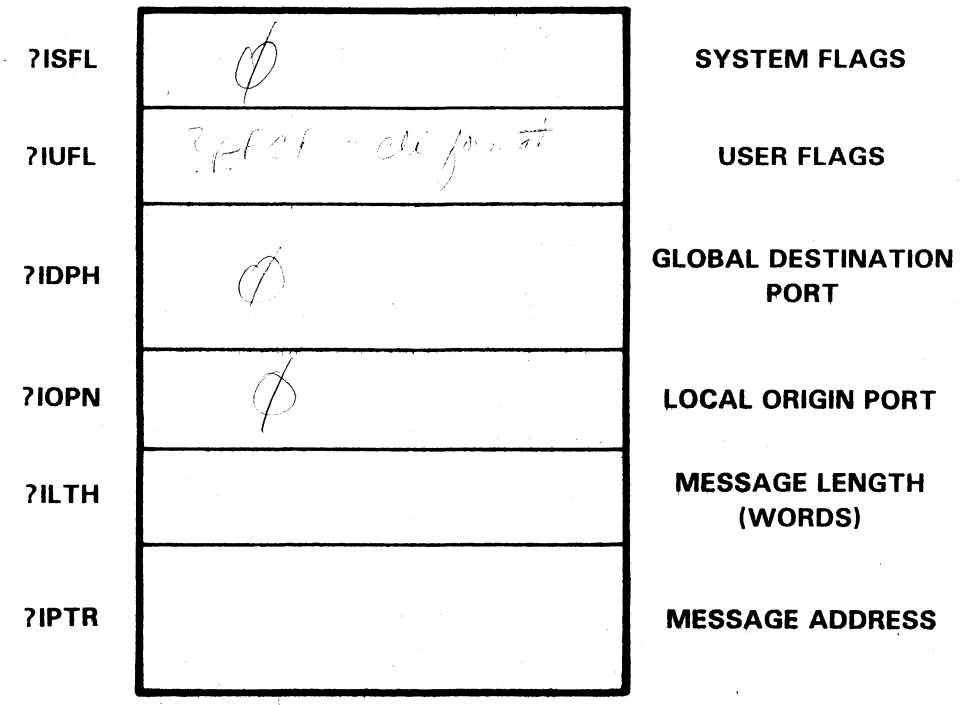
Process Creation Flags

- ?PFEX Caller is blocked until subordinate terminates
- ?PFRP Process is preemptible
- ?PFRS Process is resident
- ?PFDB Process starts in debugger
- ?PFBS Process starts off blocked
- ?PFPM Complement privileges word
- ?PFDA Do not pass DEFACL
- ?PBRK Create breakfile if process aborts
- ?PDMP Create memory dump if process aborts

Process Privilege Flags

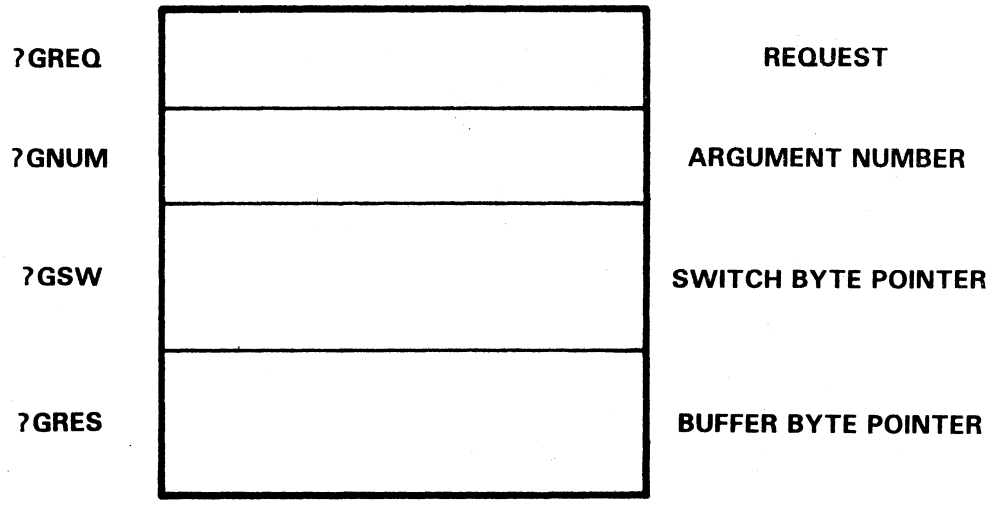
- ?PVIP Use IPC
- ?PVDV Access Devices
- ?PVSU Superuser
- ?PVSP Superprocess
- ?PVEX Create Without Blocking
- ?PVPC Unlimited Sons
- ?PVWS Change Logical Type (16 Bit / 32 Bit)
- ?PVWM Change Working Set Limits
- ?PVUI Change Username
- ?PVTY Change Type
- ?PVPR Change Priority

INITIAL IPC MESSAGE HEADER



CS-04346

?GTMES PACKET



CS-04347

?GTMES REQUESTS

?GMES	GET ENTIRE MESSAGE
?GCMD	GET COMMAND LINE
?GCNT	GET ARGUMENT COUNT
?GARG	GET ARGUMENT
?GTSW	GET SWITCH
?GSWS	GET SWITCHES

CS-04348

Process Management System Calls

- ?BLKPR Block a process
- ?UBLPR Unblock a process

- ?CTYPE Change process type
- ?PRIPR Change process priority
- ?SUSER Enable / disable superuser mode
- ?SUPROC Enable / disable superprocess mode

- ?DADID Get PID of a process' creator
- ?GUNM Get a process' username
- ?PNAME Get a process' process name
- ?PSTAT Get process statistics
- ?RUNTM Get runtime information

- ?CHAIN Chain to a different program
- ?TERM Terminate a process

```
            AC0     Optional ERROR CODE
?RETURN     AC1     Optional MESSAGE BYTEPOINTER
            AC2     FLAGS + MESSAGE BYTE LENGTH
```

FLAGS

```
            ?RFEC   AC0 CONTAINS AN ERROR CODE
            ?RFWA   "WARNING"
            ?RFER   "ERROR"
            ?RFAB   "ABORT"
            ?RFCF   MESSAGE IS IN CLI FORMAT
```

NORMAL TERMINATION

```
                    WSUB    2,2
                    ?RETURN
                    WBR     ERR
```

ERROR TERMINATION

```
                            ; ERROR CODE IS IN AC0
                    WLDAI   ?RFEC+?RFER+?RFCF,2
                    ?RETURN
                    WBR     ERR
```

OPTIONAL MESSAGE

```
                            ; ERROR CODE IS IN AC0
                    LLEFB   1,MSG*2
                    WLDAI   ?RFEC+?RFER+?RFCF+MSLN,2
                    ?RETURN
                    WBR     ERR


            MSG:    .TXT    "IN MODULE XYZ"
            MSLN = (.-MSG)*2
```

MODULE 7

INTER-PROCESS COMMUNICATION

Objectives

Upon completion of this module, the student should be able to:

1. Differentiate between local and global IPC port numbers.
2. Describe explicit and implicit port matching.
3. Create an IPC file to represent a port.
4. Use an IPC file to determine another process' port.
5. Send and receive IPC messages using ?ISEND, ?IREC, and ?IS.R.
6. Describe IPC spooling.
7. List IPC performance considerations.
8. Describe the role of the AGENT in File IPC.

Outline

- Inter-Process Communication
  - Primitive IPC
    - IPC Port Numbers
      - Local
      - Global
    - Port Matching
      - Explicit
      - Implicit
    - IPC Headers
      - ?IREC
      - ?ISEND
      - ?IS.R
    - IPC Message Transfer
      - IPC Spooling
      - Header to Header Transfer
      - Buffer to Buffer Transfer
  - IPC Performance Considerations
  - File IPC
    - Role of the AOS/VS Agent

IPC PORT NUMBERS

- Local
  - Used to refer to a process's own port
  - 1 to ? IMPRT
- Global
  - Used to refer to another process's port
  - Double word
    - Host ID
    - PID
    - Ring
    - Local port

CS-04367

PORT MATCHING

- Sender specifies local origin and global destination.
- Receiver specifies global origin and local destination.
- Both pairs must match.
- Explicit matching:
  - Global forms of port numbers identical.
- Implicit matching:
  - Receiver specifies zero port.
    - Zero global origin = Receive from any sender
    - Zero local destination = Receive on any local port

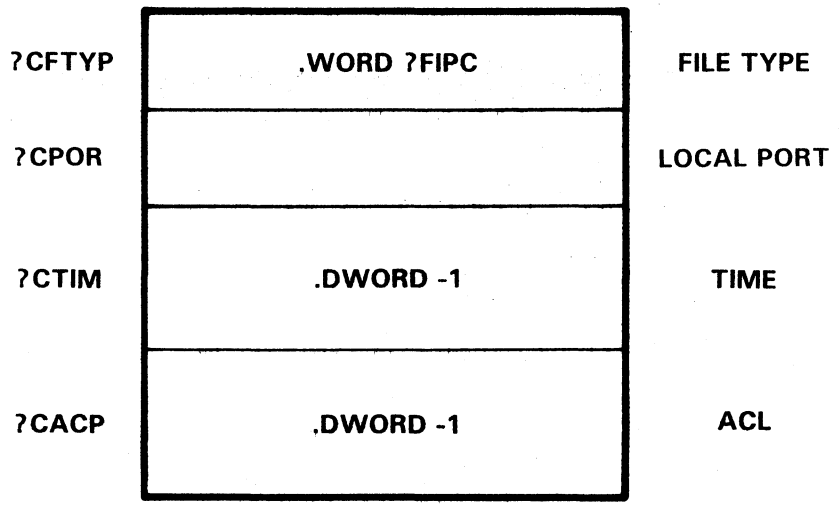
CS-04368

IPC FILE

- Stands for IPC port.
- Always empty.
- Creator specifies local port that file represents.
- Another process can determine creator's global port using ?ILKUP or ?GOPEN.
- File must be in creating process's initial directory.
- System deletes file automatically on process termination.
- Although file has ACL, it is not enforced.
- Only Superuser can delete file.

CS-04369

?CREATE PACKET FOR IPC FILE



CS-04349

?ILKUP

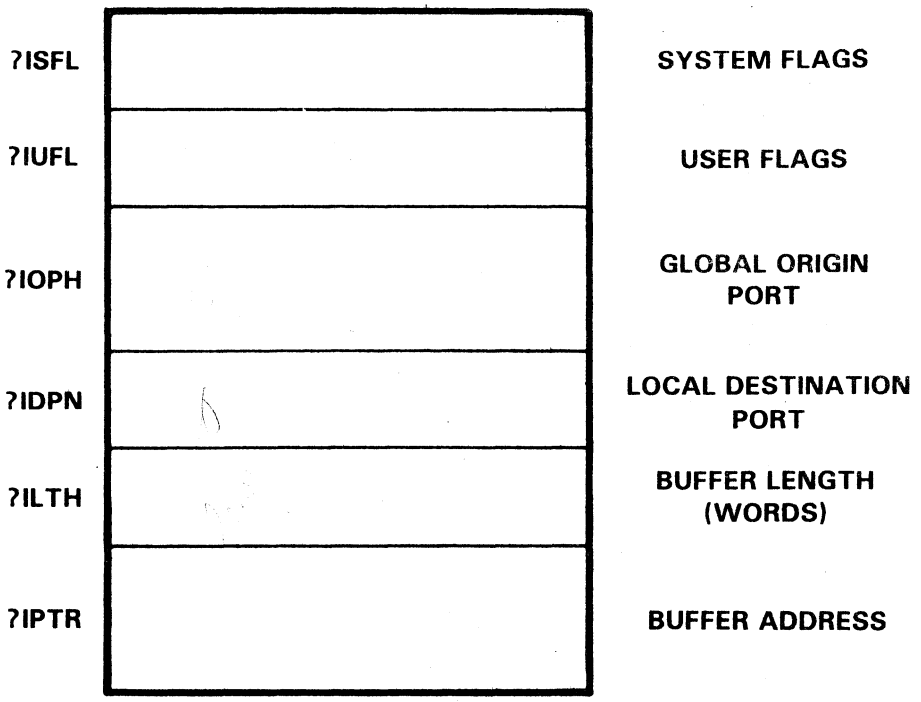
- Input
  - AC0 - B.P. to filename
  - AC1 - reserved
  - AC2 - reserved
- Output
  - AC0 - B.P. to filename
  - AC1 - Global Port Number
  - AC2 - File Type

IPC SPOOLING

- System creates an IPC spoolfile for each process. :PROC:IPS.Pid
- Headers spooled in memory.
- Messages spooled in IPC spoolfile.
- Up to 48. spooled messages per user ring (4-7).

CS-04370

?IREC HEADER

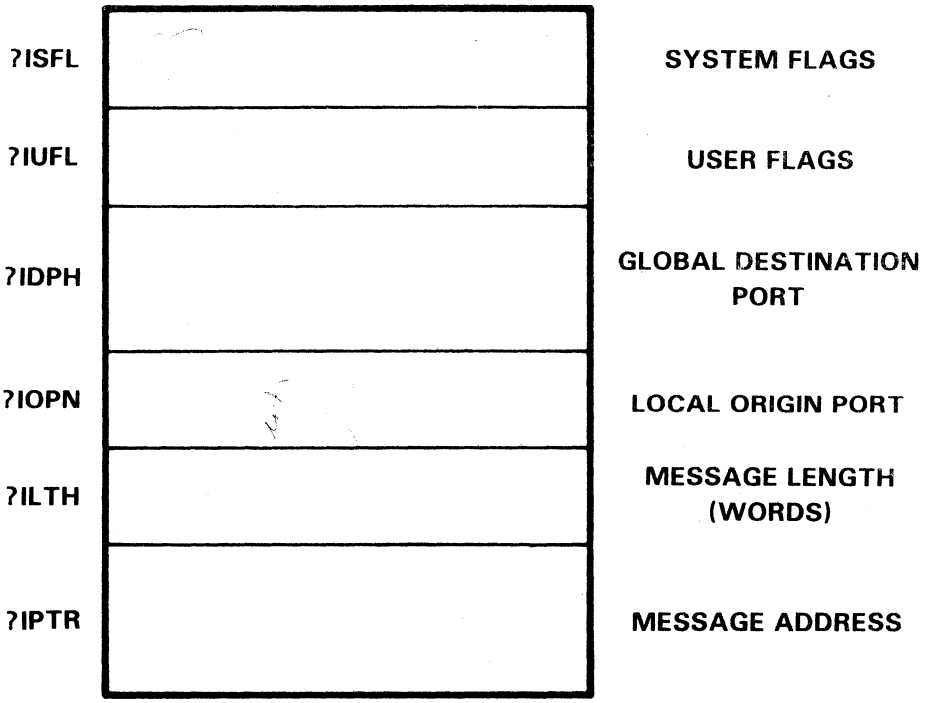


CS-04350

?IREC Flags

- If no matching message has already been sent
  - Default - Receiver is suspended until matching message is sent
  - ?IFNBK - Receiver gets exception
- If message is too long for receiver's buffer
  - Always - Receiver gets exception
  - Default - Message overflow is lost
  - ?IFSOV - Entire message is spooled

?ISEND HEADER

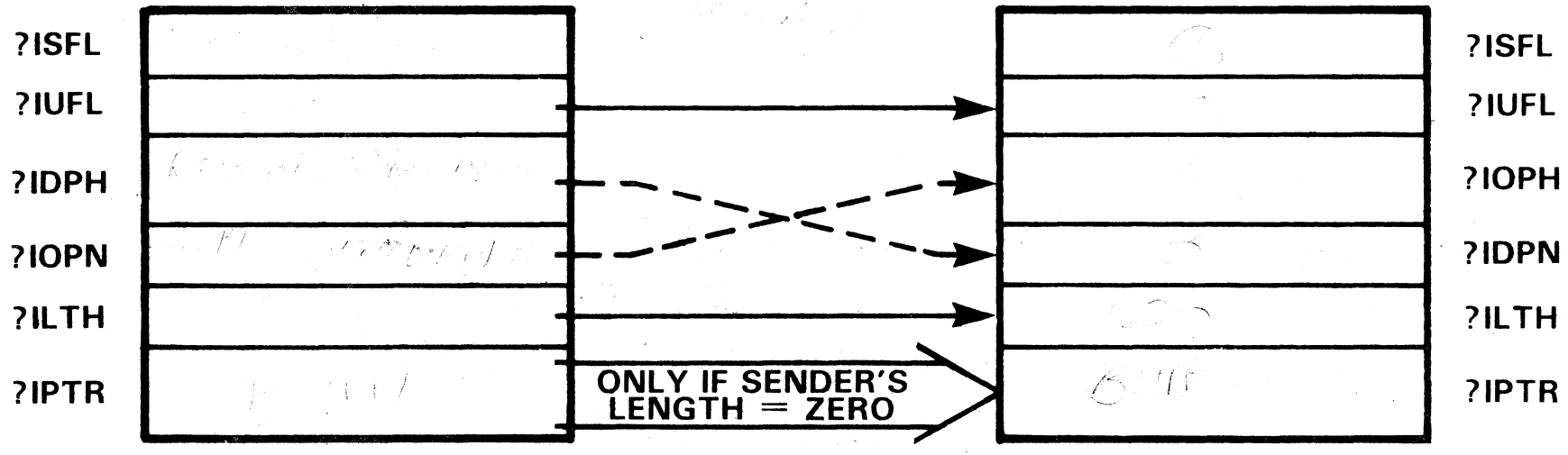


length - ?IPLTH

CS-04351

?ISEND Flag

- If no matching receive is already waiting
  - Default - Message is spooled
  - ?IFNSP - Sender gets exception


?ISEND HEADER
?IREC HEADER
?ISFL
?IUFL
?IDPH
?IOPN
?ILTH
?IPTR
?ISFL
?IUFL
?IOPH
?IDPN
?ILTH
?IPTR
ONLY IF SENDER'S LENGTH = ZERO

CS-03848

IPC SEND/RECEIVE PROCEDURE

Process A "Initial Receiver"

- Create IPC file
  - Filename
  - File type ?FIPC
  - Local port
- Set up ?IREC header
  - Zero global origin port = Receive from any sender
  - Local destination port:
    - Same as IPC file
    - Or Zero = Receive on any local port

Process B "Initial Sender"

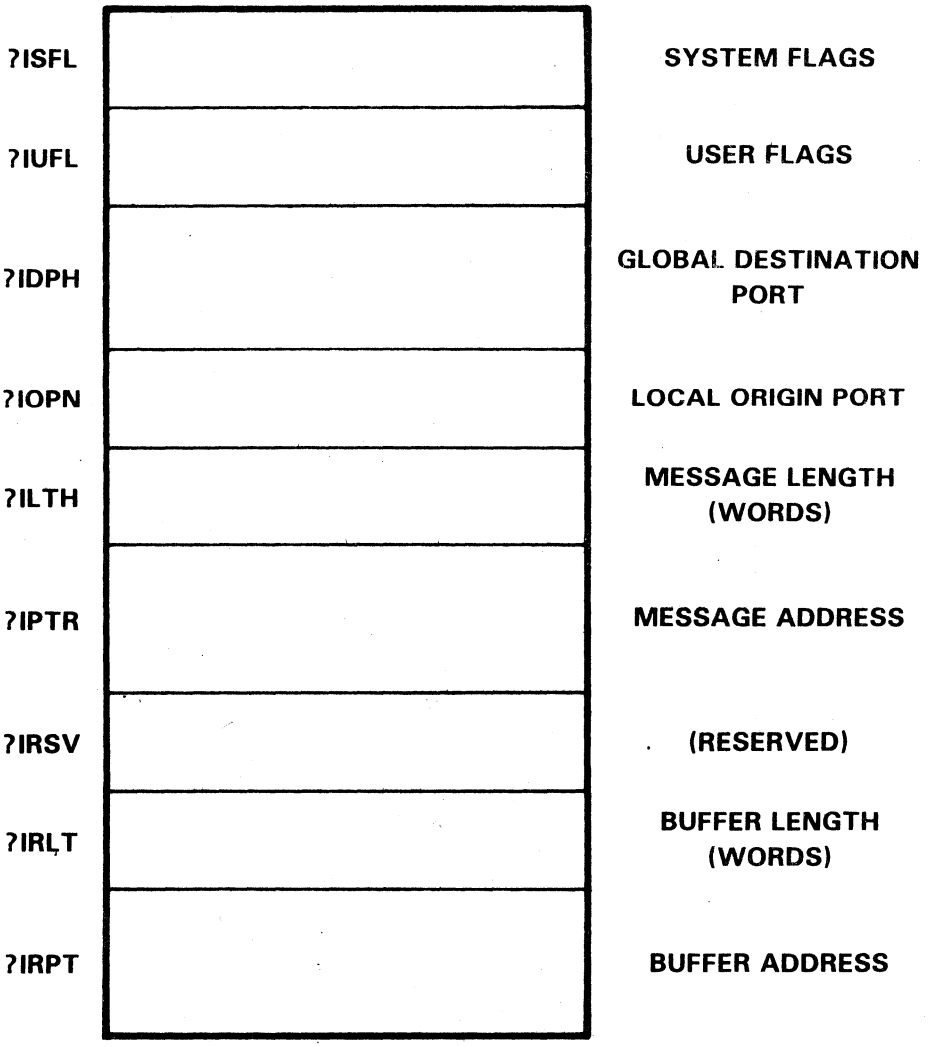
- ?ILKUP or ?GOPEN IPC file
  - Filename
  - Returns creator's global port
- Set up ?ISEND header
  - Any local origin port
  - Use result from ?ILKUP for global destination port

- ?IREC ← Sender's actual ports copied to receiver's header ← • ?ISEND

Both processes can now ?ISEND/?IREC in any order

CS-04371

?IS.R HEADER



length - ?IPRLTH

CS-04352

IPC Performance Considerations

- Always have ?IREC outstanding
  - Use implicit matching
  - Prevents Spooling
- Use zero length messages
- Use shared file to implement shared data

FILE IPC

Process A "Initial Receiver"

- ?OPEN IPC file
  - –?ISTI –?OFCR
  - –?ISTO –?FIPC
- AGENT does:
  - –?CREATE IPC file
    - Filename
    - File type ?FIPC
    - Local port
  - – Set up ?IREC header
    - Zero global origin port = Receive from any sender
    - Local destination port: Same as IPC file

Process B "Initial Sender"

- ?OPEN IPC file
  - – Expects file to already exist
- AGENT does:
  - –?GOPEN IPC file
    - Filename
    - Discovers file is type ?FIPC
    - Returns creator's global port
  - – Set up ?ISEND header
    - Any local origin port
    - Use result from ?GOPEN for global destination port

–?IREC ← Sender's actual ports copied to receiver's header ← –?ISEND

Both processes can now ?WRITE/?READ in any order

CS-04353

?SEND

- Send a message to a process' console
- AC0 - Target Identifier
  - PID
  - B.P. to Process Name
  - B.P. to Console Name
- AC1 - B.P. to message
- AC2
  - Bits 22-23 - Type of Target Identifier in AC0
    - 0 - PID
    - 1 - B.P. to Process Name
    - 2 - B.P. to Console Name
  - Bits 24-31 - Message Byte Length

```
        LLEFB   0,CONNAME*2
        LLEFB   1,MSG*2
        WLDAI   2S23+MSGLN,2
        ?SEND
        WBR     ERROR


MSG:    .TXT    "HELLO"
MSGLN   =       (.-MSG)*2
CONNAME:.TXT    "@CON0"
```

MODULE 8

MULTITASKING

Objectives

Upon completion of this module, the student should be able to:

1. Draw a sketch showing the relationships of the User Status Table, and the Task Control Blocks on the Active and Free Chains.
2. List the Task Attributes.
3. Describe Task Scheduling.
4. Identify features useful for debugging a multitask.
5. Create one or more tasks using ?TASK.
6. Identify the task management system calls and their functions.
7. Identify the task termination system calls.
8. Describe multitask support in high level languages.

Outline

- Multitasking
  - System Management of Multitasking
    - User Status Table (UST)
    - Task Control Blocks (TCBs)
      - Free Chain
      - Active Chain
    - Task Attributes
      - Task Scheduling
    - Debugging a multitasking program
  - Task Creation
    - ?TASK
      - Creating One Task
        - Task Stacks and Stack Faults
      - Creating Multiple Tasks
  - Task Management
    - ?SUS
    - ?IDSUS
    - ?PRSUS
    - ?IDRDY
    - ?PRRDY
    - ?DELAY
    - ?INTWT
    - ?KWAIT
    - ?KIOFF
    - ?KION
  - Task Termination
    - ?KILL
    - ?IDKIL
    - ?PRKIL

Multitasking

Definition: A task is an asynchronous flow of control through a program's code.

The logical concept "flow of control" is embodied at the physical level by the program counter - each task appears to have its own PC.

Asynchronous means that all the tasks in a process execute independently and appear to execute simultaneously.

Every process has at least one task and any process may have up to 32 tasks.

All the tasks in a process have access to that process' entire logical address space.

Unique Per Task Items - **The system makes it appear as though each task has its own unique set of the following items:**

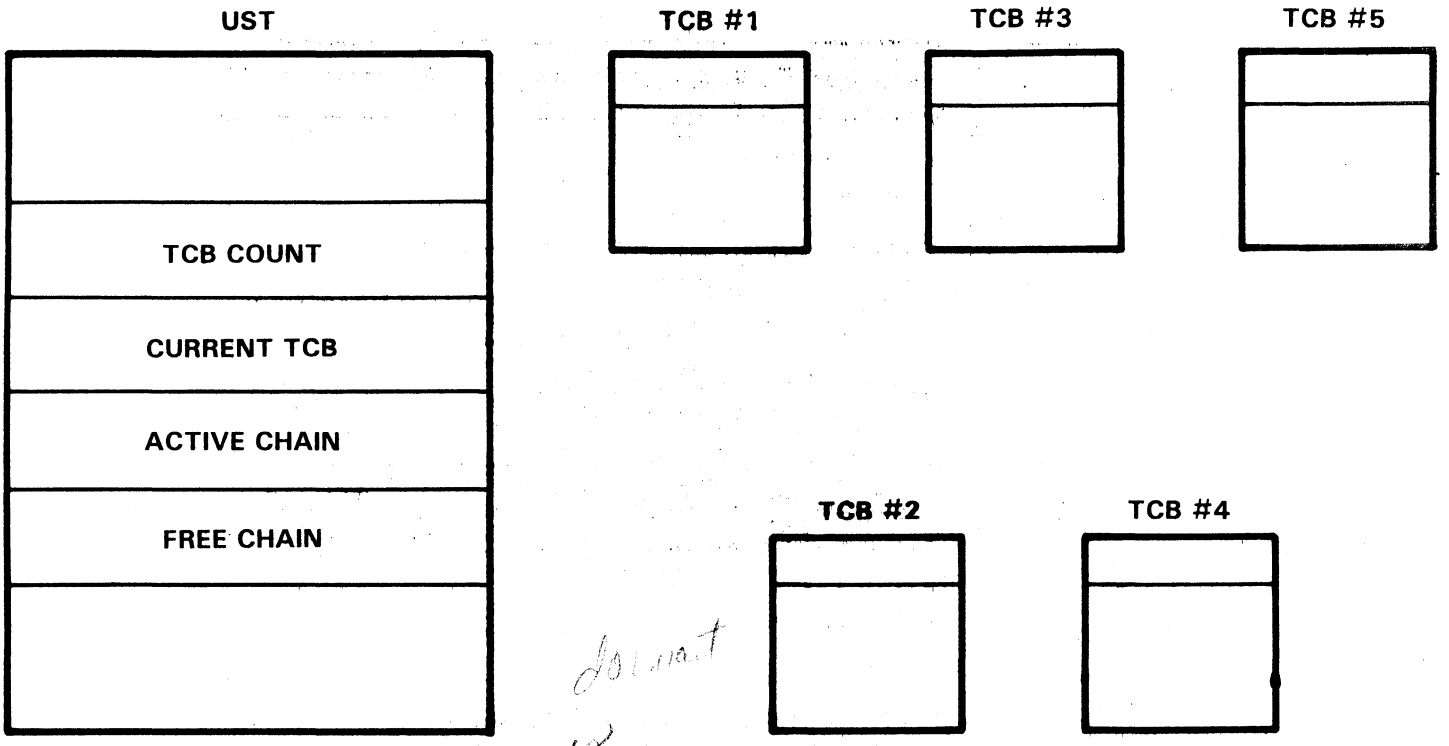
- Program Counter
- Carry Bit
- Accumulators
- Floating Point ACs (optional)
- Stack

TASK ATTRIBUTES

- **TASK ID**
  - Range 0 – 255
  - Up to 32 tasks in a process
  - 0 = anonymous
  - Default task starts with TID 1
- **TASK PRIORITY**
  - Range 0 – 255
  - Default task has priority 0
  - No hierarchy among tasks
- **TASK STATE**
  - Executing
  - Ready
  - Suspended
  - Dormant

CS-04354

TCB CHAINS



CS-04355

DEBUGGING A MULTITASK UST DISPLAY

```
_$K
UST Status:      020000
        Task Scheduling is inhibited by the User


Active Task:     1. (TID=1.) Pended on ?POKE
4. Active TCBs:  1. (TID=1.) =>  4.(4.) =>  3.(3.) =>  2.(2.)
1. Free TCBs:    5. (TID=0.) =>


Memory           SUS             EUS             SSH             ESH
        Ring 3:  06000000000     06000051777     06001064000     06002005777
        Ring 7:  16000000000     16000001777     16001776000     16001777777
```

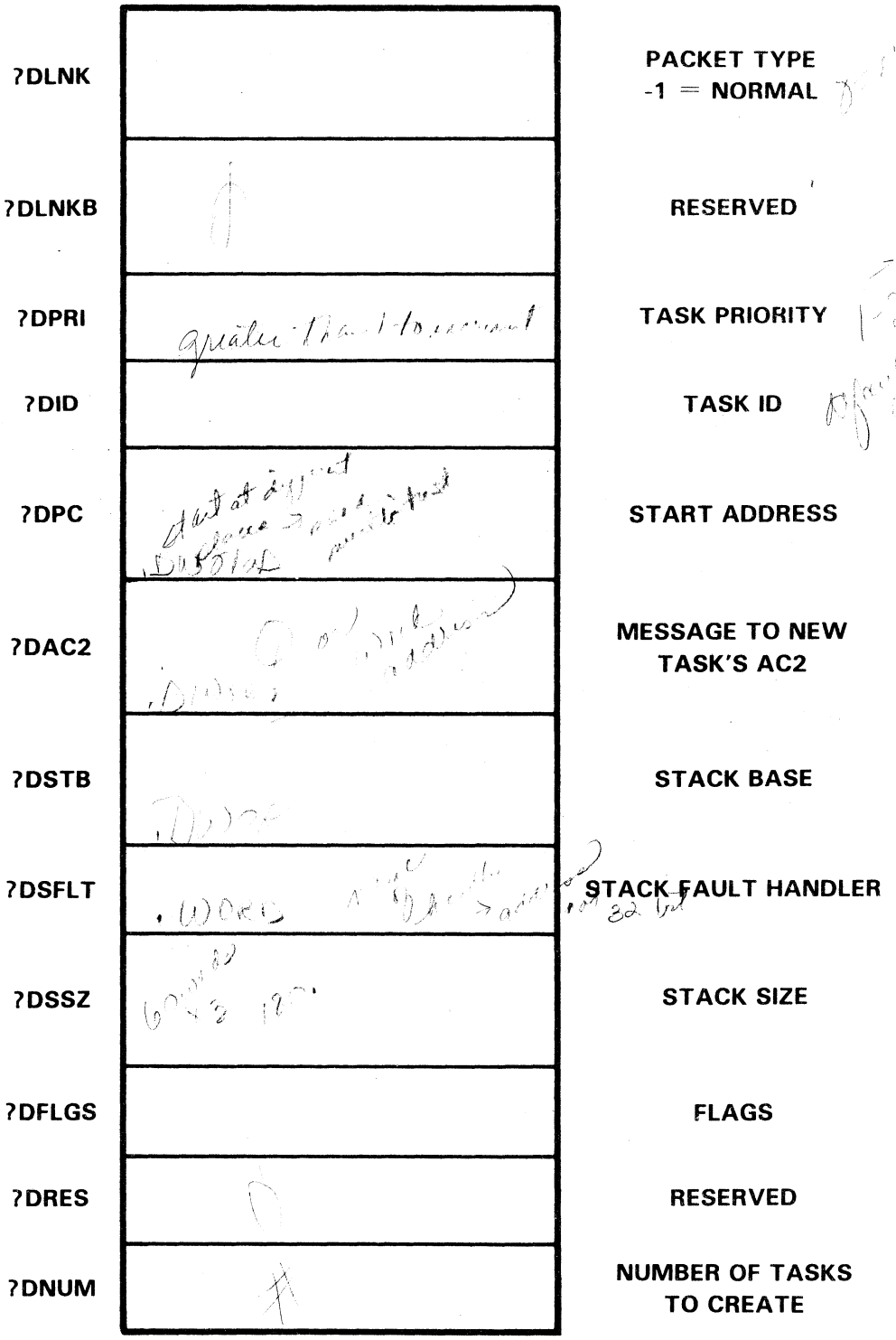
CS-04356

DEBUGGING A MULTITASK TCB DISPLAY

```
_1$K
TCB Status:      000000
System Call Word:        000257    (?POKE)
Task ID:         1.
Task Priority:   0.

Fixed Point Registers
        PC:      6001261311          Carry:  0
        AC0: 00000000044    AC1: 00000000044   AC2: 06000016404   AC3: 06000016656
Floating Point Registers
        FPAC0:   0.0000000000000000E+0      FPAC1:   0.0000000000000000E+0
        FPAC2:   0.0000000000000000E+0      FPAC3:   0.0000000000000000E+0
        FPPC:    00000000000        FPSR:     00000000000

Task Kill Post Processing Address:        0
USP Save Areas
        Ring 3: 06000005610
Current Descriptor:      00000000000
Overflow Mask:   00000000000

Stack:           SP              FP              SL              SB              Fault Addr
        Ring 3: 06000001710  06000001710  06000003753  06000001710  0
        Ring 7: 16000001670  00000000000  16000001763  16000001670  446
```

CS-04357

?TASK PACKET



length –?DSLTH

CS-04358

Task Stacks

Default Task

- Linker allocates 60. words at top of unshared partition
- or **X, LINK/STACK=n**

Stack Faults

- When stack overflow or underflow occurs, control passes to the stack fault handler routine
- For the default task, the handler must be named 'SFALT'
- The linker will include the default SFALT (from URT.32.LB) unless your program has an entry point named SFALT.
- The default SFALT just terminates your process with a stack fault error

Stacks for Other Tasks

- Set aside an unshared buffer of at least 60. words per task in unshared space.
- Put stack buffer address in ?TASK packet
- Put size of one task's stack in ?TASK packet

Reentrant Coding

- Is required when more than one task executes same code path
- Instructions which reference memory must not reference fixed addresses
- Memory references must be made relative to some unique per task item:
  - AC2 or AC3
  - Stack

Task Management Calls

All calls which use Task IDs expect the TID in AC1, and those which use task priority expect the priority in AC0.

Suspending and Readying Tasks

- ?SUS - Suspends the calling task
- ?IDSUS - Suspends task specified by TID
- ?PRSUS - Suspends all tasks of specified priority
- ?IDRDY - Readys task specified by TID
- ?PRRDY - Readys all tasks of specified priority

Delaying a Task

- ?WDELAY - Delay, in milliseconds, in AC0

Waiting for Control-C Sequences

- ?INTWT - Wait for Control-C, Control-A
- ?KWAIT - Wait for any Control-C sequence: AC0 receives second character
- ?KIOFF - Disable console interrupts
- ?KION - Enable console interrupts

Changing Task Priority

- ?PRI - Change priority of calling task
- ?IDPRI - Change priority of task specified by TID

?IDCO - al interrupt ch change path of PC

Killing Tasks

- ?KILL - Kills calling task
- ?IDKIL - Kills task specified by TID
- ?PRKIL - Kills all tasks of specified priority

MODULE 9

INTERTASK COMMUNICATION

Objectives

Upon completion of this module, the student should be able to:

1. Describe the use of intertask mailboxes.
2. Send and receive intertask messages.
3. Choose whether or not to broadcast intertask messages.
4. Synchronize tasks using Intertask Communication.
5. Describe locking a critical region using Intertask Communication.

Outline

- Intertask Communication
  - Mailboxes
  - ?REC
  - ?XMT
    - Broadcasting
  - ?RECNW
  - ?XMTW
  - Locking a Critical Region

Intertask Communication

- Provides method for synchronizing tasks
- Mailboxes
  - Double word memory locations
  - Zero indicates empty
  - Not owned by any task
  - Associated with some logical function
- Messages
  - Double word non-zero values
  - Sent to or received from a mailbox
  - Has no destination or origin task

- ?REC
  - Mailbox address in AC0
  - If mailbox is empty, task is suspended until a message is sent to mailbox
  - Message appears in AC1
  - Mailbox is zeroed when message is received

- ?XMT
  - Mailbox address in AC0
  - Message in AC1
  - Flag in AC2: -1 to broadcast message to all waiting receivers, else only highest priority receiver gets message and others remain suspended
  - If mailbox already contains a non-zero message, sender gets exception ERXMT

- ?RECNW - Like ?REC except task gets exception if mailbox is empty

- ?XMTW - Like ?XMT except sending task gets suspended until message is received

Locking A Critical Region

One example of using intertask communication comes from the classic literature of computer science. A critical region is a code path that may be executed by more than one task but which is not written reentrantly. Often a code path which is responsible for allocating and deallocating resources is a critical path.

Suppose a complex program has a pool of buffers and any task that needs a buffer will call the routine ALLOCATE. In addition, when the task no longer needs the buffer, it will call DEALLOCATE.

These two routines maintain a list of free buffers of various sizes. ALLOCATE searches the free list until it finds a buffer of the desired size, then it removes the buffer from the free list and returns its address to the caller.

But there is a possibility that a task executing ALLOCATE may be interrupted after the search but before removing the buffer from the free list. Following the servicing of the interrupt, it is possible that the scheduler would select a different task to run.

This other task might now call ALLOCATE, and ask for the same size buffer. It will find the same buffer that is about to be allocated to the first calling task!

Intertask communication can be used to lock this critical path:

```
ALLOC:  WSSVR   n
        LLEF    0,LOCKBOX
        ?REC
        WBR     ERROR
        ..
        ..
        ..
        ..
        ..
        ..
        LLEF    0,LOCKBOX
        WADC    1,1
        WSUB    2,2
        ?XMT
        WBR     ERROR
        WRTN
```

DataGeneral

Data General Corporation, 4400 Computer Drive, Westboro, MA 01580
(617) 366-8911