DECSYSTEM

Monitor Calls User's Guide

Order No. DEC-20-OMUGA-A-D

20

digital

Monitor Calls
User's Guide

Order No. DEC-20-OMUGA-A-D

digital equipment corporation · maynard, massachusetts

First Printing, May 1976

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECsystem-20	TYPESET-11

CONTENTS

FIGURES

TABLES

PREFACE

The DECsystem-20 Monitor Calls User's Guide is written for the assembly language user who is unfamiliar with the DECsystem-20. The manual introduces the user to the functions that he can request of the monitor from within his assembly language programs. The manual also teaches him how to use the basic monitor calls for performing these functions.

As such, this User's Guide is not a reference document, nor is it complete documentation on the entire set of monitor calls. It is organized according to functions, starting with the simple and proceeding to the more advanced. Each chapter should be read from beginning to end. A user who skips around in his reading will not gain the full benefit of this manual. Once the user has a working knowledge of the monitor calls in this document, he should then refer to the DECsystem-20 Monitor Calls Reference Manual (DEC-20-OMRMA-A-D) for the complete descriptions of all the calls.

To understand the examples in this manual, the user is assumed to be familiar with the MACRO language and the DECsystem-20 machine instructions. The DECsystem-20 MACRO Assembler Reference Manual (DEC-20-LMRMA-A-D) contains the documentation on the MACRO language. The Hardware Reference Manual (EK-DEC10-RF-001) contains the information on the machine instructions. These two manuals should be used in conjunction with the Monitor Calls User's Guide and should be referred to when questions arise on the MACRO language or the instruction set.

In addition, some of the examples in this manual contain macros and symbols (e.g., MOVX, TMSG, JSERR, JSHLT) from the MACSYM system file. This file is a universal file of definitions available to the user as a means of producing consistent and readable programs. The user should obtain a listing of this file for more information on its contents.

Finally, the user should be familiar with the TOPS-20 Command Language to enter and run the examples. The DECsystem-20 User's Guide (DEC-20-OUGAA-A-D) describes the TOPS-20 commands and system programs.

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

A program written in MACRO assembly language consists of a series of statements, each statement usually corresponding to one or more machine language instructions. Each statement in the MACRO program may be one of the following types:

1. A MACRO assembler directive, or pseudo-operation (pseudo-op), such as SEARCH or END. These pseudo-ops are commands to the MACRO assembler and are performed when the program is assembled. Refer to the DECsystem-20 MACRO Assembler Reference Manual for detailed descriptions of the MACRO pseudo-ops.

2. A MACRO assembler direct assignment statement. These statements are in the form

   ```
   symbol=value
   ```

   and are used to assign a specific value to a symbol. Assignment statements are processed by the MACRO assembler when the program is assembled. These statements do not generate instructions or data in the assembled program.

3. A MACRO assembler constant declaration statement, such as

   ```
   ONE:EXP 1
   ```

   These statements are processed when the program is assembled.

4. An instruction mnemonic, or symbolic instruction code, such as MOVE or ADD. These symbolic instruction codes represent the operations performed by the central processor when the program is executed. Refer to the Hardware Reference Manual for detailed descriptions of the symbolic instruction codes.

5. A monitor call, or JSYS, such as RESET or BIN. These calls are commands to the monitor and are performed when the program is executed. This manual describes the commonly-used monitor calls. However, the user should refer to the DECsystem-20 Monitor Calls Reference Manual for detailed descriptions of all the calls.

When the MACRO program is assembled, the MACRO assembler processes the statements in the program by

- translating symbolic instruction codes to binary codes.

- relating symbols to numeric values.
- assigning relocatable or absolute memory addresses.

The MACRO assembler also converts each symbolic call to the monitor into a Jump-to-System (JSYS) instruction.

1.2 MONITOR CALLS

Monitor calls are used to request monitor functions, such as input/output, error handling, and number conversions, during the execution of the program. These calls are accomplished with the JSYS instruction (operation code 104), where the address portion of the instruction indicates the particular function. This instruction is the only instruction that requests monitor functions at the assembly language level.

Each monitor call has a predefined symbol indicating the particular monitor function to be performed (e.g., OPENF to indicate opening a file). The symbols are defined in a system file called MONSYM. (Refer to DECsystem-20 Monitor Calls Reference Manual for a listing of the MONSYM file.) To use the symbols and to cause them to be defined correctly, the user's program must contain the statement

```
SEARCH MONSYM
```

at the beginning of the program. During the assembly of the program, the assembler replaces the monitor call symbol with an instruction containing the operation code 104 in the left half and the appropriate function code in the right half.

The JSYS instruction itself contains no data, nor does it contain space for returned data. Arguments for a JSYS instruction are placed in accumulators (ACs). Any data resulting from the execution of the JSYS instruction is returned in the accumulators or in an address in memory pointed to by an accumulator. Therefore, before the JSYS instruction can be executed, the appropriate arguments must be placed in the specific accumulators.

1.2.1 Calling Sequence

Arguments for the calls are placed in accumulators 1 through 4 (AC1-AC4). If more than four arguments are required for a particular call, the arguments are in a list pointed to by an accumulator. The arguments for the calls are specific bit settings or values. These bit settings and values are defined in MONSYM with symbol names, which can be used in the program. In fact, it is recommended that the user write his program using symbols whenever possible. This makes the program easier to read by another user. Use of symbols also allows the values of the symbols to be redefined without requiring the program to be changed. In this manual, the arguments for the monitor calls are described with both the bit settings and the symbol names. All program examples are written using the symbol names.

The set of instructions that place the arguments in the accumulators is followed by one line of code giving the particular monitor call symbol. During the program's execution, control is transferred to the monitor when this line of code is reached.

1.2.2 Returns

After the execution of the call, control returns to the user's program at one of two places. If an error occurs during the call's execution, control generally returns to the instruction immediately following the monitor call. In addition, an error code is stored in an accumulator to indicate the exact cause of the failure. This error code can be obtained by the program and translated into its corresponding error mnemonic and message string (refer to Appendix A for the list of error codes, mnemonics, and message strings). If the execution of the call is successful, control generally returns to the second instruction following the monitor call. Data returned from the execution of the call is stored in an accumulator or in an address pointed to by an accumulator.

However, for some monitor calls, only a single return to the instruction following the call occurs. On a successful return, that instruction is executed. If an error occurs during the execution of the call, the monitor examines the instruction following the call. If the instruction is a JUMP instruction with the AC field specified as either 16 or 17, the monitor transfers control to a user-specified address. If the instruction is not a JUMP instruction, the monitor generates a software interrupt indicating an illegal instruction, which the user's program can process via the software interrupt system (refer to Chapter 4). If the user's program is not prepared to process the interrupt, it is terminated, and a message is output stating the reason for failure.

To place a JUMP instruction in his program, the user can include a statement using one of two predefined symbols. These symbols are

```
ERJMP  address
ERCAL  address
```

and cause the assembler to generate a JUMP instruction. The JUMP instruction is a non-operation instruction (i.e., a no-op) as far as the hardware is concerned. However, the monitor executes the JUMP instruction by transferring control to the address specified, which is normally the beginning of an error processing routine written by the user. If the user includes the ERJMP symbol, control is transferred as though a JUMPA instruction had been executed, and control will not return to his program after the error routine is finished. If the user includes the ERCAL symbol, control is transferred as though a PUSHJ 17, address instruction had been executed. If the error routine executes a POPJ 17, instruction, control will return to the user's program at the location following the ERCAL.

The ERJMP and ERCAL symbols can be used after all monitor calls, regardless of whether the call has one or two returns. To handle errors consistently, users are encouraged to employ these symbols with all calls. The ERJMP or ERCAL is a no-op unless it immediately follows a monitor call that fails.

The following is an example of executing a monitor call (BIN, refer to Chapter 3) that has a single return. If the execution of the call is successful, the program reads and stores a character. If the execution of the call is not successful, the program transfers control to an error routine. This routine processes the error and then returns control back to the main program sequence. Note that the ERCAL stores the return address.

```
MOVE T1,INJFN                   ;obtain JFN for input file
BIN                             ;input one character
 ERCAL [PUSH P,T2               ;save character that was input
        GTSTS                   ;read file status
        TXNE T2,GS%EOF          ;end of file?
        JRST EOF                ;yes, process end-of-file condition
        HRROI T1,[ASCIZ/        ;no, data error
        ?INPUT ERROR, CONTINUING
        /]
        PSOUT                   ;print message
        POP P,T2                ;retrieve character that was input
        RET]                    ;return to program (POPJ 17,)
MOVEM T2,CHAR                   ;store character
```

1.3 PROGRAM ENVIRONMENT

The user program environment in the TOPS-20 operating system consists of a job structure that may contain many processes. A process is a runnable or schedulable entity capable of performing computations in parallel with other processes. This means that a runnable program is associated with at least one process.

Each process has its own independent 256K address space for storing its computations. This address space is called virtual space because it is actually a "window" into physical storage. Because the TOPS-20 operating system operates on pages, address spaces and storage are divided into 512 (decimal) pages, each of which is 512 words. (A word on the DECsystem-20 is 36 bits.)

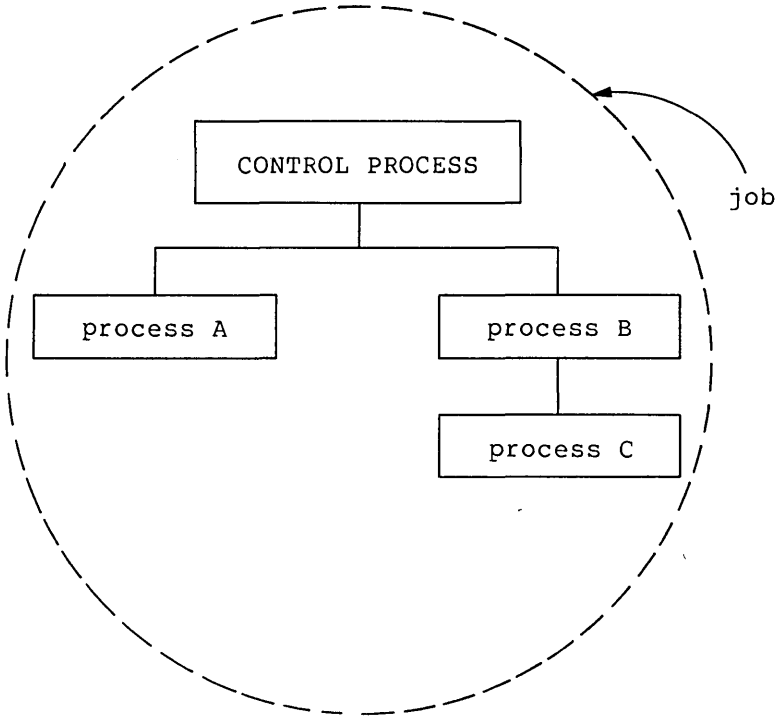
A process can communicate with other processes

- explicitly by software interrupts or system facilities (e.g., IPCF).
- implicitly by changing parts of its environment (i.e., its address space) that are being shared with other processes.

A process can create other processes inferior to it, but there is one control process from which the chain of creations begin. A process is said to exist when a superior process creates it and is said to end when a superior process deletes it. Refer to Chapter 5 for more information on the process structure.

A set of one or more related processes, normally under control of a single user, is a job. Each active process is part of some job on the system. A job is defined by a user name, an account number, some open files, and a set of running and/or suspended processes. This means that a job can be composed of several running or suspended programs.

The following diagram illustrates a job structure consisting of four processes.



Both process A and process B are created by the control process and thus are inferior to it. Process C is created by process B and thus is inferior to process B only.

In summary, processes can be considered as independent virtual machines with well-defined relationships to other processes in the system, and a job is a collection of these processes.

CHAPTER 2

INPUT AND OUTPUT USING THE TERMINAL

One of the main reasons for using monitor calls is to transfer data from one location to another. This chapter discusses moving data to and from the user's terminal.

2.1 OVERVIEW

Data transfers to and from the terminal are in the form of either individual bytes or text strings. The bytes are 7-bit bytes. The strings are ASCII strings ending with a 0 byte. These strings are called ASCIZ strings (i.e., a string of up to five 7-bit characters per word followed by a 7-bit character of zero).

To designate the desired string, the user's program must include a statement that points to the beginning of the string being read or written. The MACRO pseudo-op, POINT, can be used to set up this pointer, as shown in the following sequence of statements:

```
        MOVE AC1,PTR

             .
             .
PTR:    POINT 7,MSG
MSG:    ASCIZ/TEXT MESSAGE/
```

Accumulator 1 contains the symbolic address (PTR) of the pointer. At the address specified by PTR is the pointer to the beginning of the string. The pointer is set up by the POINT pseudo-op. The general format of the POINT pseudo-op is:

```
POINT decimal-byte-size,address,decimal-byte-position
```

(Refer to the MACRO Assembler Reference Manual for more information on the POINT pseudo-op.) In the example above, the POINT pseudo-op has been written to indicate 7-bit bytes starting at the left-most bit in the address specified by MSG.

Another way of setting up an accumulator to contain the address of the pointer is with the following statement:

```
HRROI AC1,[ASCIZ/TEXT MESSAGE/]
```

The instruction mnemonic HRROI causes a -1 to be placed in the left half of accumulator 1 and the address of the string to be placed in the right half. However, in the above statement, a literal (enclosed in square brackets) has been used instead of a symbolic address. The literal causes the MACRO assembler to:

- store the data within brackets (i.e., the string) in a table.
- assign an address to the first word of the data.
- insert that address as the operand to the HRROI instruction.

Literals have the advantage of showing the data at the point in the program where it will be used, instead of showing it at the end of the program.

As far as the I/O monitor calls are concerned, a word in this format (-1 in the left half and an address in the right half) designates the system's standard pointer (i.e., a pointer to a 7-bit ASCIZ string beginning at the leftmost byte of the string). The HRROI statement is interpreted by the monitor to be functionally equivalent to the word assembled by the POINT 7, address pseudo-op and is the recommended statement to use. However, byte manipulation instructions (e.g., ILDB, IBP, ADJBP) will not operate properly with this type of pointer.

After a string is read, the pointer is advanced to the character following the terminating character of the string. After a string is written, the pointer is advanced to the character following the last non-null character written.

2.2 PRIMARY I/O DESIGNATORS

To transfer data from one location to another, the user's program must indicate the source from which the data is to be obtained and the destination where the data is to be placed. The two designators used to represent the user's terminal are:

1. The symbol .PRIIN to represent the user's terminal as the source (input) device.
2. The symbol .PRIOU to represent the user's terminal as the destination (output) device.

These symbols are called the primary input and output designators and by convention are used to represent the terminal controlling the program. They are defined in the symbol file MONSYM and do not have to be defined in the user's program as long as the program contains the statement

```
SEARCH MONSYM
```

2.3 PRINTING A STRING

Many times a program may need to print an error message or some other string, such as a prompt to request input from the user at the terminal. The PSOUT (Primary String Output) monitor call is used to print such a string on the terminal. This call copies the designated string from the program's address space. Thus, the source of the data is the program's address space, and the destination for the data is the terminal. The program need only supply the pointer to the string being printed.

Accumulator 1 (AC1) is used to contain the address of the pointer. After AC1 is set up with the pointer to the string, the next line of code is the PSOUT call. Thus, an example of the PSOUT call is:

```
HRROI AC1,[ASCIZ/TEXT MESSAGE/]     ;string to print
PSOUT                               ;print TEXT MESSAGE
```

The ASCIZ pseudo-op specifies an ASCII string terminated with a null (i.e., 0) byte. The PSOUT call prints on the terminal all the characters in the string until it encounters a null byte. Note that the string is printed exactly as it is stored in the program, starting at the current position of the terminal's print head or cursor and ending at the last character in the string. If a carriage return and line feed are to be output, either before or after the string, these characters should be inserted as part of the string. For example, to print TEXT MESSAGE on one line and to output a carriage return-line feed after it, the user's program includes the call

```
        HRROI AC1,[ASCIZ/TEXT MESSAGE
/]
        PSOUT
```

After the string is printed, the instruction following the PSOUT call in the user's program is executed. Also, the pointer in AC1 is updated to point to the character following the last non-null character written.

If an error occurs during the execution of the call, the monitor looks for an ERJMP or ERCAL instruction as the next instruction following the call. If the next instruction is either one of these, the monitor transfers control to the address specified. If the next instruction is not an ERJMP or ERCAL, the monitor generates a software interrupt.

2.4 READING A NUMBER

The NIN (Number Input) monitor call is used to read an integer. This call does not assume the terminal as the source designator; therefore, the user's program must specify this. The NIN call accepts the number from any valid source designator, including a string in memory. This section discusses reading a number directly from the terminal. Refer to Section 2.9 for an example of using the NIN call to read the number from a string in memory. The destination for the number is AC2, and the NIN call places the binary value of the number read into this accumulator. The user's program also specifies a number in AC3 that represents the radix of the number being input. The radix given cannot be greater than base 10.

Thus, the setup for the NIN monitor call is the following:

```
MOVEI AC1,.PRIIN    ;AC1 contains the primary input designator
                    ;(i.e., the user's terminal)

MOVEI AC3,^D10      ;AC3 contains the radix of the number being
                    ;input (i.e., decimal number)

NIN                 ;The call to input the number
```

After completion of the NIN call, control returns to the program at one of two places (refer to Section 1.2.2). If an error occurs during the execution of the call, control returns to the instruction following the call. This instruction should be a jump-type instruction to an error processing routine. Also, an error code is placed in AC3 (refer to Appendix A for the error codes). If the execution of the NIN call is successful, control returns to the second instruction following the call. The number input from the terminal is placed in AC2.

The NIN call terminates when it encounters a nondigit character (e.g., a letter, a punctuation character, or a control character). This means that if 32X1 were typed on the terminal, on return AC2 would contain a 40 (octal) because the NIN call terminated when it read the X.

The following program prints a message and then accepts a decimal number from the user at the terminal. Note that the NIN call terminates reading on any nondigit character; therefore, the user cannot edit his input with any of the editing characters (e.g., DELETE, CTRL/W). The RDTTY call (refer to Section 2.9) should be used in programs that read from the terminal because it allows the user to edit his input as he is typing it.

```
        SEARCH MONSYM
        HRROI AC1,[ASCIZ/Enter # of seconds:  /]
        PSOUT                   ;output a prompt message
        MOVEI AC1,.PRIIN        ;input from the terminal
        MOVEI AC3,^D10          ;use the decimal radix
        NIN                     ;input a decimal number
         ERJMP NINERR           ;error-go to error routine
        MOVEM AC2, NUMSEC       ;save number entered
          .
          .
          .
NUMSEC:BLOCK 1
          .
          .
          .
```

2.5 WRITING A NUMBER

The NOUT (Number Output) monitor call is used to output an integer. The number to be output is placed in AC2. The user's program must specify the destination for the number in AC1 and the radix in which the number is to be output in AC3. The radix given cannot be greater than base 36. In addition, the user's program can specify certain formatting options to be used when printing the number.

Thus, the general setup for the NOUT monitor call is as follows:

AC1: output designator

AC2: number being output

AC3: format options in left half and radix in right half

The format options that can be specified in the left half of AC3 are described in Table 2-1.

Table 2-1
NOUT Format Options

Bit	Symbol	Meaning
0	NO%MAG	Print the number as a positive 36-bit number. For example, -1 would be printed as 777777 777777.
1	NO%SGN	Print the appropriate sign (+ or -) before the number. If bits NO%MAG and NO%SGN are both on, a plus sign is always printed.
2	NO%LFL	Print leading filler. If this bit is not set, trailing filler is printed.
3	NO%ZRO	Use 0's as the leading filler if the specified number of columns allows filling. If this bit is not set, blanks are used as the leading filler if the number of columns allows filling.
4	NO%OOV	Use the setting of bit 5 (NO%AST) if column overflows and give an error return. If this bit is not set, column overflow is not printed.
5	NO%AST	Print asterisks when the column overflows. If this bit is not set, and bit 4 (NO%OOV) is set, all necessary digits are printed when the columns overflow.
6-10		Reserved for DEC (must be zero).
11-17	NO%COL	Print the number of columns indicated. This value includes the sign column. If this field is 0, as many columns as necessary are printed.

Like the NIN call, the NOUT call returns control to the user's program at one of two places. Control returns to the instruction following the call if an error is encountered, and an error code is placed in AC3. Control returns to the second instruction following the call if no error is encountered.

The following example illustrates the use of the three monitor calls described so far. The RESET and HALTF monitor calls are described in Section 2.6.

```
        SEARCH MONSYM
START:  RESET
        HRROI AC1,[ASCIZ/PLEASE TYPE A DECIMAL NUMBER: /]
        PSOUT
        MOVEI AC1,.PRIIN            ;source designator
        MOVEI AC3,^D10              ;decimal radix
        NIN
         ERJMP ERROR
        HRROI AC1,[ASCIZ/THE OCTAL EQUIVALENT IS  /]
        PSOUT
        MOVEI AC1,.PRIOU
        MOVEI AC3,^D8               ;octal radix
        NOUT
         ERJMP ERROR
        HALTF                       ;return to command language
        JRST START                  ;begin again, if continued
ERROR:  HRROI AC1,[ASCIZ/
?ERROR-TYPE START TO BEGIN AGAIN/]
        PSOUT
        HALTF
        JRST START
        END START
```

2.6 INITIALIZING AND TERMINATING THE PROGRAM

Two monitor calls that have not yet been described were used in the above program - RESET and HALTF.

2.6.1 RESET Monitor Call

A good programming practice is to include the RESET monitor call at the beginning of every assembly language program. This call initializes the program's address space and closes any existing open files. The format of the call is

```
RESET
```

and control always returns to the next instruction following the call.

2.6.2 HALTF Monitor Call

To stop the execution of his program and to return control to the TOPS-20 Command Language, the user must include the HALTF monitor call as the last instruction performed in his program. He can then resume execution of his program at the instruction following the HALTF call by typing the CONTINUE command after control has been returned to command level.

2.7 READING A BYTE

The PBIN (Primary Byte Input) monitor call is used to read a single byte (i.e., one character) from the terminal. The user's program does not have to specify the source and destination for the byte because this call uses the primary input designator (i.e., the user's terminal) as the source and accumulator 1 as the destination. After execution of the PBIN call, control returns to the instruction

following the PBIN. If execution of the call is successful, the byte read from the terminal is right-justified in AC1. If execution of the call is not successful, a software interrupt (refer to Chapter 4) is generated if the user's program does not have, immediately after the PBIN call, an ERJMP or ERCAL instruction to an error routine.

2.8 WRITING A BYTE

The PBOUT (Primary Byte Output) monitor call is used to write a single byte to the terminal. This call uses the primary output designator (i.e., the user's terminal) as the destination for the byte; thus, the user's program does not have to specify the destination. The source of the byte being written is accumulator 1; therefore, the user's program must place the byte right-justified in AC1 before the call.

After execution of the PBOUT call, control returns to the instruction following the PBOUT. If execution of the call is successful, the byte is written to the user's terminal. If execution of the call is not successful, a software interrupt is generated if the user's program does not have, immediately after the PBOUT call, an ERJMP or ERCAL instruction to an error routine.

2.9 READING A STRING

Up to this point, monitor calls have been presented for printing a string, reading and writing an integer, and reading and writing a byte. The next call to be discussed obtains a string from the terminal and, in addition, allows the user at the terminal to edit his input as he is typing it.

The RDTTY (Read from Terminal) monitor call reads input from the user's terminal (i.e., from .PRIIN) into the program's address space. Input is read until the user either types an appropriate terminating (break) character or inputs the maximum number of characters allowed in the string, whichever occurs first. Output generated as a result of character editing is printed on the user's terminal (i.e., output to .PRIOU).

The RDTTY call handles the following editing functions:

1. Delete the last character in the string if the user presses the DELETE key while typing his input.

2. Delete back to the last punctuation character in the string if the user types CTRL/W while typing his input.

3. Delete the current line if the user types CTRL/U while typing his input.

4. Retype the current line if the user types CTRL/R while typing his input.

Because the RDTTY call can handle these editing functions, a program can accept input from the terminal and allow this input to be corrected by the user as he is typing it. For this reason, the RDTTY call should be used to read input from the terminal before processing that input with calls such as NIN.

The RDTTY call accepts three words of arguments in AC1 through AC3.

AC1: pointer to area in program's address space where input is to be placed. This area is called the text input buffer.

AC2: control bits in the left half, and maximum number of bytes in the text input buffer in the right half.

AC3: pointer to buffer for text to be output before the user's input if the user types a CTRL/R, or 0 if only the user's input is to be output on a CTRL/R.

The control bits in the left half of AC2 specify the characters on which to terminate the input. These bits are described in Table 2-2.

Table 2-2
RDTTY Control Bits

Bit	Symbol	Meaning
0	RD%BRK	Terminate input when user types a CTRL/Z or presses the ESC key.
1	RD%TOP	Terminate input when user types one of the following: CTRL/G CTRL/L CTRL/Z ESC key RETURN key Line feed key
2	RD%PUN	Terminate input when user types one of the following: CTRL/A-CTRL/F CTRL/H-CTRL/I CTRL/K CTRL/N-CTRL/Q CTRL/S-CTRL/T CTRL/X-CTRL/Y ASCII codes 34-36 ASCII codes 40-57 ASCII codes 72-100 ASCII codes 133-140 ASCII codes 173-176 The ASCII codes listed above represent the punctuation characters in the ASCII character set. Refer to an ASCII character set table for these characters.
3	RD%BEL	Terminate input when user types the RETURN or line feed key (i.e., end of line).
4	RD%CRF	Store only the line feed in the input buffer when the user presses the RETURN key. A carriage return will

Table 2-2 (Cont.)
RDTTY Control Bits

Bit	Symbol	Meaning
		still be output to the terminal but will not be stored in the buffer. If this bit is not set and the user presses the RETURN key, both the carriage return and the line feed will be stored as part of the input.
5	RD%RND	Return to program if the user attempts to delete past the beginning of his input. This allows the program to take control if the user tries to delete all of his input. If this bit is not set, the program waits for more input.
6		Reserved for DEC (must be zero).
7	RD%RIE	Return to program when there is no input (i.e., the text input buffer is empty). If this bit is not set, the program waits for more input.
8-9		Reserved for DEC (must be zero).
10	RD%RAI	Convert lower case input to upper case.
11	RD%SUI	Suppress the CTRL/U indication on the terminal when a CTRL/U is typed by the user. This means that if the user types a CTRL/U, XXX will not be printed and, on display terminals, the characters will not be deleted from the screen. If this bit is not set and the user types a CTRL/U, XXX will be printed and, if appropriate, the characters will be deleted from the screen. In neither case is the CTRL/U stored in the input buffer.
12-17		Reserved for DEC (must be zero).

If no control bits are set in the left half of AC2, the input will be terminated when the user presses the RETURN or line feed key (i.e., terminated on an end-of-line condition only).

The count in the right half of AC2 specifies the number of bytes available for storing the string in the program's address space. The input is terminated when this count is exhausted, even if a specified break character has not yet been typed.

The pointer in AC3 is to the beginning of a buffer containing the text to be output if the user types a CTRL/R. When this happens, the text in this separate buffer is output, followed by any text that has been typed by the user. The text in this buffer cannot be edited with any

of the editing characters (i.e., DELETE, CTRL/W, or CTRL/U). If the contents of AC3 is zero, then no such buffer exists, and if the user types CTRL/R, only the text in the input buffer will be output.

If execution of the RDTTY call is successful, the input is in the specified area in the program's address space. The character that terminated the input is also stored. (If the terminating character is a carriage return followed by a line feed, the line feed is also stored.) Control returns to the user's program at the second location following the call. The pointer in AC1 is advanced to the character following the last character read. The count in the right half of AC2 is updated, and appropriate bits are set in the left half of AC2. The bits that can be set on a successful return are:

Bit 12	RD%BTM	The input was terminated because one of the specified break characters was typed. This break character is placed in the input buffer. If this bit is not set, the input was terminated because the byte count was exhausted.
Bit 13	RD%BFE	Control was returned to the program because there is no more input and RD%RIE was set in the call.
Bit 14	RD%BLR	The limit to which the user can backup for editing his input was reached.

If execution of the RDTTY call is not successful, an error code is returned in AC1. Control returns to the user's program at the instruction following the RDTTY call.

The following example illustrates the recommended method for reading data from the terminal. This example is essentially the same as the one in Section 2.5; however, the RDTTY call is used to read the number before the NIN call processes it.

```
          SEARCH MONSYM
START:    RESET
          HRROI AC1,PROMPT
          PSOUT
          HRROI AC1,BUFFER
          MOVEI AC2,BUFLEN*5
          HRROI AC3,PROMPT
          RDTTY
           ERJMP ERROR
          HRROI AC1,BUFFER
          MOVEI AC3,^D10
          NIN
           ERJMP ERROR
          HRROI AC1,[ASCIZ/THE OCTAL EQUIVALENT IS /]
          PSOUT
          MOVEI AC1,.PRIOU
          MOVEI AC3,^D8
          NOUT
           ERJMP ERROR
          HALTF
          JRST START
```

```
PROMPT:   ASCIZ/PLEASE TYPE A DECIMAL NUMBER: /
          BUFLEN==10
BUFFER:   BLOCK BUFLEN
ERROR:    HRROI AC1,[ASCIZ/
?ERROR-TYPE START TO BEGIN AGAIN/]
          PSOUT
          HALTF
          JRST START
          END START
```

2.10 SUMMARY

Data transfers of sequential bytes or text strings can be made to and from the terminal. The monitor calls for transferring bytes are PBIN and PBOUT and for transferring strings are PSOUT and RDTTY. The NIN and NOUT monitor calls can be used for reading and writing a number. In general, the user's program must specify a source from which the data is to be obtained and a destination where the data is to be placed. In the case of terminal I/O, the symbol .PRIIN represents the user's terminal as the source, and the symbol .PRIOU represents the user's terminal as the destination.

CHAPTER 3

USING FILES

3.1 OVERVIEW

All information stored in the DECsystem-20 is kept in files. The basic unit of storage in a file is a page containing bytes from 1 to 36 bits in length. Thus, a sequence of pages constitutes a file. In most cases, files have names. Although all files are handled in the same manner, certain operations are unavailable for files on particular devices.

Programs can reference files by several methods:

- In a sequential byte-by-byte manner.
- In a multiple byte or string manner.
- In a random byte-by-byte manner if the particular file-storage device allows it.
- In a page-mapping manner for files on the disk.

Byte and string input/output are the most common types of operations.

Generally, all programs perform I/O by moving bytes of data from one location to another. For example, programs can move bytes from one memory area to another, from memory to a disk file, and from the user's terminal to memory. In addition, a program can map multiple 512-word pages from a disk file into memory or vice versa.

Data transfer operations on files require four steps:

1. Establishing a correspondence between a file and a Job File Number (JFN), because all files are referenced by JFNs.
2. Opening the file to establish the data mode, access mode, and byte size and to set up the monitor tables that permit data to be accessed.
3. Transferring data either to or from the file.
4. Closing the file to complete any I/O, to update the directory if the file is on the disk, and to release the monitor table space used by the file.

Some operations on files do not require the execution of all four steps above. Examples of these operations are: deleting or renaming a file, or changing the access code or account of a file. Although these operations do not require all four steps, they do require that the file has a JFN associated with it (step 1 above).

It is possible for disk files on the DECsystem-20 to be simultaneously read or written by any number of processes. To make sharing of files possible, all instances of opening a specific file in a specific directory cause a reference to the same data. Therefore, data written into a file by one process can immediately be seen by other processes reading the file.

Access to files is controlled by the 6-digit file access code assigned to a file when it is created. This code indicates the types of access allowed to the file for the three classes of users: the owner of the file, the users with group access to the file, and all other users. A user has group access to a file if he is in the same group as the directory in which the file resides is in. (Refer to the DECsystem-20 User's Guide for more information on the file access codes.) If the user is allowed access to a file, according to its file access code, he requests the type of access desired by including an OPENF monitor call (refer to Section 3.4) in his program. If the access requested in the OPENF call does not conflict with the current access to the file, the user is granted access. Essentially, the current access to the file is set by the first user who opens it.

Thus, for a user to be granted access to a specific file, two conditions must be met:

1. The file access code must allow the user to access the file in the desired manner (e.g., read, write).

2. The file must not be opened for a conflicting type of access.

3.2 JOB FILE NUMBER

The Job File Number (JFN) is one of the more important concepts in the operating system because it serves as the unique identifier of a particular file on a particular device during a process' execution. It is a small integer assigned by the system upon a request from the user's program. JFNs are usually assigned sequentially starting with 1.

The JFN is valid for the job in which it is assigned and therefore, may be used by any process in the job. The system uses the JFN as an index into the table of files associated with the job and thus, always assigns a JFN that is unique to the job. Even though a particular JFN within the job can refer to only one file, a single file can be associated with more than one JFN. This occurs when two or more processes are using the same file concurrently. In this case, each of the processes will probably have a different JFN for the file, but all of the JFNs will be associated with the same file.

3.3 ASSOCIATING A FILE WITH A JFN

In order to reference a file, the first step the user program must complete is to associate the specific file with a JFN. This correspondence is established with the GTJFN (Get Job File Number) monitor call. One of the arguments to this call is the string representing the desired file. The string can be specified within the program (i.e., come from memory) or can be accepted as input from the user's terminal or from another file. The string can represent the complete specification for the file:

dev:<directory>name.typ.gen;T(temporary);P(protection);A(account)

If any fields of the specification are omitted, the system can provide values for all except the name field. Refer to the DECsystem-20 User's Guide for a complete explanation of the specification for a file.

Table 3-1 lists the values the system will assign to fields not specified by the input string.

Table 3-1
Standard System Values For File Specifications

Field	Value
Device	DSK:
Directory	Directory to which user is currently connected.
Name	No default; this field must be specified.
Type	Null.
Generation number	The highest existing generation number if the file is an input file. The next higher generation number if the file is an output file.
Protection	Protection of next lower generation of file, if one exists; otherwise, protection as specified in the directory.
Account	Account specified when user logged in.

If the string specified identifies a single file, the monitor returns a JFN that remains associated with that file until either the process releases the JFN or the job logs off the system. After the assignment of the JFN is complete, the user's program uses the JFN in all references to that file.

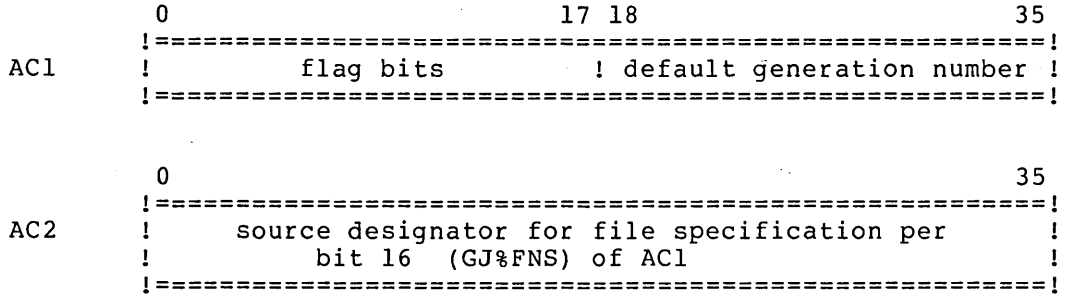
The user's program can set up either the short or the long form of the GTJFN monitor call. The short form indicates that the file specification is to be obtained from a string in memory or from a file, but not from both. Fields not specified by the input are taken from the standard system values for those fields (refer to Table 3-1). This form is sufficient for most uses of the call. The long form indicates that the file specification is to be obtained from both a string in memory and a file. If both are given as arguments, the string is used first, and then the file is used if more fields are needed to complete the specification. This form also allows the user's program to specify nonstandard values to be used for fields not given and to request the assignment of a specific JFN.

3.3.1 GTJFN Monitor Call

The GTJFN monitor call assigns a JFN to the specified file. It accepts two words of arguments. These argument words are different depending on the form of GTJFN being used. The user's program indicates the desired GTJFN form by setting bit 17 of AC1 to 1 for the short form or by clearing bit 17 to 0 for the long form.

3.3.1.1 Short Form Of GTJFN

- The short form of the GTJFN monitor call requires the following two words of arguments.



The flag bits that can be specified in AC1 are described in Table 3-2.

Table 3-2
GTJFN Flag Bits

Bit	Symbol	Meaning
0	GJ%FOU	The file specification given is to be assigned the next higher generation number. This bit indicates that a new version of a file is to be created and is normally set if the file is for output use.
1	GJ%NEW	The file specification given must not refer to an existing file (i.e., the file must be a new file).
2	GJ%OLD	The file specification given must refer to an existing file (i.e., the file must be an old file).
3	GJ%MSG	One of the appropriate messages is to be printed after the file specification is obtained. The message is printed only if the user types the ESC key to end his file specification (i.e., he is using recognition input). [NEW FILE] [NEW GENERATION] [OLD GENERATION]

Table 3-2 (Cont.)
GTJFN Flag Bits

Bit	Symbol	Meaning
		[OK] if GJ%CFM (bit 4) is off [CONFIRM] if GJ%CFM (bit 4) is on
4	GJ%CFM	Confirmation from the user will be required to verify that the file specification obtained is correct. To confirm the file specification, the user can press the RETURN key.
5	GJ%TMP	The file specified is to be a temporary file.
6	GJ%NS	Only the first file specification in a multiple logical name assignment is to be searched for the file.
7	GJ%ACC	The JFN specified is not to be accessed by inferior processes in this job. However, any process can access the file by acquiring a different JFN. To prevent the file from being accessed by other processes, the user's program can set OF%RTD (bit 29) in the OPENF call (refer to Section 3.4.1).
8	GJ%DEL	The file specified is not to be considered as deleted, even if it is marked as deleted.
9-10	GJ%JFN	These bits are off in the short form of the GTJFN call (refer to Section 3.3.1.2 for their description).
11	GJ%IFG	The file specification given is allowed to have one or more of its fields specified with a wildcard character (* or %). This bit is used to process a group of files and is generally used for input files. The monitor verifies that at least one value exists for each field that contains a wildcard and assigns the JFN to the first file in the group. The monitor also verifies that fields not containing wildcards represent a new or old file according to the setting of GJ%NEW and GJ%OLD.
12	GJ%OFG	The JFN is to be associated with the given file specification string only and not to the actual file. The string may contain a wildcard character (* or %) in one or more of its fields. It is checked for correct punctuation between fields, but is not checked for the validity of any field.

Table 3-2 (Cont.)
GTJFN Flag Bits

Bit	Symbol	Meaning
		This bit allows a JFN to be associated with a file specification even if the file specification does not refer to an actual file. The JFN returned cannot be used to refer to an actual file (e.g., cannot be used in an OPENF call) but can be used to obtain the original input string via the JFNS monitor call (refer to Section 3.7.2).
13	GJ%FLG	Flags are to be returned in the left half of AC1 on a successful return.
14	GJ%PHY	Logical names specified for the current job are to be ignored and the physical device is to be used.
15	GJ%XTN	This bit is off in the short form of the GTJFN call (refer to Section 3.3.1.2 for its description).
16	GJ%FNS	The contents of AC2 are to be interpreted as follows: 1. If this bit is on, AC2 contains an input JFN in the left half and an output JFN in the right half. The input JFN is used to obtain the file specification to be associated with the JFN. The output JFN is used to indicate the destination for printing the names of any fields being recognized. To omit either JFN, the user's program must specify the symbol .NULIO (377777). 2. If this bit is off, AC2 contains a pointer to a string in memory that specifies the file to be associated with the JFN.
17	GJ%SHT	This bit must be on for the short form of the GTJFN call.
18-35		The generation number of the file. The following values are permitted; however, 0 is the normal case. 0 to indicate that the next higher generation number is to be used if GJ%FOU (bit 0) is on, or to indicate that the highest existing generation number is to be used if GJ%FOU is off.

Table 3-2 (Cont.)
GTJFN Flag Bits

Bit	Symbol	Meaning
		1-377777 to indicate that the specified number is to be used as the generation if no generation number is supplied. -1 to indicate that the next higher generation number is to be used if no generation number is supplied. -2 to indicate that the lowest existing generation number is to be used if no generation number is supplied. -3 to indicate that all generation numbers are to be used and that the JFN is to be assigned to the first file in the group if no generation number is supplied. (Bit GJ%IFG must be set.)

If the GTJFN call is given with the appropriate flag bit set (GJ%IFG or GJ%OFG), the file specification given as input can have a wildcard character (either an asterisk or a percent sign) appearing in the directory, name, type, or generation number field. (The percent sign cannot appear in the generation number field.) The wildcard character is interpreted as matching any existing occurrence of the field. For example, the specification

```
<LIBARY>*.MAC
```

identifies all the files with the file type .MAC in the directory named <LIBARY>. The specification

```
<LIBARY>MYFILE.FO%
```

identifies all the files in directory <LIBARY> with the name MYFILE and a three-character file type in which the first two characters are .FO. Upon completion of the GTJFN call, the JFN returned is associated with the first file found in the group according to the following:

- in numerical order by directory number
- in alphabetical order by filename
- in alphabetical order by file type
- in ascending numerical order by generation number

The GNJFN (Get Next JFN) monitor call can then be given to assign the JFN to the next file in the group (refer to Section 3.7.3). Normally, a program that accepts wildcard characters in a file specification will successively reference all files in the group using the same JFN and not obtain another JFN for each one.

If execution of the GTJFN call is not successful because problems were encountered in performing the call, the JFN is not assigned and an error code is returned in the right half of AC1. The execution of the program continues at the instruction following the GTJFN call.

If execution of the GTJFN call is successful, the JFN assigned is returned in the right half of AC1 and various bits are set in the left half, if flag bits 11, 12, or 13 were on in the call. (The bits returned on a successful call are described in Table 3-3.) If bit 11, 12, or 13 was not on in the call, the left half of AC1 is zero. The execution of the program continues at the second instruction after the GTJFN call.

Table 3-3
Bits Returned on GTJFN Call

Bit	Symbol	Meaning
0-1		Reserved for DEC.
2	GJ%DIR	The directory field of the file specification contained wildcard characters.
3	GJ%NAM	The filename field of the file specification contained wildcard characters.
4	GJ%EXT	The file type field of the file specification contained wildcard characters.
5	GJ%VER	The generation number field of the file specification contained wildcard characters.
6	GJ%UHV	The file used has the highest generation number because a generation number of 0 was given in the call.
7	GJ%NHV	The file used has the next higher generation number because a generation number of 0 or -1 was given in the call.
8	GJ%ULV	The file used has the lowest generation number because a generation number of -2 was given in the call.
9	GJ%PRO	The protection field of the file specification was given.
10	GJ%ACT	The account field of the file specification was given.
11	GJ%TFS	The file specification is for a temporary file.
12	GJ%GND	Files marked for deletion will not be considered when assigning JFNs in subsequent calls.

Examples of the short form of the GTJFN monitor call are shown in the following paragraphs.

The following sequence of instructions is used to obtain, from the user's terminal, the specification of an existing file.

```
MOVSI AC1,(GJ%OLD+GJ%FNS+GJ%SHT)
MOVE AC2,[.PRIIN,,.PRIOU]
GTJFN
```

The bits specified for AC1 indicate that the file specification given must refer to an existing file (GJ%OLD), that the file specification is to be accepted from the input JFN in AC2 (GJ%FNS), and that the short form of the GTJFN call is being used (GJ%SHT). Because the right half of AC1 is zero, the standard generation number algorithm will be used. In this GTJFN call, the file with the highest existing generation number will be used/a Because GJ%FNS is set in AC1, the contents of AC2 are interpreted as containing an input JFN and an output JFN. In this example, the file specification is obtained from the terminal (.PRIIN).

The following sequence of instructions is used to obtain, from the user's terminal, the specification of an output file and to require confirmation from the user once the file specification has been obtained.

```
MOVSI AC1,(GJ%FOU+GJ%MSG+GJ%CFM+GJ%FNS+GJ%SHT)
MOVE AC2,[.PRIIN,,.PRIOU]
GTJFN
```

In this example, the bits specified for AC1 indicate that

- the file obtained is to be an output file (GJ%FOU),
- after the file specification is obtained, a message is to be typed (GJ%MSG),
- the user is required to confirm the file specification that was obtained (GJ%CFM),
- the file specification is to be obtained from the input JFN in AC2 (GJ%FNS),
- the short form of the GTJFN call is being used (GJ%SHT).

Because the right half of AC1 is zero, the generation number given to the file will be one greater than the highest generation number existing for the file. The contents of AC2 are interpreted as containing an input JFN and an output JFN because GJ%FNS is set in AC1.

The following sequence of instructions is used to obtain the name of an existing file from a location in the user's program.

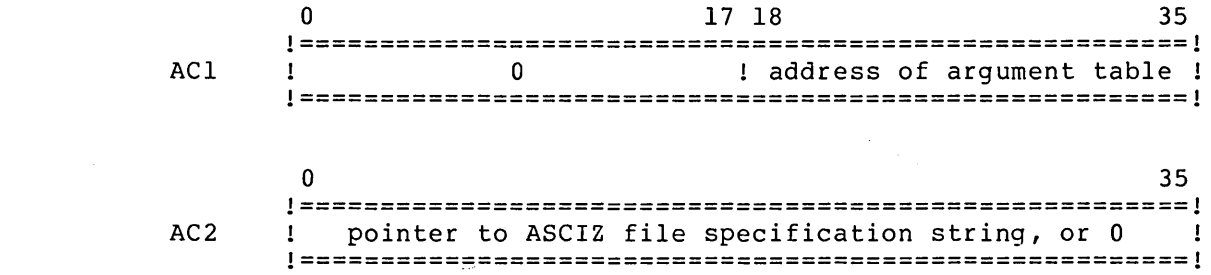
```
MOVSI AC1,(GJ%OLD+GJ%SHT)
MOVE AC2,[POINT 7,NAME]
GTJFN
   .
   .
   .

NAME:ASCIZ/MYFILE.TXT/
```

The bits specified for AC1 indicate that the file obtained is to be an existing file (GJ%OLD) and that the short form of the GTJFN call is being used (GJ%SHT). Since the right half of AC1 is zero, the file with the highest generation number will be used. Because GJ%FNS is not set, the contents of AC2 are interpreted as containing a pointer to a string in memory that specifies the file to be associated with the JFN. The setup of AC2 indicates that the string begins at location NAME in the user's program. The file specification obtained from location NAME is MYFILE.TXT.

An alternate way of specifying the same file is the sequence

```
MOVSI AC1,(GJ%OLD+GJ%SHT)
HRROI AC2,[ASCIZ/MYFILE.TXT/]
GTJFN
```

3.3.1.2 Long Form Of GTJFN - The long form of the GTJFN monitor call requires the following two words of arguments.



The argument table for the long form is described in Table 3-4 below.

Table 3-4
Long Form GTJFN Argument Block

Word	Symbol	Meaning
0	.GJGEN	Flag bits appear in the left half and generation number appears in the right half.
1	.GJSRC	An input JFN appears in the left half and an output JFN appears in the right half. To omit either JFN, the user's program must specify the symbol .NULIO (377777).
2	.GJDEV	Pointer to ASCIZ string that specifies the device to be used when none is given. If this word is 0, DSK will be used.
3	.GJDIR	Pointer to ASCIZ string that specifies the directory to be used when none is given. If this word is 0, the user's connected directory will be used.

Table 3-4 (Cont.)
Long Form GTJFN Argument Block

Word	Symbol	Meaning
4	.GJNAM	Pointer to ASCIZ string that specifies the filename to be used when none is given. If this word is 0, the input must specify the filename.
5	.GJEXT	Pointer to ASCIZ string that specifies the file type to be used when none is given. If this word is 0, a null type will be used.
6	.GJPRO	Pointer to ASCIZ string or 3B2+octal protection code. This word indicates the protection to be used when none is given. If this word is 0, the protection as specified in the directory will be used.
7	.GJACT	Pointer to ASCIZ string or 3B2+decimal account number. This word indicates the account to be used when none is given. If this word is 0, the account specified when the user logged in will be used.
10	.GJJFN	The JFN to assign to the file specification if flag bit GJ%JFN is set in word .GJGEN (word 0) of the argument block.
11-15		Additional words allowed if flag bit GJ%XTN (bit 15) is set in word .GJGEN (word 0) of the argument block. These additional words are used when performing command input parsing and are described in the DECsystem-20 Monitor Calls Reference Manual.

The flag bits accepted in the left half of .GJGEN (word 0) of the argument block are basically the same as those accepted in the short form of the GTJFN call. The entire set of flag bits is listed below. For further explanations of the bits, refer to Table 3-2.

Bit	Symbol	Meaning
0	GJ%FOU	A new version of the file is to be created.
1	GJ%NEW	The file must not exist.
2	GJ%OLD	The file must exist.
3	GJ%MSG	A message is to be typed if the user terminates his input with the ESC key.

4	GJ%CFM	The user must confirm the file specification.
5	GJ%TMP	The file is temporary.
6	GJ%NS	Only the first file specification is to be searched in a multiple logical name definition.
7	GJ%ACC	The JFN cannot be accessed by other processes in the job.
8	GJ%DEL	The "file deleted" bit is to be ignored.
9-10	GJ%JFN	The JFN supplied in .GJJFN(word 10) of the argument block is to be associated with the file specification given. The settings of bit 9 and 10 are interpreted as follows: 1. If bit 9 is on and bit 10 is off, an attempt is made to assign the JFN. An error return is given if the JFN is not available. 2. If bit 9 is on and bit 10 is on, an attempt is made to assign the JFN. If it is not available, some other JFN is assigned. 3. For any other combinations of these bits, the JFN supplied is ignored.
11	GJ%IFG	The file specification is allowed to contain wildcard characters.
12	GJ%OFG	The JFN is to be associated with the file specification string and not the file itself.
13	GJ%FLG	Flags are to be returned in AC1 on successful completion of the call.
14	GJ%PHY	The physical device is to be used.
15	GJ%XTN	The argument block contains more than 10 (octal) words. Refer to the DECsystem-20 Monitor Calls Reference Manual.
16	GJ%FNS	This bit is ignored for the long form of the GTJFN call.
17	GJ%SHT	This bit must be off for the long form of the GTJFN call.

The generation number values accepted in the right half of .GJGEN (word 0) of the argument block can be 0, -1, -2, -3, or a specified number, although 0 is the normal case. Refer to Bits 18-35 of Table 3-2 for explanations of these values.

If execution of the GTJFN call is successful, the JFN assigned is returned in the right half of AC1 and various bits are set in the left half if flag bits 11, 12 or 13 were on in the call. Refer to Table 3-3 for the explanations of the bits returned. Execution of the program continues at the second instruction following the call.

If execution of the GTJFN call is not successful, the JFN is not assigned and an error code is returned in the right half of AC1. The execution of the program continues at the instruction following the GTJFN call.

The following sequence of instructions obtains a specification for an existing file from the user's terminal, assigns the JFN to the next higher generation of that file, and specifies default fields to be used if the user omits a field when he gives his file specification.

```
          MOVEI AC1,JFNTAB
          SETZ AC2,
          GTJFN
            .
            .
            .

JFNTAB:   GJ%FOU
          XWD .PRIIN,.PRIOU
          0
          POINT 7,[ASCIZ/TRAIN/]    ;default directory
          0
          POINT 7,[ASCIZ/MEM/]      ;default file type
          0
          0
          0
```

The address of the argument table for the GTJFN call (JFNTAB) is given in the right half of AC1. AC2 contains 0, which means no pointer to a string is given; thus, fields for the file specification will be taken only from the user's terminal. The first word of the argument block contains a flag bit for the GTJFN call. This bit (GJ%FOU) indicates that the next higher generation number is to be assigned to the file. The second word of the argument block indicates that the file specification is to be obtained from the user's terminal, and any output generated because of the user employing recognition is to be printed on his terminal. If the user does not supply a directory name as part of his file specification, the directory <TRAIN> will be used. And if the user does not give a file type, the type MEM will be used. If the user omits other fields from his specification, the system standard value (refer to Table 3-1) will be used.

3.3.1.3 Summary Of GTJFN - The GTJFN monitor call is required to associate a JFN with a particular file. In most cases, the short form of the GTJFN call is sufficient for establishing this association. However, the long form is more powerful because it provides the user's program more control over the file specification that is obtained. The following summary compares the characteristics of the two forms of the GTJFN monitor call.

Short Form	Long Form
Assigns a JFN to a file. System decides the JFN to assign.	Assigns a JFN to a file. User program may request a particular JFN.
Accepts the file specification from a string in memory or a file.	Accepts the file specification from a string in memory and a file.
Uses standard system values for fields not given in the file specification.	Allows user-supplied values to be used for fields not given in the file specification.

3.4 OPENING A FILE

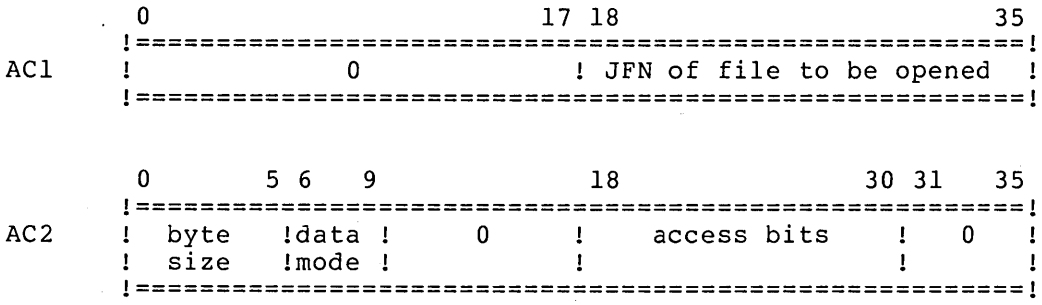
Once a JFN has been obtained for a file, the user's program must open the file in order to transfer data. The user's program supplies the JFN of the file to be opened and a word of bits indicating the desired byte size, data mode, and access to the file.

The desired access to the file is specified by a separate bit for each type of access. The file is successfully opened only if the desired access does not conflict with the current access to the file (refer to Section 3.1). For example, if the user requests both read and write access to the file, but write access is not allowed, then the file is not opened for this user. The allowed types of access to a file are:

- Read access. The file can be read with byte, string, or random input.
- Write access. The file can be written with byte, string, or random output.
- Append access. The file can be written only with sequential byte or dump output, and the current byte pointer (refer to Section 3.5.1) cannot be changed.
- Frozen access. The file can be concurrently accessed by at most one user writing the file, but by any number of users reading the file. This is the default access to a file.
- Thawed access. The file can be accessed even if other users are reading and writing the file.
- Restricted access. The file cannot be accessed if another user already has opened the file.

3.4.1 OPENF Monitor Call

The OPENF (Open File) monitor call opens a specified file. It requires the following two words of arguments.



If the left half of AC1 is not zero, the contents of AC1 is interpreted as a pointer to a string and not as a JFN of a file. Therefore, if the user's program requested bits to be returned in AC1 from the GTJFN call, it must clear these bits before executing the OPENF call.

The byte size (OF%BSZ) in AC2 specifies the number of bits in each byte of the file and can be between 1 and 36 (decimal). This field can be 0 if subsequent I/O to the file will be performed with the PMAP call (refer to Section 3.5.6).

The file data mode field (OF%MOD) can be one of two values:

Value	Meaning
0	Normal data mode of the file (i.e., byte I/O). Dump I/O is illegal.
17	Dump mode (i.e., unbuffered word I/O). Byte I/O is illegal and the byte size is ignored.

The access bits are described in Table 3-5.

Table 3-5
OPENF Access Bits

Bit	Symbol	Meaning
18	OF%HER	Halt on the occurrence of an I/O device or medium error during subsequent I/O to the file. If this bit is not set, a software interrupt is generated if a device or medium error occurs during subsequent I/O.
19	OF%RD	Allow read access.
20	OF%WR	Allow write access.
21		Reserved for DEC.
22	OF%APP	Allow append access.
23-24		Reserved for DEC.

Table 3-5 (Cont.)
OPENF Access Bits

Bit	Symbol	Meaning
25	OF%THW	Allow thawed access. If this bit is not set, the file is opened for frozen access.
26	OF%AWT	Block (i.e., temporarily suspend) the program until access to the file is permitted.
27	OF%PDT	Do not update the access dates of the file.
28	OF%NWT	Return an error if access to the file cannot be permitted.
29	OF%RTD	Allow access to the file to only one process (i.e., restricted access).
30	OF%PLN	Do not check for line numbers in the file.

If bits OF%AWT and OF%NWT are both off, an error code is returned if access to the file cannot be permitted (i.e., the action taken is identical to OF%NWT being on).

If execution of the OPENF monitor call is successful, the file is opened, and the execution of the program continues at the second instruction after the OPENF call.

If execution of the OPENF call is not successful, the file is not opened, and an error code is returned in AC1. The execution of the program continues at the next instruction after the OPENF call.

Two samples of the OPENF call follow.

The sequence of instructions below opens a file for input.

```
HRRZ AC1,JFNEXT
MOVE AC2,[44B5+OF%RD+OF%PLN]
OPENF
```

The JFN of the file to be opened is contained in the location indicated by the address in AC1 (JFNEXT). The bits specified for AC2 indicate that the byte size is one word (44B5), that read access is being requested to the file (OP%RD), and that no check will be made for line numbers in the file; i.e., the line numbers will not be discarded (OF%PLN). Because bit OF%THW is not set, the file can be accessed for reading by any number of processes.

The following sequence of instructions can be used to open a file for output.

```
MOVE AC1,JFN
MOVE AC2,[7B5+OF%HER+OF%WR+OF%AWT]
OPENF
```

The right half of AC1 contains the address that has the JFN of the file to be opened. The bits specified for AC2 indicate that the byte size is 7-bit bytes (7B5), that the program is to be halted when an I/O error occurs in the file (OF%HER), that write access is being requested to the file (OF%WR), and that the program is to be blocked if access cannot be granted (OF%AWT). Because bit OF%THW is not set, if another user has been granted write access to the file, this user's program will be blocked until access can be granted.

3.5 TRANSFERRING DATA

Data transfers of sequential bytes are the most common form of transfer and can be used with any file. For disk files, nonsequential bytes and entire pages can also be transferred.

3.5.1 File Pointer

Every open file is associated with a pointer that indicates the last byte read from or written to the file. When the file is initially opened, this pointer is normally positioned before the beginning of the file so that the first data operation will reference the first byte in the file. The pointer is then advanced through the file as data is transferred. However, if the file is opened for append-only access (bit OF%APP set in the OPENF call), the pointer is positioned after the last byte of the file. This allows the first write operation to append data to the end of the file.

For disk files, the pointer may be repositioned arbitrarily throughout the file, such as in the case of nonsequential data transfers. When the pointer is positioned beyond the end of the file, an end-of-file indication is returned when the program attempts a read operation using byte input. When the program performs a write operation beyond the end of the file using byte output, the end-of-file indicator is updated to point to the end of the new data. However, if the program writes pages beyond the end of the file with the PMAP monitor call (refer to section 3.5.6), the end-of-file indicator is not updated. Therefore, it is possible for a file to contain pages of data beyond the end-of-file indicator. To allow sequential I/O to be performed later to the file, the program should update the end-of-file indicator before closing the file. (Refer to the CHFDB monitor call description in the DECsystem-20 Monitor Calls Reference Manual.)

3.5.2 Source And Destination Designators

Because I/O operations occur by moving data from one location to another, the user's program must supply a source and a destination for any I/O operation. The most commonly-used source and destination designators are the following:

1. A JFN associated with a particular file. The JFN must be previously obtained with the GTJFN or GNJFN monitor call before it can be used.

2. The primary input and output designators .PRIIN and .PRIOU, respectively (refer to Section 2.2). These designators should be used when referring to the terminal.

3. A byte pointer to the beginning of the string of bytes in the program's address space that is being read or written. The byte pointer can take one of two forms:

   - A word with a -1 in the left half and an address in the right half. This form is used to designate a 7-bit ASCIZ string starting in the left-most byte of the specified address. A word in this form is functionally equivalent to a word assembled by the POINT 7,ADR pseudo-op.

   - A full word byte pointer with a byte size of 7 bits.

Most monitor calls dealing with strings deal specifically with ASCII strings. Normally, ASCII strings are assumed to terminate with a byte of 0 (i.e., are assumed to be ASCIZ strings). However some calls optionally accept an explicit byte count and/or terminating byte. These calls are generally ones that handle non-ASCII strings and byte sizes other than 7 bits.

3.5.3 Transferring Sequential Bytes

The BIN (Byte Input) and BOUT (Byte Output) monitor calls are used for sequential byte transfers. The BIN call takes the next byte from the given source and places it in AC2. The BOUT call takes the byte from AC2 and writes it to the given destination. The size of the byte is that given in the OPENF call for the file.

The BIN monitor call accepts a source designator in AC1, and upon successful execution of the call, the byte is right-justified in AC2. If execution of the call is not successful, a software interrupt is generated (refer to Chapter 4). Control returns to the user's program at the instruction following the BIN call.

The BOUT monitor call accepts a destination designator in AC1 and the byte to be output, right-justified in AC2. Upon successful execution of the call, the byte is written to the destination. If execution of the call is not successful, a software interrupt is generated (refer to Chapter 4). Control returns to the user's program at the instruction following the BOUT call.

The following sequence shows the transferring of bytes from an input file to an output file. The bytes are read from the file indicated by INJFN and written to the file indicated by OUTJFN.

```
LOOP:   MOVE 1,INJFN        ;get source designator from INJFN
        BIN                 ;read a byte from the source
        JUMPE 2,DONE        ;check for end of file, if 0
LOOP2:  MOVE 1,OUTJFN       ;get destination from OUTJFN
        BOUT                ;write the byte to the destination
        JRST LOOP           ;continue until 0 byte is found
DONE:   GTSTS               ;obtain status of source
        TLNN 2,(GS%EOF)     ;test for end of file
        JRST NOTYET         ;no, test for 0 in input file
        :                   ;yes, process end of file condition
NOTYET:MOVEI 2,0            ;0 in input file
        JRST LOOP2
```

3.5.4 Transferring Strings

The SIN (String Input) and SOUT (String Output) monitor calls are used for string transfers. These calls transfer either a string of a specified number of bytes or a string terminated with a specific byte.

The SIN monitor call reads a string from the specified source into the program's address space. The call accepts four words of arguments in AC1 through AC4.

AC1: source designator

AC2: pointer to area in program's address space

AC3: count of number of bytes to read, or 0

AC4: byte on which to terminate input (optional)

The contents of AC3 are interpreted as the number of characters to read.

- If AC3 is 0, then reading continues until a 0 byte is found in the input.
- If AC3 is positive, then reading continues until either the specified number of bytes is read, or a byte equal to that given in AC4 is found in the input, whichever occurs first.
- If AC3 is negative, then reading continues until minus the specified number of bytes is read.

The contents of AC4 needs to be specified only if the contents of AC3 is a positive number. The byte in AC4 is right-justified.

The input is terminated when one of the following occurs:

- The byte count becomes zero.
- The specified terminating byte is reached.
- The end of the file is reached.
- An error occurs during the transfer (e.g., a data error occurs).

Control returns to the user's program at the instruction following the SIN call. If an error occurs (including the end of the file is reached), a software interrupt is generated (refer to Chapter 4). In addition, several locations are updated:

1. The position of the file's pointer is updated for subsequent I/O to the file.
2. The pointer to the string in AC2 is updated to reflect the last byte read or, if AC3 contained 0, the last nonzero byte read.
3. The count in AC3 is updated, if pertinent, by subtracting the number of bytes actually read from the number of bytes requested to be read (i.e., the count is updated toward zero). From this count, the user's program can determine the number of bytes actually transferred.

The SOUT monitor call writes a string from the program's address space to the specified destination. Like the SIN call, this call accepts four words of arguments in AC1 through AC4.

AC1: destination designator

AC2: pointer to string to be written

AC3: count of the number of bytes to write, or 0

AC4: byte on which to terminate output (optional)

The contents of AC3 and AC4 are interpreted in the same manner as they are in the SIN monitor call.

The transfer is terminated when one of the following occurs.

- The byte count becomes zero.
- The specified terminating byte is reached. This terminating byte is written to the destination.
- An error occurs during the transfer.

Control returns to the user's program at the instruction following the SOUT call. If an error occurs, a software interrupt is generated (refer to Chapter 4). In addition, the position of the file's pointer, the pointer to the string in AC2, and the count in AC3, if pertinent, are also updated in the same manner as in the SIN monitor call.

The following sequence of instructions shows transferring a string from an input file to an output file. It is the same procedure as at the end of Section 3.5.3, but it uses SIN and SOUT calls instead of BIN and BOUT calls.

```
LOOP:  MOVE 1,INJFN        ;get source from INJFN
       HRROI 2,BUF128      ;pointer to string to read into (128
                           ;word buffer)
       MOVNI 3,^D128*5     ;input a maximum of 640 bytes
       SIN                 ;transfer until end of buffer or end of
                           ;file
        ERCAL EOFQ         ;error occurred
       ADDI 3,^D128*5      ;determine number of bytes transferred
       MOVN 3,3
       MOVE 1,OUTJFN       ;get destination from OUTJFN
       HRROI 2,BUF128      ;pointer to string to write from
       SOUT                ;transfer as many bytes as read
EOFQ:  MOVE 1,INJFN
       GTSTS               ;obtain status of source
       TLNN 2,(GS%EOF)     ;test for end of file
       RET                 ;no, continue copying
```

3.5.5 Transferring Nonsequential Bytes

As discussed in Section 3.5.3, the BIN and BOUT calls transfer bytes sequentially, starting at the current position of the file's pointer. The RIN (Random Input) and ROUT (Random Output) monitor calls allow the user's program to specify where the transfer will begin by accepting a byte number within the file. The size of the byte is the size given in the OPENF call for the file. The RIN and ROUT calls can only be used when transferring data to or from disk files.

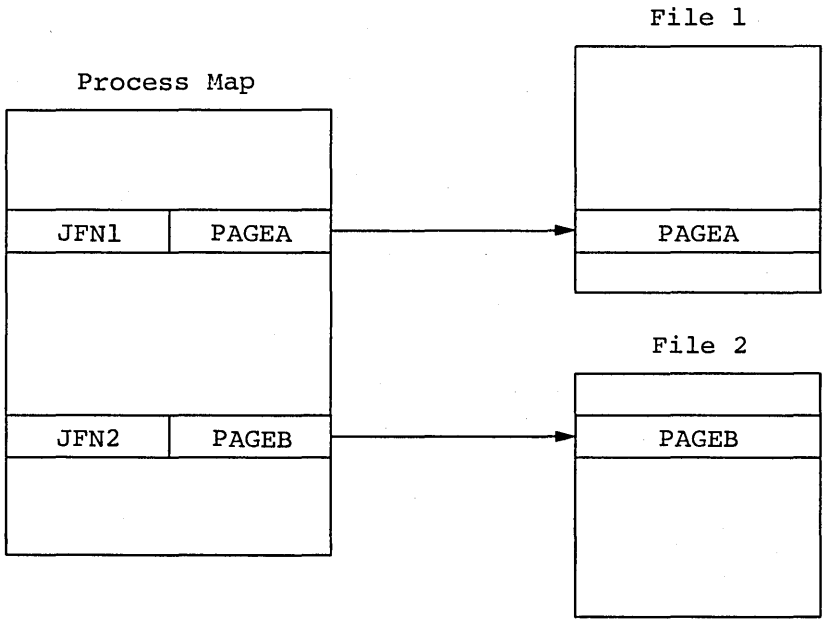
The RIN monitor call takes a byte from the specified location in the file and places it into the accumulator. The call accepts the JFN of the file in AC1 and the byte number within the file in AC3. Upon successful completion of the call, the byte is right-justified in AC2, and the file's pointer is updated to point to the byte following the one just read. If an error occurs, a software interrupt is generated (refer to Chapter 4). Control returns to the user's program at the instruction following the RIN call.

The ROUT monitor call takes a byte from the accumulator and writes it into the specified location in the file. The call accepts the JFN of the file in AC1, the byte to write right-justified in AC2, and the byte number within the file in AC3. Upon successful completion of the call, the byte is written into the specified byte in the file, and the file's pointer is updated to point to the byte following the one just written. If an error occurs, a software interrupt is generated (refer to Chapter 4). Control returns to the user's program at the instruction following the ROUT call.

3.5.6 Mapping Pages

Up to this point, monitor calls have been presented for transferring bytes of data. The next call to be discussed is used to transfer entire pages of data between a file and a process.

Both files and process address spaces are divided into pages of 512(decimal) words. A page within a file can be identified by one word, where the JFN of the file is in the left half and the page number within the file is in the right half. A page within a process address space can also be identified by one word, where the identifier of the process (refer to Section 5.3) is in the left half and the page number within the process' address space is in the right half. Each one-word identifier for the pages in the process address space is placed in what is called the process page map. When identifiers for file pages are placed in the process page map, references to the process page actually refer to the file page. The following diagram illustrates a process map that has identifiers for pages from two files.



The PMAP (Page Mapping) monitor call is used to map one or more entire pages from a file to a process (for input), from a process to a file (for output), or from one process to another process. In general, this call changes the entries in the process map by accepting file page identifiers and process page identifiers as arguments. Mapping pages between a file and a process is described below; mapping pages between two processes is described in Chapter 5.

3.5.6.1 Mapping File Pages To A Process - This use of the PMAP call changes the map of the process so that references to pages in the process reference pages in a file. This does not actually cause data to be transferred; it simply changes the contents of the map. Later when changes are made to the actual page in the process, the changes will also be made to the page in the file, if write access has been specified for the file.

The PMAP call accepts three words of arguments in AC1 through AC3.

AC1: JFN of the file in the left half, and the page number in the file in the right half

AC2: process identifier (refer to Section 5.3) in the left half, and page number in the process in the right half

AC3: repetition count and access

The repetition count and access bits that can be specified in AC3 are described below.

Bit	Symbol	Meaning
0	PM%CNT	Repeat the mapping operation the number of times specified by the right half of AC3. The file page number and the process page number are incremented by 1 each time the operation is performed.
2	PM%RD	Allow read access to the page.
3	PM%WR	Allow write access to the page.
9	PM%CPY	Create a private copy of the page if the process writes into the page. This is called copy-on-write and causes the map to be changed so that it identifies the copy instead of the original. Write access is allowed to the copy even if it was not allowed to the original. This allows a process to change a page of data without changing the data for other processes that have also mapped the page.
18-35		The number of times to repeat the mapping operation if bit 0(PM%CNT) is set.

With this use of the PMAP call, the present contents of the page in the process are removed. If the page in the file is currently nonexistent, it will be created when it is written.

This use of the PMAP call is valid only if the file is opened for at least read access. If write access is requested in the PMAP call, it is not granted unless it was also specified in the OPENF call when the file was opened.

A file cannot be closed while any of its pages are mapped into any process. Thus, before a file is closed, its pages must be unmapped (refer to Section 3.5.6.3).

After execution of the PMAP call, control returns to the user's program at the instruction following the call. If an error occurs, a software interrupt is generated (refer to Chapter 4).

3.5.6.2 Mapping Process Pages To A File - This use of the PMAP call actually transfers data by moving the specified page in the process to the specified page in the file. The process map for the page is now empty. Both the page in the process and the page in the file must be private; that is, no other process can have the page mapped into its address space. The ownership of the process page is transferred to the file page. The previous contents of the page in the file are deleted.

The three words of arguments are as follows:

AC1: process identifier (refer to Section 5.3) in the left half, and page number in the process in the right half

AC2: JFN of the file in the left half, and the page number in the file in the right half

AC3: repetition count and access (refer to Section 3.5.6.1)

The access requested in the PMAP call is granted only if it does not conflict with the access specified in the OPENF call when the file was opened.

This use of the PMAP call does not automatically update the end-of-file indicator and the file's byte size. To allow the file to be read later with sequential I/O monitor calls, the program should update the end-of-file indicator and the byte size. (Refer to the CHFDB monitor call in the DECsystem-20 Monitor Calls Reference Manual).

3.5.6.3 Unmapping Pages In A Process - As stated previously, a file cannot be closed if any of its pages are mapped in any process. To unmap a file's pages from a process, the program must execute the following form of the PMAP call:

AC1: -1

AC2: process identifier in the left half, and page number in the process in the right half.

AC3: the repeat count for the number of pages to remove from the process (refer to Section 3.5.6.1).

3.6 CLOSING A FILE

Once data has been transferred to or from a file, the user's program must close the file. When a file is closed, the system automatically performs the following:

1. Updates the directory information for the file. For example, for a file to which sequential bytes had been written, the byte size and byte count are updated when the file is closed.

2. Deassigns the JFN associated with the file. However, the user's program can request to close the file, but retain the JFN assignment. This is useful if the program plans to reopen the same file later, but does not want to execute another GTJFN call.

3.6.1 CLOSF Monitor Call

The CLOSF (Close File) monitor call closes either the specified file or all files that are opened for the process executing the call. The CLOSF call accepts one word of arguments in AC1 - flag bits in the left half and the JFN of the file to be closed in the right half. The flag bits are as follows:

Bit	Symbol	Meaning
0	CO%NRJ	Do not disassociate the JFN from the file.
6	CZ%ABT	Abort any output operations currently being done. That is, close the file but do not perform normal cleanup operations (e.g., do not output any data remaining in the buffers). If output to a new disk file that has not been closed is aborted, the file is closed and then deleted.

If the contents of AC1 is -1, all files that are opened for this process are closed.

If the execution of the CLOSF call is successful, the specified file is closed, and the JFN associated with the file is released if CO%NRJ was not set in the call. The execution of the user's program continues at the second location after the CLOSF call.

If the execution of the CLOSF call is not successful, the file is not closed and an error code is returned in the right half of AC1. The execution of the user's program continues at the instruction following the CLOSF call.

The following sequence illustrates the closing of two files.

```
CLOSIF:  MOVE 1,INJFN     ;obtain input JFN
         CLOSF            ;close input file
          ERJMP FATAL     ;if error, print message and stop
CLOSOF:  MOVE 1,OUTJFN    ;obtain output JFN
         CLOSF            ;close output file
          ERJMP FATAL     ;if error, print message and stop
```

3.7 ADDITIONAL FILE I/O MONITOR CALLS

3.7.1 GTSTS Monitor Call

The GTSTS (Get Status) monitor call obtains the status of a file. This call accepts one argument word - the JFN of the file in the right half of the AC1. The left half of AC1 is zero.

Control always returns to the user's program at the instruction following the GTSTS call. Upon return, appropriate bits reflecting the status of the specified JFN are set in AC2. These bits, and their meanings, are described in Table 3-6. Note that if the JFN is illegal or unassigned, bit 10 (GS%NAM) will not be set.

Table 3-6
Bits Returned on GTSTS Call

Bit	Symbol	Meaning
0	GS%OPN	The file is open. If this bit is not set, the file is not open.
1	GS%RDF	If the file is open (e.g., GS%OPN is set), it is open for read access.
2	GS%WRF	If the file is open, it is open for write access.
3		Reserved for DEC.
4	GS%RND	If the file is open, it is open for non-append access (i.e., its pointer can be reset).
5-6		Reserved for DEC.
7	GS%LNG	File has pages in existence beyond page number 511.
8	GS%EOF	The last read operation to the file was at the end of the file.
9	GS%ERR	The file may be in error (e.g., the bytes read may be erroneous).
10	GS%NAM	A file specification is associated with this JFN. This bit will not be set if the JFN is in any way illegal.
11	GS%AST	One or more fields of the file specification associated with this JFN contain a wildcard character.
12	GS%ASG	The JFN is currently being assigned (i.e., a process other than the one executing the GTSTS call is assigning this JFN).
13	GS%HLT	An I/O error is considered to be a terminating condition for this JFN. That is, the OPENF call for this JFN had bit OF%HER set.

Table 3-6 (Cont.)
Bits Returned on GTSTS Call

Bits	Symbol	Meaning
14-16		Reserved for DEC.
17	GS%FRK	Access to the file is restricted to only one process.
18-31		Reserved for DEC.
32-35		The data mode of the file (refer to the OPENF call). Value Symbol Meaning 0 .GSNRM Normal (sequential) I/O 10 .GSIMG Image (binary) I/O 17 .GSDMP Dump I/O

An example of the GTSTS call is shown in the first program in Section 3.9.

3.7.2 JFNS Monitor Call

The JFNS (JFN to String) monitor call returns the file specification currently associated with the specified JFN. The call accepts three words of arguments in AC1 through AC3.

AC1: destination designator where the file specification associated with the JFN is to be written. This specification is an ASCIZ string.

AC2: JFN or pointer to string (see below)

AC3: format to be used when returning the specification (see below)

The contents of AC1 can be any valid destination designator (refer to Section 3.5.2).

The contents of AC2 can be one of two formats. The first format is a word with either flag bits or 0 in the left half and the JFN in the right half. The bits that can be given in the left half of AC2 are the ones returned from the GTJFN call (refer to Table 3-3). When the left half of AC2 is nonzero (i.e., contains the bits returned from the GTJFN call), the string returned will contain wildcard characters for appropriate fields and 0, -1, or -2 as a generation number if the corresponding bit is on in the JFNS call. When the left half of AC2 is 0, the string returned is the exact specification for the file (e.g., wildcard characters are not returned for any fields). If the JFN is associated only with a file specification and not with an actual file (i.e., bit GJ%OFG was set in the GTJFN call), the string returned will contain null fields for unspecified fields and the actual values for specified fields. The second format allowed for AC2 is a pointer to the string in the program's address space that is to be returned upon execution of the call. Refer to the DECsystem-20 Monitor Calls Reference Manual for the explanation of this format.

The contents of AC3 specify the format in which the specification is written to the destination. Bits 0 through 20 are divided into 3-bit bytes, each byte representing a field in the file specification. The value of the byte indicates the format for that field. The possible values are:

0 Do not return this field when returning the file specification.
1 Always return this field when returning the file specification.
2 Suppress this field if it is the standard system value for this field (refer to Table 3-1).

If the contents of AC3 is zero, the file specification is written in the format

dev:<directory>name.typ.gen;T

with fields the same as the standard system value (see Table 3-1) not returned and protection and account fields returned only if bit 9 and bit 10 in AC2 are on, respectively. The temporary attribute (;T) is returned only if the file is temporary.

Table 3-7 describes the bits that can be set in AC3.

Table 3-7
JFNS Format Options

Bit	Symbol	Meaning
0-2	JS%DEV	Format for device field.
3-5	JS%DIR	Format for directory field.
6-8	JS%NAM	Format for filename field. A value of 2 (i.e., bit 7 set) for this field is illegal.
9-11	JS%TYP	Format for file type field. A value of 2 (i.e., bit 10 set) for this field is illegal.
12-14	JS%GEN	Format for generation number field.
15-17	JS%PRO	Format for protection field.
18-20	JS%ACT	Format for account field.
21	JS%TMP	Return temporary file indication ;T if the file specification is for a temporary file.
22	JS%SIZ	Return size of file in pages (see below).
23	JS%CRD	Return creation date of file (see below).
24	JS%LWR	Return date of last write operation to file (see below).
25	JS%LRD	Return date of last read operation from file (see below).

Table 3-7 (Cont.)
JFNS Format Options

Bit	Symbol	Meaning
26	JS%PTR	AC2 contains a pointer to the string containing the field to be returned (refer to the DECsystem-20 Monitor Calls Reference Manual for a description of this use of the JFNS call).
27-31		Reserved for DEC.
32	JS%PSD	Punctuate the size and date fields (see below) in the file specification returned.
33	JS%TBR	Place a tab before all fields returned (i.e., fields whose value is given as 1 in the 3-bit field) in the file specification, except for the first field.
34	JS%TBP	Place a tab before all fields that may be returned (i.e., fields whose value is given as 1 or 2 in the 3-bit field) in the file specification, except for the first field.
35	JS%PAF	Punctuate all fields (see below) returned in the file specification from the device field through the ;T field. If bits 32 through 35 are not set, no punctuation is used between the fields.

The punctuation used on each field is shown below. (The punctuation is underscored.)

dev:<directory>name.typ.gen;A(account);P(protection);T(temporary)
,size,creation date,write date,read date

Control always returns to the user's program at the instruction following the JFNS call. If an error occurs, a software interrupt is generated (refer to Chapter 4).

3.7.3 GNJFN Monitor Call

Occasionally a program may be written to perform similar operations on a group of files instead of only on one file. However, the program should not require the user to give a file specification for each file. Because the GTJFN call associates a JFN with only one file at a time, the program needs a method of assigning a JFN to all the files in the group. By using the GTJFN call to initially obtain the JFN and the GNJFN call to assign the same JFN to each subsequent file in the group, a program can accept a specification for a group of files and process each file in the group individually. After the user gives the initial file specification, the program requires no additional input.

Before an example showing the interaction of these two calls is given, a description of the GNJFN (Get Next JFN) monitor call is appropriate.

The GNJFN monitor call assigns a JFN to the next file in a group of files that have been specified with wildcard characters. The next file is determined by searching the directory in the order described in Section 3.3.1.1 using the current file as the first file. This call accepts one argument word in AC1 - the flags returned from the GTJFN call in the left half and the JFN of the current file in the right half. In other words, the information returned in AC1 from the GTJFN call is given as an argument to the GNJFN call. Therefore, the program must save this information for use with the GNJFN call.

If execution of the GNJFN call is successful, the same JFN is assigned to the next file in the group. The left half of AC1 contains various flags and the right half contains the JFN. The execution of the program continues at the second instruction after the GNJFN call.

The following bits can be returned in AC1 on a successful GNJFN call.

Bit	Symbol	Meaning
14	GN%DIR	A change in directory occurred between the previous file and this file.
15	GN%NAM	A change in filename occurred between the previous file and this file.
16	GN%EXT	A change in file type occurred between the previous file and this file. If GN%NAM is on, this bit will also be on because the system considers two files with different filenames but with the same file type as a change in both the name and type.

If execution of the GNJFN call is not successful, an error code is returned in the right half of AC1. Conditions that can cause an error return are:

1. The file currently associated with the JFN must be closed, and it is not. This means that the program must execute a CLOSF call (with CO%NRJ set to retain the JFN) before executing a GNJFN call.

2. There are no more files in this group. This return occurs on the first GNJFN call if no flags indicating wildcard fields are on in AC1 of the call. The JFN is released when there are no more files.

The execution of the program continues at the next instruction after the GNJFN call.

Consider the following situation. The user wants to write a program that will accept from his terminal a specification for a group of files and then perform an operation on each file individually without requiring additional input. Assume the user's directory <TRAIN> contains the following files:

```
FIRST.MAC.1
FIRST.REL.1
```

```
SECOND.REL.1
THIRD.EXE.1
```

As discussed in Section 3.3.1.1, a group of files can be given to the GTJFN call by supplying a specification that contains wildcard characters in one or more of its fields. Thus, the specification

```
<TRAIN>*.*
```

would refer to all four files in the user's directory <TRAIN>.

In his program, the user includes a GTJFN call that will accept the above specification.

The call is

```
MOVSI AC1,(GJ%OLD+GJ%IFG+GJ%FLG+GJ%FNS+GJ%SHT)
MOVE AC2,[.PRIIN,,.PRIOU]
GTJFN
```

and indicates that

1. The file specification given must refer to an existing file (GJ%OLD).

2. The file specification given is allowed to contain wildcard characters (GJ%IFG).

3. Flags will be returned in AC1 on a successful call (GJ%FLG). The flags must be returned because they will be given to the GNJFN call as arguments.

4. The contents of AC2 will be interpreted as containing an input and output JFN (GJ%FNS).

5. The short form of the GTJFN call is being used (GJ%SHT).

6. The file specification is to be read from the user's terminal (.PRIIN,,.PRIOU).

When the user types the specification <TRAIN>*.* as input, the system associates the JFN with one file only. This file is the first one found when searching the directory in the order specified in Section 3.3.1.1. Thus the JFN returned is associated with the file FIRST.MAC.1.

After the GTJFN call is successfully executed, AC1 contains appropriate flags in the left half and the JFN assigned in the right half. The flags that will be returned in this particular situation are:

GJ%NAM (bit 3)	A wildcard character appeared in the name field of the file specification given.
GJ%EXT (bit 4)	A wildcard character appeared in the type field of the file specification given.
GJ%GND (bit 12)	Any files marked for deletion will not be considered.

These flags inform the program of the fields that contained wildcard characters.

The user's program must now save the contents of AC1 because this word will be used as the argument to the GNJFN call. The program then performs its desired operation on the first file. Once its processing is completed, the program is ready for the specification of the next file. But instead of requesting the specification from the user, the program executes the GNJFN call to obtain it. The argument to the GNJFN call is the contents of AC1 returned from the previous GTJFN call. Thus, the call in this case is equivalent to:

```
MOVE AC1,[GJ%NAM+GJ%EXT+GJ%GND,,JFN]
GNJFN
```

Upon successful execution of the GNJFN call, the JFN is now associated with the next file in the group (i.e., FIRST.REL.1). AC1 contains appropriate flags in the left half and the same JFN in the right half. In this example, the flag returned is GN%EXT (bit 16) to indicate that the file type changed between the two files.

After processing the second file, the user's program executes another GNJFN call using the original contents of AC1 returned from the GTJFN call. The original contents must be used because this word indicates the fields containing wildcard characters. If the current contents of AC1 (i.e., the flags returned from the GNJFN call) are used, a subsequent GNJFN call would fail because there are no flags set indicating fields containing wildcard characters. This second GNJFN call associates the JFN with the file SECOND.REL.1. The flags returned in AC1 are GN%NAM (bit 15) and GN%EXT (bit 16) indicating that the filename and file type changed between the two files. (Remember that a change in filename implies a change in file type even if the two file types are the same.)

After processing this third file, the user's program executes another GNJFN call using the original contents of AC1. Upon execution of the call, the JFN is now associated with THIRD.EXE.1, and the flags returned are GN%NAM and GN%EXT, indicating a change in the filename and file type.

After processing the file THIRD.EXE.1, the user's program executes a final GNJFN call. Since there are no more files in the group, the call returns an error code and releases the JFN. Execution of the user's program continues at the instruction following the GNJFN call.

3.8 SUMMARY

To read from or write to a file, the user's program must:

1. Obtain a JFN on the file with the GTJFN monitor call (refer to Section 3.3.1).

2. Open the file with the OPENF monitor call (refer to Section 3.4.1).

3. Transfer the data with byte, string, or page I/O monitor calls (refer to Section 3.5).

4. Close the file with the CLOSF monitor call (refer to Section 3.6.1).

3.9 FILE EXAMPLES

Example 1 - This program assigns JFNs, opens an input file and an output file, and copies data from the input file to the output file. Data is copied until the end of the input file is reached. Refer to the DECsystem-20 Monitor Calls Reference Manual for explanation of the ERSTR monitor call.

```
;*** PROGRAM TO COPY INPUT FILE TO OUTPUT FILE. ***
;       (USING BIN/BOUT AND IGNORING NULL'S)

        TITLE FILEIO                ;TITLE OF PROGRAM
        SEARCH MONSYM               ;SEARCH SYSTEM JSYS-SYMBOL LIBRARY

;*** IMPURE DATA STORAGE AND DEFINITIONS ***


INJFN:  BLOCK 1                     ;STORAGE FOR INPUT JFN
OUTJFN: BLOCK 1                     ;STORAGE FOR OUTPUT JFN

        PDLEN=3                     ;STACK HAS LENGTH 3
PDLST:  BLOCK PDLEN                 ;SET ASIDE STORAGE FOR STACK

A==1                                ;JSYS AC'S
B==2
C==3
D==4
T1==5                               ;TEMPORARY AC'S
                                    ;....
P==17                               ;PUSH DOWN POINTER

;*** PROGRAM INITIALIZATION ***

START:  RESET                       ;CLOSE FILES, ETC.
        MOVE P,[IOWD PDLEN,PDLST]   ;ESTABLISH STACK

;*** GET INPUT-FILE ***

INFIL:  HRROI A,[ASCIZ /
INPUT FILE: /]                      ;PROMPT FOR INPUT FILE
        PSOUT                       ;ON CONTROLLING  TERMINAL
        MOVE A,[GJ%OLD+GJ%FNS+GJ%SHT];SEARCH MODES FOR GTJFN
                                    ;[EXISTING FILE ONLY , FILE-NR'S IN B
                                    ; SHORT CALL ]

        MOVE B,[.PRIIN,,.PRIOU]     ;GTJFN'S I/O WITH CONTROLLING TERMINAL
        GTJFN                       ;GET JOB FILE NUMBER (JFN)
         JRST [ PUSHJ P,WARN        ;IF ERROR, GIVE WARNING
                JRST INFIL]         ;AND LET HIM TRY AGAIN
        MOVEM A,INJFN               ;SUCCESS, SAVE THE JFN
```

```
;*** GET OUTPUT-FILE ***

OUTFIL: HRROI A,[ASCIZ /
OUTPUT FILE: /]                    ;PROMPT FOR OUTPUT FILE
        PSOUT                      ;PRINT IT
        MOVE A,[GJ%FOU+GJ%MSG+GJ%CFM+GJ%FNS+GJ%SHT];GTJFN SEARCH MODES
                                   ;[DEFAULT TO NEW GENERATION , PRINT
                                   ; MESSAGE , REQUIRE CONFIRMATION
                                   ; FILE-NR'S IN B , SHORT CALL ]

        MOVE B,[.PRIIN,,.PRIOU]    ;I/O WITH CONTROLLING TERMINAL
        GTJFN                      ;GET JOB-FILE NUMBER
         JRST [ PUSHJ P,WARN       ;IF ERROR, GIVE WARNING
                JRST OUTFIL]       ;AND LET HIM TRY AGAIN
        MOVEM A,OUTJFN             ;SAVE THE JFN


;NOW, OPEN THE FILES WE JUST GOT

;    INPUT

        MOVE A,INJFN               ;RETRIEVE THE INPUT JFN
        MOVE B,[7B5+OF%RD]         ;DECLARE MODES FOR OPENF [7-BIT BYTES +
                                   ;INPUT]
        OPENF                      ;OPEN THE FILE
         JRST FATAL                ;IF ERROR, GIVE MESSAGE AND STOP

;    OUTPUT

        MOVE A,OUTJFN              ;GET THE OUTPUT JFN
        MOVE B,[7B5+OF%WR]         ;DECLARE MODES FOR OPENF [7-BIT BYTES +
                                   ;OUTPUT]
        OPENF                      ;OPEN THE FILE
         JRST FATAL                ;IF ERROR, GIVE MESSAGE AND STOP


;*** MAIN LOOP ;COPY BYTES FROM INPUT TO OUTPUT ***

LOOP:   MOVE A,INJFN               ;GET THE INPUT JFN
        BIN                        ;TAKE A BYTE FROM THE SOURCE
        JUMPE B,DONE               ;IF 0, CHECK FOR END OF FILE.
        MOVE A,OUTJFN              ;GET THE OUTPUT JFN
        BOUT                       ;OUTPUT THE BYTE TO DESTINATION
        JRST LOOP                  ;LOOP, STOP ONLY ON A 0 BYTE (FOUND
                                   ;AT LOOP+2)

;*** TEST FOR END OF FILE, ON SUCCESS FINISH UP ***

DONE:   GTSTS                      ;GET THE STATUS OF INPUT FILE.
        TLNN B,(GS%EOF)            ;AT END OF FILE?
        JRST LOOP                  ;NO, FLUSH NULL AND CONTINUE COPY

CLOSIF: MOVE A,INJFN               ;YES, RETRIEVE INPUT JFN
        CLOSF                      ;CLOSE INPUT FILE
         JRST FATAL                ;IF ERROR, GIVE MESSAGE AND STOP

CLOSCF: MOVE A,OUTJFN              ;RETRIEVE OUTPUT JFN
        CLOSF                      ;CLOSE OUTPUT FILE
         JRST FATAL                ;IF ERROR, GIVE MESSAGE AND STOP

        HRROI A,[ASCIZ/
[DONE]/]                           ;SUCCESSFULLY DONE
        PSOUT                      ;PRINT IT
        JRST ZAP                   ;STOP
```

```
;*** ERROR HANDLING ***

FATAL:  HRROI A,[ASCIZ/
?/]                             ;FATAL ERRORS PRINT ? FIRST
        PUSHJ P,ERROR           ;THEN PRINT ERROR MESSAGE,
        JRST ZAP                ;AND STOP

WARN:   HRROI A,[ASCIZ/
%/]                             ;WARNINGS PRINT % FIRST
                                ; AND FALL THRU 'ERROR' BACK TO CALLER

ERROR:  PSOUT                   ;PRINT THE ? OR %
        MOVE A,[.PRIOU]         ;DECLARE PRINCIPAL OUTPUT DEVICE FOR
                                ;ERROR MESSAGE
        MOVE B,[.FHSLF,,-1]     ;CURRENT FORK,, LAST ERROR
        SETZ C,                 ;NO LIMIT,, FULL MESSAGE
        ERSTR                   ;PRINT THE MESSAGE
         JFCL                   ;IGNORE UNDEFINED ERROR NUMBER
         JFCL                   ;IGNORE ERROR DURING EXECUTION OF ERSTR
        POPJ P,                 ;RETURN TO CALLER


ZAP:    HALTF                   ;STOP
        JRST START              ;WE ARE RESTARTABLE
        END START               ;TELL LINKING LOADER START ADRESS
```

Example 2 - This program accepts input from a user at the terminal and then outputs the data to the line printer. Refer to Section 2.9 for explanation of the RDTTY call.

```
TITLE LPTPNT - PROGRAM TO PRINT TERMINAL INPUT ON THE PRINTER

        SALL
        SEARCH MACSYM,MONSYM
        .REQUIRE SYS:MACREL

        T1=1
        T2=2
        T3=3
        T4=4

        P=17

BUFSIZ==200
PDLEN==50

COUNT:  BLOCK 1
BUFFER: BLOCK BUFSIZ
PDL:    BLOCK PDLEN

START:  RESET                       ;RESET I/O, ETC.
        HRROI T1,[ASCIZ/ENTER TEXT TO BE PRINTED (END WITH ^Z):

/]                                  ;GET POINTER TO PROMPTING TEXT
        PSOUT                       ;OUTPUT PROMPTING MESSAGE
        HRROI T1,BUFFER             ;GET POINTER TO BUFFER
        MOVE T2,[RD%BRK+BUFSIZ*5]   ;GET FLAG AND MAX # OF CHARACTERS TO
                                    ;READ
        SETZM T3                    ;NO RE-TYPE BUFFER
        RDTTY                       ;INPUT TEXT FROM THE TERMINAL
         JSHLT                      ;ERROR, STOP
        ADD T2,BUFSIZ*5             ;COMPUTE NUMBER OF CHARACTERS READ
        MOVEM T2,COUNT              ;SAVE # OF CHARACTERS INPUT

; GET A JFN FOR THE PRINTER AND OPEN THE PRINTER

        MOVSI T1,(GJ%SHT!GJ%FOU)    ;OUTPUT FILE, SHORT CALL
        HRROI T2,[ASCIZ /LPT:/]     ;GET POINTER TO NAME OF FILE
        GTJFN                       ;GET A JFN FOR THE PRINTER
         JRST JFNERR                ;ERROR, PRINT ERROR MESSAGE
        MOVE T2,[7B5+OF%WR]         ;7-BIT BYTES, WRITE ACCESS WANTED
        OPENF                       ;OPEN THE PRINTER FOR OUTPUT
         JRST OPNERR                ;ERROR, PRINT ERROR MESSAGE

; NOW OUTPUT THE TEXT WHICH WAS INPUT FROM THE TERMINAL

        HRROI T2,BUFFER             ;GET POINTER TO TEXT (PRINTER JFN STILL
                                    ;IN T1)
        MOVN T3,COUNT               ;GET NUMBER OF CHARACTERS TO OUTPUT
        SOUT                        ;OUTPUT STRING OF CHARACTERS TO THE
                                    ;PRINTER
         ERJMP DATERR               ;ERROR, PRINT ERROR MESSAGE
        HRROI T1,[ASCIZ/
OUTPUT HAS BEEN SENT TO THE PRINTER...
/]
        PSOUT                       ;OUTPUT CONFIRMATION MESSAGE
        HALTF                       ;FINISHED
        JRST START                  ;IF CONTINUED, GO BACK TO START
```

```
; ERROR ROUTINES

JFNERR: HRROI T1,[ASCIZ/
? COULD NOT GET A JFN FOR THE PRINTER
/]
        HALTF
        JRST START

OPNERR: HRROI T1,[ASCIZ/
? COULD NOT OPEN THE PRINTER FOR OUTPUT
/]
        HALTF
        JRST START

DATERR: HRROI T1,[ASCIZ/
? DATA ERROR DURING OUTPUT TO PRINTER
/]
        HALTF
        JRST START

        END START
```

CHAPTER 4

USING THE SOFTWARE INTERRUPT SYSTEM

4.1 OVERVIEW

Program execution usually occurs in a sequential manner, whereby one instruction is executed immediately followed by the next one. However, there are many occasions when a program must be able to receive asynchronous signals from terminals or other programs or as a result of its own execution. By using the software interrupt system, the user can specify certain conditions that will cause his program to deviate from its sequential method of execution.

An interrupt is defined as a break in the normal flow of control during a program's execution. The break, or interrupt, is caused by the occurrence of a prespecified condition. By specifying the conditions that can cause an interrupt, the program has the capability of dynamically responding to external events and error conditions and of generating requests for services. Because the program can respond to special conditions as they occur, it does not have to explicitly and repeatedly test for them. In addition, the program's size is reduced and its execution is faster because the program does not have to include a special test after the possible occurrence of the condition.

When an interrupt occurs, the system transfers control from the main program sequence to a previously-specified routine that will process the interrupt. After the routine has completed its processing of the interrupt, the system can transfer control back to the program at the point it was interrupted, and execution can continue. See Figure 4-1.

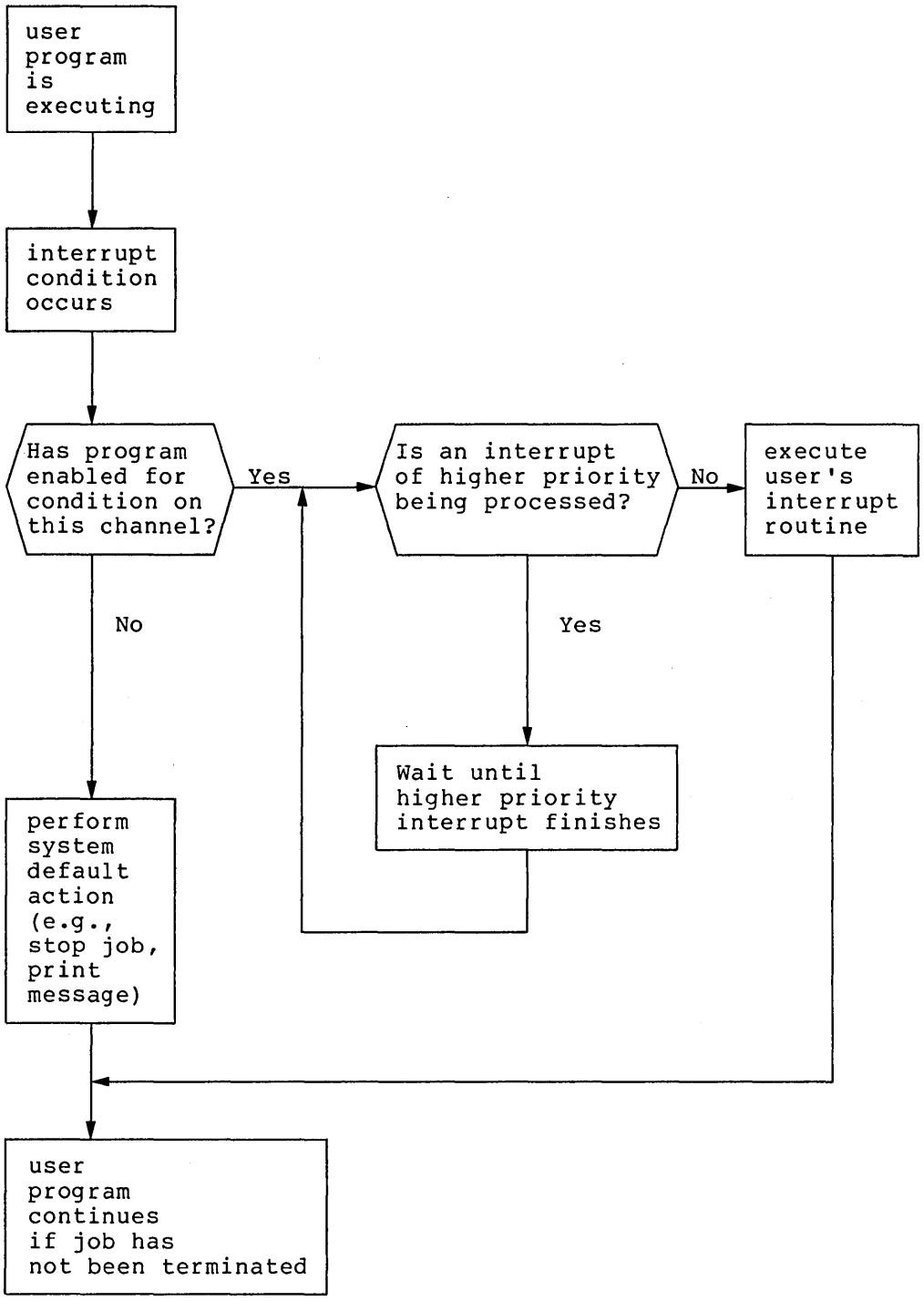


Figure 4-1 Basic Operational Sequence of the Software Interrupt System

4.2 INTERRUPT CONDITIONS

Conditions that cause the program to be interrupted when the interrupt system is enabled are:

1. Conditions generated when specific terminal keys are typed. There are 36 possible codes; each one specifies the particular terminal character or condition on which an interrupt is to be initiated. Refer to Table 4-2 for the possible codes.

2. Invalid instructions (e.g., I/O instructions given in user mode) or privileged monitor calls.

3. Memory conditions, such as a reference to unassigned memory.

4. Arithmetic processor conditions, such as arithmetic overflow or underflow.

5. Certain file or device conditions, such as end of file.

6. Program-generated software interrupts.

7. Termination of an inferior process.

8. System resource unavailability.

9. Interprocess communication (IPCF) and Enqueue/Dequeue interrupts.

4.3 SOFTWARE INTERRUPT CHANNELS AND PRIORITIES

Each condition is associated with one of 36 software interrupt channels. Most conditions are permanently assigned to specific channels; however, the user's program can associate some conditions (e.g., conditions generated by specific terminal keys) to any one of the assignable channels. (Refer to Table 4-1 for the channel assignments.) When the condition associated with a channel occurs, and that channel has been activated, an interrupt is generated. Control can then be transferred to the routine responsible for processing interrupts on that channel.

The user program assigns each channel to one of three priority levels. Priority levels allow the occurrence of some conditions to suspend the processing of other conditions. The levels are referred to as level 1, 2, or 3 with level 1 having the highest priority. Level 0 is not a legal priority level.[1]

[1]If an interrupt is generated in a process where the priority level is 0, the system considers that the process is not prepared to handle the interrupt. The process is then suspended or terminated according to the setting of bit 17 (SC%FRZ) in its capability word.

Table 4-1
Software Interrupt Channel Assignments

Channel	Symbol	Meaning
0-5		Assignable by user program
6	.ICAOV	Arithmetic overflow
7	.ICFOV	Arithmetic floating point overflow
8		Reserved for DEC
9	.ICPOV	Pushdown list (PDL) overflow[1]
10	.ICEOF	End of file condition
11	.ICDAE	Data error file condition[1]
12-14		Reserved for DEC
15	.ICILI	Illegal instruction[1]
16	.ICIRD	Illegal memory read[1]
17	.ICIWR	Illegal memory write[1]
18		Reserved for DEC
19	.ICIFT	Inferior process termination
20	.ICMSE	System resources exhausted[1]
21		Reserved for DEC
22	.ICNXP	Nonexistent page reference
23-35		Assignable by user program

[1]These channels (called panic channels) cannot be completely deactivated. An interrupt generated on one of these channels terminates the process if the channel is not activated.

The software interrupt system processes interrupts on activated channels only, and each channel can be activated and deactivated independently of other channels. When activated, the channel can generate an interrupt for its associated priority level. An interrupt for any priority level is initiated only if there are no interrupts in progress for the same or higher priority levels. If there are, the system remembers the interrupt request and initiates it after all equal or higher priority level interrupts finish. This means that a higher priority level request can suspend a routine processing a lower level interrupt. Thus, the user must be concerned with several items when he assigns his priority levels. He must consider 1) when one interrupt request can suspend the processing of another and 2) when the processing of a second interrupt cannot be deferred until the completion of the first. See Figure 4-2.

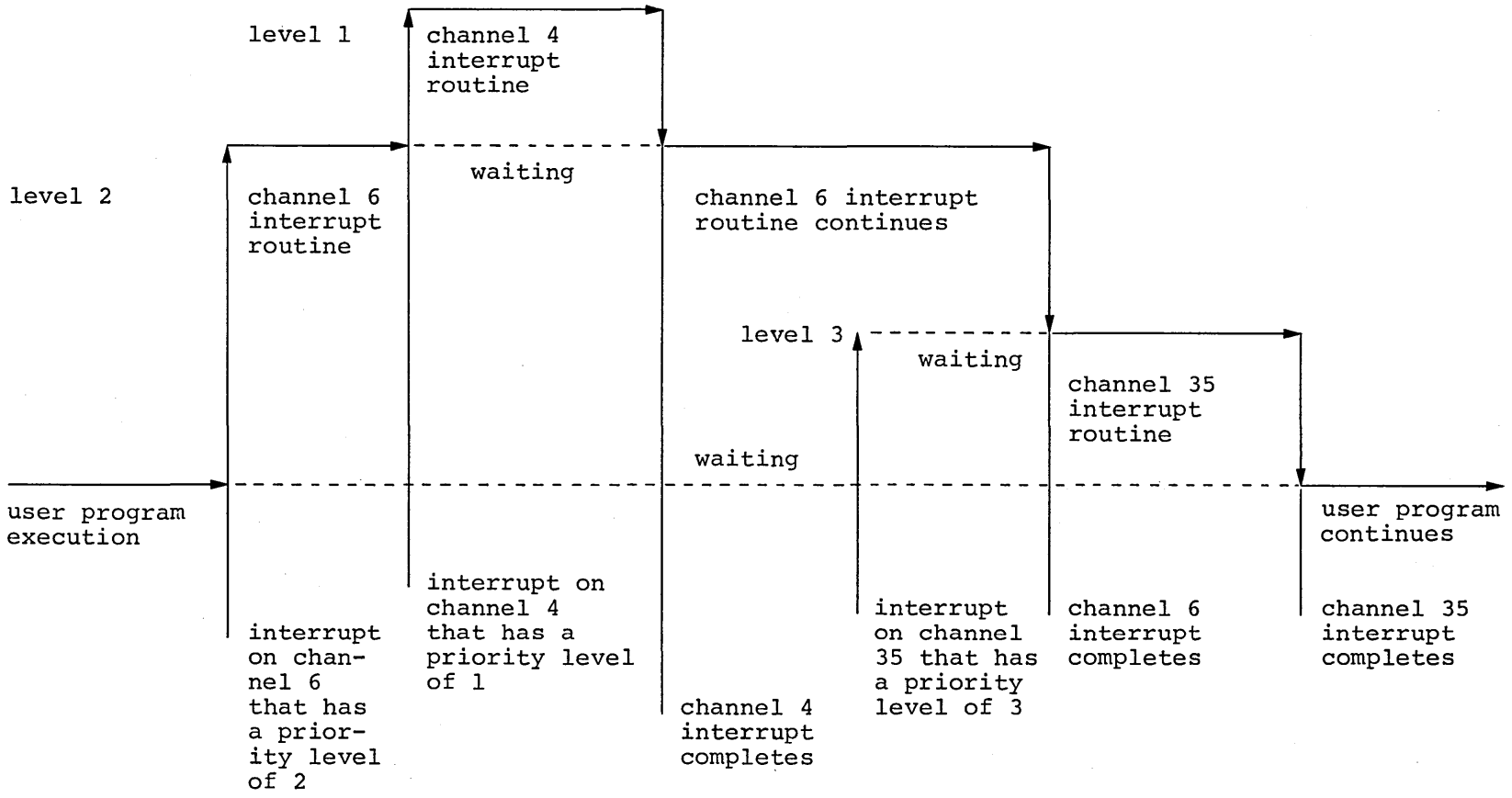


Figure 4-2 Channels and Priority Levels

4.4 SOFTWARE INTERRUPT TABLES

To process interrupts, the user includes, as part of his program, special service routines for the channels he will be using. He must then specify the addresses of these routines to the system by setting up a channel table. In addition, the user must also include a priority level table as part of his program. Finally, he must declare the addresses of these tables to the system.

4.4.1 Channel Table

The channel table, CHNTAB[1], contains a one-word entry for each channel; thus the table has 36 entries. Each entry corresponds to a particular channel, and each channel is associated at any given time with only one interrupt condition. (Refer to Table 4-1 for the interrupt conditions associated with each channel.)

The CHNTAB table is indexed by the channel number (0 through 35). The left half of each entry contains the priority level to which the channel is assigned. The right half of each entry contains the address of the interrupt routine for that channel. If a particular channel is not planned to be used, the corresponding entry in the channel table should be zero.

The following is an example of a channel table.

```
CHNTAB:  2,,CHN0SV       ;channel 0
         2,,CHN1SV       ;channel 1
         2,,CHN2SV       ;channel 2
         2,,CHN3SV       ;channel 3
         0,,0            ;channel 4
         0,,0            ;channel 5
         1,,APRSRV       ;channel 6
         0,,0            ;channel 7
         0,,0            ;channel 8
         1,,STKSRV       ;channel 9
         0,,0            ;channel 10
           .               .
           .               .
           .               .
         0,,0            ;channel 35
```

In this example, channels 0 through 3 are assigned to priority level 2, with the interrupt routine at CHN0SV servicing channel 0, the routine at CHN1SV servicing channel 1, the routine at CHN2SV servicing channel 2, and the routine at CHN3SV servicing channel 3. Channels 6 and 9 are assigned to priority level 1, with the routine at APRSRV servicing channel 6 and the routine at STKSRV servicing channel 9. All remaining channels are not assigned.

[1]The user can call his channel table any name he desires; however, it is a good practice to call the table CHNTAB.

4.4.2 Priority Level Table

The priority level table, LEVTAB[1], contains a one-word entry for each of the three priority levels. The left half of each entry is zero. The right half of each entry contains the address in the user's program where the system will store the flags and program counter (PC) for the associated priority level. The system must save the value of the program counter so that it can return control at the appropriate point in the program once the interrupt routine has completed processing an interrupt. If a particular priority level is not used, its corresponding entry in the level table should be zero.

The following is a sample of a level table.

```
LEVTAB:  0,,PCLEV1       ;Addresses to save PC for interrupts
         0,,PCLEV2       ;occurring on priority levels 1 and 2.
         0,,0            ;No priority level 3 interrupts are
                         ;planned.
```

4.4.3 Specifying The Software Interrupt Tables

Before using the software interrupt system, the user's program must set up the contents of the channel table and the priority level table. The program must then specify their addresses with the SIR monitor call.

The SIR monitor call accepts two words of arguments - the identifier for the program (or process) in AC1 and the table addresses in AC2. Refer to Section 5.3 for the description of process identifiers.

```
MOVEI 1,.FHSLF            ;identifier of current process
MOVE 2,[LEVTAB,,CHNTAB]   ;addresses of the tables
SIR
```

Control always returns to the user's program at the instruction following the SIR call. If the call is successful, the table addresses are stored in the monitor. If the call is not successful, a software interrupt is generated.

Any changes made to the contents of the tables after the SIR call has been executed will be in effect at the time of the next interrupt.

4.5 ENABLING THE SOFTWARE INTERRUPT SYSTEM

Once the interrupt tables have been set up and their addresses defined with the SIR monitor call, the user's program must enable the interrupt system. When the interrupt system is enabled, interrupts occurring on activated channels are processed by the user's interrupt routines. When the interrupt system is disabled, interrupts are processed by the monitor as if the channels for these interrupts were not activated.

---

[1]The user can call his priority level table any name he desires; however, it is good practice to call it LEVTAB.

The EIR monitor call, used to enable the system, accepts one argument- the identifier for the process in AC1.

```
MOVEI 1,.FHSLF          ;identifier of current process
EIR
```

Control always returns to the instruction following the EIR call.

4.6 ACTIVATING INTERRUPT CHANNELS

Once the software interrupt system is enabled, the channels on which interrupts can occur must be activated (refer to Table 4-1 for the channel assignments). The channels to be activated have a nonzero entry in the appropriate word in the channel table.

The AIC monitor call is used to activate one or more of the 36 interrupt channels. This call accepts two words of arguments - the identifier for the process in AC1 and the channels to be activated in AC2. The channels are indicated by setting the appropriate bits (i.e., setting bit n indicates channel n is to be activated). The current state of any channel not specified in the AIC call is not changed.

```
MOVEI 1,.FHSLF                      ;identifier of current process
MOVE 2,[1B<.ICAOV>+1B<.ICPOV>]      ;activate channels 6 and 9
AIC
```

Control always returns to the instruction following the AIC call.

Some channels, called panic channels by convention, cannot be deactivated by disabling the channel or the entire interrupt system (refer to Table 4-1 for these channels). This is because the occurrence of the conditions associated with these channels cannot be completely ignored by the monitor. If one of these conditions occurs, an interrupt is generated whether the channel is activated or not. If the channel is not activated, the process is terminated, and usually a message is output before control returns to the monitor. If the channel is activated, control is given to the user's interrupt routine for that channel.

4.7 PROCESSING AN INTERRUPT

When a software interrupt occurs on a given priority level, the monitor stores the current program counter (PC) word in the address indicated in the priority level table (refer to Section 4.4.2). The monitor then transfers control to the interrupt routine associated with the channel on which the interrupt occurred. The address of this routine is specified in the channel table (refer to Section 4.4.1).

Since the user's program cannot determine when an interrupt will occur, the interrupt routine is responsible for preserving the state of the program so that the program can be resumed properly. Thus, the first action taken by the routine is to store the contents of any user accumulators that will be used during the processing of the interrupt. After the accumulators are saved, the interrupt routine processes the interrupt.

Occasionally, an interrupt routine may need to alter locations in the main section of the program. For example, a routine may change the stored PC word to resume execution at a location different from where

the interrupt occurred. Or it may alter a value that caused the interrupt. It is important that care be used when writing routines that alter data because any changes will remain when control is returned to the main program. For example, if data is inadvertently stored in the PC word, return to the main section of the program would be incorrect when the system attempted to use the word as the value of the program counter.

If a higher priority interrupt occurs during the execution of an interrupt routine, the executing routine is suspended. The value of its program counter is stored at the location indicated in the priority level table for the new interrupt. When the routine for this new interrupt is completed, the suspended routine is resumed. If an interrupt of the same or lower priority occurs during the execution of a routine, the monitor holds the interrupt until all higher or equal level interrupts have been processed.

The system considers the user's program unable to process an interrupt on an activated channel if:

1. The priority level associated with the channel is 0.

2. The program has not defined its interrupt tables by executing a SIR monitor call.

3. The process has not enabled the interrupt system by executing an EIR monitor call, and the channel on which the interrupt occurs is a panic channel.

In any of the above cases, the occurrence of an interrupt terminates the user's program.

4.7.1 Dismissing An Interrupt

Once the processing of an interrupt is complete, the interrupt routine should restore the user accumulators to their initial values. Then it returns control to the interrupted code via the DEBRK monitor call. This call restores the PC word and resumes the program. The call has no arguments and must be the last statement in the interrupt routine.

The user's program is restored to its state prior to the interrupt if the stored PC word has not been changed. For example, if the program was interrupted while waiting for I/O to complete, it is restored to that state after execution of the DEBRK call. If the PC word was changed, the program resumes execution at the new PC location.

4.8 TERMINAL INTERRUPTS

The user's program can associate channels 0 through 5 and channels 24 through 35 with occurrences of various conditions, such as the occurrence of a particular character typed at the terminal or the receipt of an IPCF message. This section discusses terminal interrupts; refer to Chapters 6 and 7 for other types of assignable interrupts.

There are 36 codes used to specify terminal characters or conditions on which interrupts can be initiated. These codes, along with their associated conditions, are shown in Table 4-2.

Table 4-2
Terminal Codes and Conditions

Code	Symbol	Character or Condition
0	.TICBK	CTRL/@ or break
1	.TICCA	CTRL/A
2	.TICCB	CTRL/B
3	.TICCC	CTRL/C
4	.TICCD	CTRL/D
5	.TICCE	CTRL/E
6	.TICCF	CTRL/F
7	.TICCG	CTRL/G
8	.TICCH	CTRL/H
9	.TICCI	CTRL/I
10	.TICCJ	CTRL/J
11	.TICCK	CTRL/K
12	.TICCL	CTRL/L
13	.TICCM	CTRL/M
14	.TICCN	CTRL/N
15	.TICCO	CTRL/O
16	.TICCP	CTRL/P
17	.TICCQ	CTRL/Q
18	.TICCR	CTRL/R
19	.TICCS	CTRL/S
20	.TICCT	CTRL/T
21	.TICCU	CTRL/U
22	.TICCV	CTRL/V
23	.TICCW	CTRL/W
24	.TICCX	CTRL/X
25	.TICCY	CTRL/Y
26	.TICCZ	CTRL/Z
27	.TICES	ESC key
28	.TICRB	Delete (or rubout) key

Table 4-2 (Cont.)
Terminal Codes and Conditions

Code	Symbol	Character or Condition
29	.TICSP	Space
30	.TICRF	Dataset carrier off
31	.TICTI	Typein
32	.TICTO	Typeout
33-35		Reserved

To cause terminal interrupts to be generated, the user's program must assign the desired terminal code to one of the assignable channels. The ATI monitor call is used to assign this code. This call accepts one word of arguments - the terminal code in the left half of AC1 and the channel number in the right half.

```
MOVE 1,[.TICCE,,INTCH1]  ;assign CTRL/E to channel INTCH1
ATI
```

Control always returns to the instruction following the ATI call. If the current job is not attached to a terminal (i.e., there is no terminal controlling the job), the terminal code assignments are remembered; they will be in effect when a terminal is attached.

The monitor handles the receipt of a terminal interrupt character in either immediate mode or deferred mode. In immediate mode, the terminal character causes the system to initiate an interrupt as soon as the user types the character (i.e., as soon as the system receives it). In deferred mode, the terminal character is placed in the input stream in sequence with other characters of the input, unless two of the same character are typed in succession. In this case, an interrupt occurs at the time the second one is typed. If only one character enabled in deferred mode is typed, the system initiates an interrupt only when the program attempts to read the character. Deferred mode allows interrupt actions to occur in sequence with other actions specified in the input (e.g., when characters are typed ahead of the time that the program actually requests them). In either mode, the character is not passed to the program as data. The system assumes that interrupts are to be handled immediately unless a program has issued the STIW (Set Terminal Interrupt Word) monitor call. (Refer to DECsystem-20 Monitor Calls Reference Manual for a description of this call.)

4.9 ADDITIONAL SOFTWARE INTERRUPT MONITOR CALLS

Additional monitor calls are available that allow the user's program to check and to clear various parts of the software interrupt system. Also, there is a call useful for multiple process communication (refer to the IIC call in Section 5.10).

4.9.1 SKPIR Monitor Call

The SKPIR monitor call is used to test the software interrupt system to see if it is currently enabled. The call accepts in AC1 the identifier of the process. After execution of the call, control returns to the next instruction if the system is off or returns to the second instruction if the system is on.

```
MOVEI 1,.FHSLF          ;identifier of current process
SKPIR                   ;test interrupt system
 return                 ;system is off
return                  ;system is on
```

4.9.2 RIR Monitor Call

The RIR monitor call is used to read the channel and priority level table addresses for the process as set by the SIR monitor call. This call is useful when several processes in one job want to share the interrupt tables. The call accepts in AC1 the identifier of the process and returns in AC2 the table addresses. The left half of AC2 contains the priority level table address and the right half contains the channel table address. If the SIR monitor call has not been executed by the process, AC2 contains zero.

```
MOVEI 1,.FHSLF          ;identifier of current process
RIR                     ;return the table addresses
```

Control always returns to the instruction following the RIR call.

4.9.3 DIR Monitor Call

The DIR monitor call is used to disable the software interrupt system for the process. It accepts in AC1 the identifier of the process.

```
MOVEI 1,.FHSLF          ;identifier of current process
DIR                     ;disable system
```

Control always returns to the instruction following the DIR call.

If interrupts occur while the interrupt system is disabled, they are remembered until the system is reenabled. At that time, the interrupts take effect unless an intervening CIS monitor call (refer to Section 4.9.6) has been issued. Software interrupts assigned to panic channels are not completely disabled by the DIR call. These interrupts terminate the process, and the superior process is notified if it has enabled channel .ICIFT. In addition, if the terminal code for CTRL/C (.TICCC) is assigned to a channel, it still causes an interrupt that cannot be disabled by the DIR call. However, the CTRL/C interrupt can be disabled by deactivating the channel assigned to the CTRL/C terminal code.

4.9.4 DIC Monitor Call

The DIC monitor call is used to deactivate one or more of the 36 interrupt channels. The call accepts two words of arguments - the identifier for the process in AC1 and the channels to be deactivated in AC2. The channels are indicated by setting the appropriate bits (i.e., setting bit n indicates channel n is to be deactivated).

```
MOVEI 1,.FHSLF                      ;identifier of current process
MOVE 2,[1B<.ICAOV>+1B<.ICPOV>]      ;deactivate channels 6 and 9
DIC
```

Control always returns to the instruction following the DIC call.

When a channel is deactivated, interrupt requests for that channel are ignored except for interrupts generated on panic channels (refer to Section 4.6).

4.9.5 DTI Monitor Call

The DTI monitor call is used to deassign a terminal code from a particular channel. This call accepts one argument word - the terminal code in the left half of AC1 and the channel number in the right half.

```
MOVE 1,[.TICCE,,INTCH1]  ;deassign CTRL/E from channel INTCH1
DTI
```

Control always returns to the instruction following the DTI call. This monitor call is ignored if the specified terminal code has not been defined by the current job.

4.9.6 CIS Monitor Call

The CIS monitor call is used to clear the interrupt system for the current process. This causes interrupts in progress and all waiting interrupts to be cleared. This call has no arguments, and control always returns to the instruction following the CIS call. The RESET monitor call (refer to Section 2.6.1) performs these same actions as part of its initializing procedures.

4.10 SUMMARY

To use the software interrupt system, the user's program must:

1. Supply routines that will process the interrupts.
2. Set up a channel table containing the addresses of the routines (refer to Section 4.4.1) and a priority level table containing the addresses for storing the program counter (PC) values (refer to Section 4.4.2).
3. Specify the addresses of the tables with the SIR monitor call (refer to Section 4.4.3).
4. Enable the software interrupt system with the EIR monitor call (refer to Section 4.5).

5. Activate the desired channels with the AIC monitor call (refer to Section 4.6).

4.11 SOFTWARE INTERRUPT EXAMPLE

This program copies one file to another. It accepts the input and output filenames from the user. The end of file is detected by a software interrupt, and CTRL/E is enabled as an escape character.

```
                TITLE SOFTWARE INTERRUPT EXAMPLE
                SEARCH MONSYM
                T1=1
                T2=2
                INTCH1=1

START:  RESET                                   ;RELEASE FILES, ETC.
        MOVEI T1,.FHSLF                         ;CURRENT PROCESS
        MOVE T2,[LEVTAB,,CHNTAB]                ;INTERRUPT TABLES
        SIR
        EIR                                     ;ENABLE SYSTEM
        MOVE T2,[1B<INTCH1>+1B<.ICEOF>]         ;ACTIVATE CHANNELS
        AIC
        MOVE T1,[.TICCE,,INTCH1]                ;ASSIGN CTRL/E TO CHANNEL 1
        ATI
GETIF:  HRROI T1,[ASCIZ/INPUT FILE: /]
        PSOUT                                   ;PROMPT USER FOR INPUT NAME
        MOVSI T1,(GJ%OLD+GJ%MSG+GJ%CFM+GJ%FNS+GJ%SHT)
        MOVE T2,[.PRIIN,,.PRIOU]
        GTJFN                                   ;GET FILENAME FROM USER
         ERJMP ERROR1
        MOVEM T1,INJFN
GETOF:  HRROI T1,[ASCIZ/OUTPUT FILE: /]
        PSOUT                                   ;PROMPT USER FOR OUTPUT NAME
        MOVSI T1,(GJ%FOU+GJ%MSG+GJ%CFM+GJ%FNS+GJ%SHT)
        MOVE T2,[.PRIIN,,.PRIOU]
        GTJFN                                   ;GET FILENAME FROM USER
         ERJMP ERROR2
        MOVEM T1,OUTJFN
OPNIF:  MOVE T1,INJFN
        MOVE T2,[7B5+OF%RD]
        OPENF                                   ;OPEN INPUT FILE
         ERJMP ERROR3
OPNOF:  MOVE T1,OUTJFN
        MOVE T2,[7B5+OF%WR]
        OPENF                                   ;OPEN OUTPUT FILE
         ERJMP ERROR3
CPYBYT: MOVE T1,INJFN                           ;READ INPUT BYTE
        BIN
        MOVE T1,OUTJFN                          ;WRITE OUTPUT BYTE
        BOUT
        JRST CPYBYT                             ;LOOP UNTIL EOF
DONE:   MOVE T1,INJFN
        CLOSF                                   ;CLOSE INPUT FILE
          JFCL
        MOVE T1,OUTJFN
        CLOSF                                   ;CLOSE OUTPUT FILE
          JFCL
        HALTF
```

```
;ROUTINE TO HANDLE ^E - ABORTS OPERATION

CTRLE:  MOVEI T1,.PRIOU
        CFOBF                                   ;CLEAR OUTPUT BUFFER
        HRROI T1,[ASCIZ/ABORTED./]
        PSOUT                                   ;INFORM USER
        CIS                                     ;CLEAR SYSTEM
        JRST START

;ROUTINE TO HANDLE EOF - COMPLETES OPERATION NORMALLY

EOFINT: MOVEM T1,INTAC1                         ;SAVE AC'S
        MOVEI T1,DONE                           ;CHANGE PC
        MOVEM T1,PC2                            ;TO DONE
        MOVE T1,INTAC1                          ;RESTORE AC'S
        DEBRK                                   ;DISMISS INTERRUPT

;LEVEL TABLE
LEVTAB: 0
        PC2
        0
PC2:    BLOCK 1
;CHANNEL TABLE
CHNTAB: 0                                       ;UNUSED CHANNELS HAVE 0
        2,,CTRLE                                ;CHANNEL 1 IS CTRL/E
        REPEAT ^D8,<0>                          ;CHANNEL 2-9 NOT USED
        2,,EOFINT                               ;CHANNEL 10 IS EOF
        REPEAT ^D25,<0>                         ;CHANNEL 11-35 NOT USED
INJFN;  BLOCK 1
OUTJFN: BLOCK 1
INTAC1: BLOCK 1
ERROR1: TMSG <
?INVALID FILE SPECIFICATION>
        HALTF
ERROR2: TMSG <
?INVALID FILE SPECIFICATION>
        HALTF
ERROR3: TMSG <
?CANNOT OPEN FILE>
        HALTF
        LIT
        END     START
```

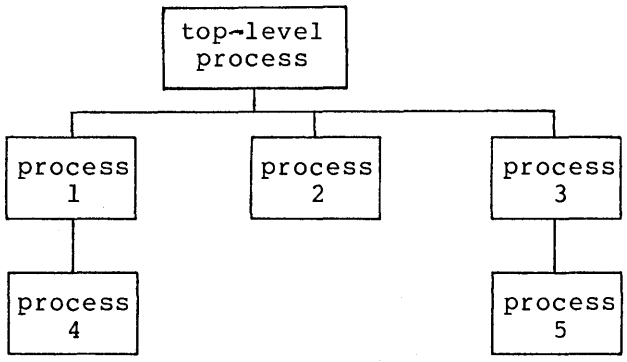
CHAPTER 5

PROCESS STRUCTURE

As stated in Chapter 1, the DECsystem-20 operating system allows each job to have multiple, simultaneously-runnable processes. Each process has its own environment called its address space. Associated with the environment is the program counter (PC) of the process and a well-defined relationship with other processes in the job.

The DECsystem-20 operating system schedules the running of processes, not entire jobs. A process can be scheduled independent of other processes because it has a definite existence: its beginning is the time at which it is created, and its end is the time at which it is killed. At any point in its existence, a process can be described by its state, which is represented by a status word and a PC word (refer to Section 5.9).

The relationships among processes in a job are shown in the diagram below. Each process has one immediate superior process (except for the top-level process) and can have one or more inferior processes. Two processes are parallel if they have the same immediate superior. A process can create an inferior process but not a parallel or superior process.



Process 1 is the superior process of process 4, and process 3 is the superior of process 5. Processes 4 and 5 are the inferiors of processes 1 and 3, respectively. Process 2 has no inferior process. Processes 1, 2 and 3 are parallel because they have the same superior process (i.e., the top-level process). Processes 4 and 5, although at the same depth in the structure, are not parallel because they do not have the same superior process. Process 1 created process 4 but could not have created any other process shown in the structure above.

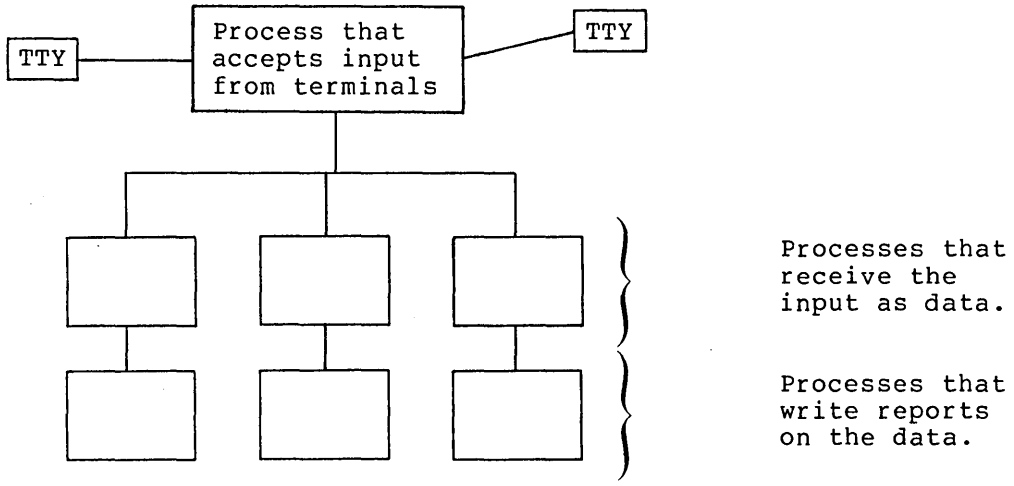
5.1 USES FOR MULTIPLE PROCESSES

A multiple-process job structure allows:

1. One job to have more than one program runnable at the same time. These programs can be independent programs, each one compiled, debugged, and loaded separately. Each program can then be placed in a separate process. These processes can be parallel to each other, but are inferior to the main process that created them. This use allows parallel processing of the individual programs.

2. One process to wait for an event to occur (e.g., the completion of an I/O operation) while another process continues its computations. Communication between the two processes is such that when the event occurs, the process that is computing can be notified via the software interrupt system. This use allows two processes within a job to overlap I/O with computations.

One application of a multiple-process job structure is the following situation: a superior process is responsible for accepting input from various terminals. After receiving this input, the process sends it to various inferior processes as data. These inferior processes can then initiate other processes, for example, to write reports on the data that was received.



Another application is that used for the user interface on the DECsystem-20. On the DECsystem-20, the top-level process in the job structure is the Command Language. This process services the user at the terminal by accepting input. When the user runs a program (e.g., MACRO, FORTRAN), the Command Language process creates an inferior process, places the requested program in it, and executes it. The Command Language can then wait for an event to occur, either from the program or from the user. An event from the program can be its completion, and an event from the user can be his typing of a certain terminal key (e.g., CTRL/C).

5.2 PROCESS COMMUNICATION

A process can communicate with other processes in the system in several ways:

direct process control

software interrupts

IPCF and ENQ/DEQ facilities

memory sharing

5.2.1 Direct Process Control

A process can create and control other processes inferior to it within the job structure. The superior process can cause the inferior process to begin execution and then to suspend and later resume execution. After the inferior process has completed its tasks, the superior process can delete the inferior from the job structure.

Some of the monitor calls used for direct process control are: CFORK, to create a process; SFORK, to start a process; WFORK, to wait for a process to terminate; RFSTS, to obtain the status of a process; and KFORK, to delete a process. Refer to the DECsystem-20 Monitor Calls Reference Manual for descriptions of additional monitor calls dealing with process control.

5.2.2 Software Interrupts

The software interrupt facility enables a process to receive asynchronous signals from other processes, the system, or the terminal user or to receive signals as a result of its own execution. For example, a superior process can enable the interrupt system so that it receives an interrupt when one of its inferiors terminates. In addition, processes within a job structure can explicitly generate interrupts to each other for communication purposes.

Some of the monitor calls used when communication occurs via the software interrupt system are: SIR, to specify the interrupt tables; EIR, to enable the interrupt system; AIC, to activate the interrupt channels; and IIC, to initiate an interrupt on a channel. Refer to Chapter 4 and Section 5.10 for more information.

5.2.3 IPCF And ENQ/DEQ Facilities

The Inter-Process Communication Facility (IPCF) enables processes and jobs to communicate by sending and receiving informational messages. The MSEND call is used to send a message, the MRECV call is used to receive a message, and the MUTIL call is used to perform utility functions. Refer to Chapter 7 for descriptions of these calls.

The ENQ/DEQ facility allows cooperating processes to share resources and facilitates dynamic resource allocation. The ENQ call is used to obtain a resource, the DEQ call is used to release a resource, and the ENQC call is used to obtain status about a resource. Refer to Chapter 6 for descriptions of these calls.

5.2.4 Memory Sharing

Each page in a process' address space is either private to the process or shared with other processes. Pages are shared among processes when the same page is represented in more than one process address space. This means that two or more processes can identify and use the same page of physical storage. Even when several processes have identified the same page, each process can have a different access to that page, such as access to read or write that page.

A type of page access that facilitates sharing is the copy-on-write access. A page with this access remains shared as long as all processes read the page. As soon as a process writes to the page, the system makes a private copy of the page for the process doing the writing. Other processes continue to read and execute the original page. This access provides the capability of sharing as much as possible but still allows the process to change its data without changing the data of other processes. A monitor call used when sharing memory is PMAP. Refer to Section 5.6.2 for more information.

5.3 PROCESS IDENTIFIERS

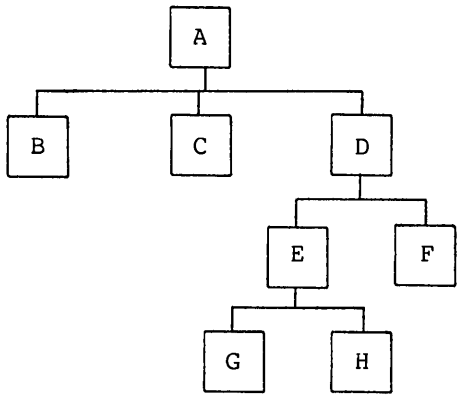
In order for processes to communicate with each other, a process must have an identifier, or handle, for referencing another process. When a process creates an inferior process, it is given a handle on that inferior. This handle is a number in the range 400001 to 400777 and is meaningful only to the process to which it is given (i.e., to the superior process). For example, if process A creates process B, process A is given a handle (e.g., 400003) on process B. Process A then specifies this handle when it uses monitor calls that refer to process B. However, process B is not known by this handle to any other process in the structure, including itself. The handle 400003 may in fact be known to process B, but it would describe a process inferior to process B.

There are several standard process handles that are never assigned by the system but have a specific meaning when used by any process in the structure. These handles are used when a process needs to communicate with a process other than its immediate inferior or with multiple processes at once. These handles are described in Table 5-1.

Table 5-1
Process Handles

Number	Symbol	Meaning
400000	.FHSLF	The current process (or self).
-1	.FHSUP	The immediate superior of the current process.
-2	.FHTOP	The top-level process in the job structure.
-3	.FHSAI	The current process and all of its inferiors.
-4	.FHINF	All of the inferiors of the current process.
-5	.FHJOB	All processes in the job structure.

Consider the job structure below.



The following indicates the specific process or processes being referenced if process E gives the handle:

.FHSLF	refers to process E
.FHSUP	refers to process D
.FHTOP	refers to process A
.FHSAI	refers to processes E, G, and H
.FHINF	refers to processes G and H
.FHJOB	refers to processes A through H

The process must have the appropriate capability enabled in its capability word to use the handles .FHSUP and .FHTOP (refer to Section 5.5.1).

Process E can reference one of its inferiors (e.g., G) with the handle it was given when it created the inferior. Process E can reference other processes in the structure (e.g., F) by executing the GFRKS monitor call to obtain a handle on the desired process. Refer to the DECsystem-20 Monitor Calls Reference Manual for a description of the GFRKS call.

5.4 OVERVIEW OF MONITOR CALLS FOR PROCESSES

Monitor calls exist for creating, loading, starting, suspending, resuming, interrupting, and deleting processes. When a process is created, its address space is assigned, and the process is added to the job structure of the creating process. The contents of its address space can be specified at the time the process is created or at a later time. The process can also be started at the time it is created. A process remains potentially runnable until it is explicitly deleted or its superior is deleted.

A process may be suspended if one of the following conditions occurs:

1. The process executes an instruction that causes a software interrupt to occur, and it is not prepared to process the interrupt.

2. The process executes the HALTF monitor call.

3. The superior process requests suspension of its inferior.

4. The superior process is suspended. When a process is suspended, all of its inferior processes are also suspended.

5.5 CREATING A PROCESS

A process creates an inferior process by executing the CFORK (Create Process) monitor call. (The term fork is synonymous with the term process.) This monitor call can also be used to specify the address space, capabilities, ACs, and PC for the inferior process and to start the execution of the inferior.

The CFORK call accepts two words of arguments in AC1 and AC2.

AC1: characteristics for the inferior in the left half, and PC address for the inferior in the right half.

AC2: address of a 20(octal) word block containing the AC values for the inferior.

The characteristics for the inferior process are defined by the following bits:

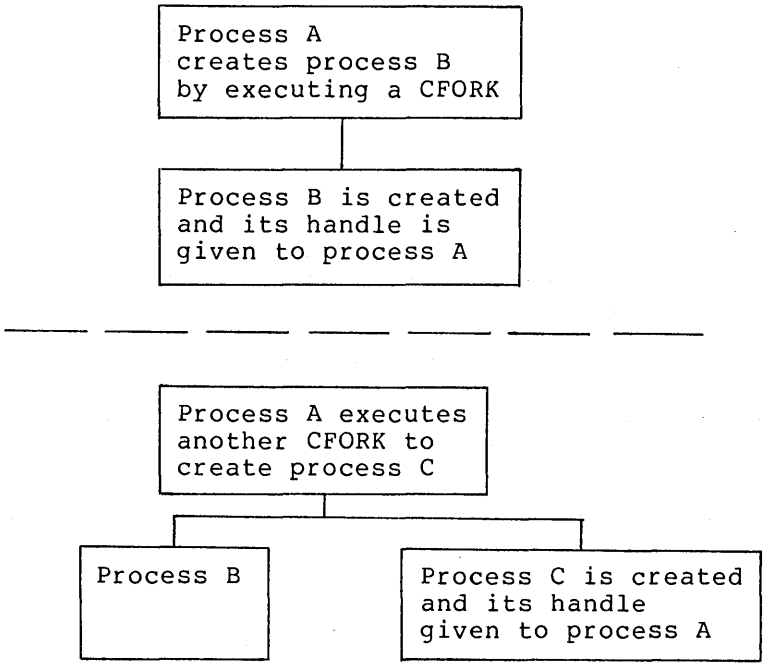
Bit	Symbol	Meaning
0	CR%MAP	Set the map of the inferior process to the same as the map of the superior (i.e., creating) process. This means that the superior and the inferior will share the same address space. Changes made by one process will be seen by the other process. If this bit is not on in the call, the inferior's map will contain all zeros.
1	CR%CAP	Set the capability word of the inferior process to the same as the capability word of the superior process. (Refer to Section 5.5.1 for the description of the capability word.) If this bit is not on in the call, the inferior will have no special capabilities.
2		Reserved for DEC (must be zero).
3	CR%ACS	Set the ACs of the inferior process to the values beginning at the address given in AC2. If this bit is not on in the call, the inferior's ACs will be set to zero, and the contents of AC2 is ignored.
4	CR%ST	Set the PC for the inferior process to the address given in the right half of AC1 and start execution of the inferior.

If this bit is not on in the call, the right half of AC1 is ignored, and the inferior is not started.

If execution of the CFORK call is not successful, the inferior process is not created and an error code is returned in AC1. The execution of the program in the superior process continues at the instruction following the CFORK call.

If execution of the CFORK call is successful, the inferior process is created and its process handle is returned in the right half of AC1. This handle is then used by the superior process when communicating with its inferior process. The execution of the program in the superior process continues at the second instruction following the CFORK call.

Assume that process A executes the CFORK monitor call twice to create two parallel inferior processes. This is represented pictorially below.



Note that process A has been given two handles, one for process B and one for process C. Process A can refer to either of its inferiors by giving the appropriate handle or to both of its inferiors by giving a handle of -4.

5.5.1 Process Capabilities

When a new process is created, it is given the same capabilities as its superior, or it is given no special capabilities. This is indicated by the setting of the CR%CAP bit in the CFORK call. The capabilities for a process are indicated by two capability words. The first word indicates if the capability is available to the process, and the second word indicates if the capability is enabled for the process. This second word is the one being set by the CR%CAP bit in the CFORK call.

Types of capabilities represented in the capability words are job, process, and user capabilities. Each capability corresponds to a particular bit in the capability words and thus can be activated and protected independently of the other capabilities. Refer to the DECsystem-20 Monitor Calls Reference Manual for more information on the capability words.

5.6 SPECIFYING THE CONTENTS OF THE ADDRESS SPACE OF A PROCESS

Once a process is created, the contents of its address space can be specified. This can be accomplished by one of three ways. As mentioned in Section 5.5, bit CR%MAP can be set in the CFORK call to indicate that the address space of the inferior process is to be the same as the address space of the creating process. In addition, the creating process can execute the GET monitor call to map specified pages from a file into the address space of the inferior process. Finally, the creating process can execute the PMAP monitor call to map specified pages from another process into the address space of the inferior process.

If the creating process does not specify the contents of the inferior's address space, the address space will be filled with zeros.

5.6.1 GET Monitor Call

The GET monitor call is used to map pages from a file into the address space of the specified process. The file must be a saved file that was created with either the SAVE or SSAVE monitor call (refer to the DECsystem-20 Monitor Calls Reference Manual).

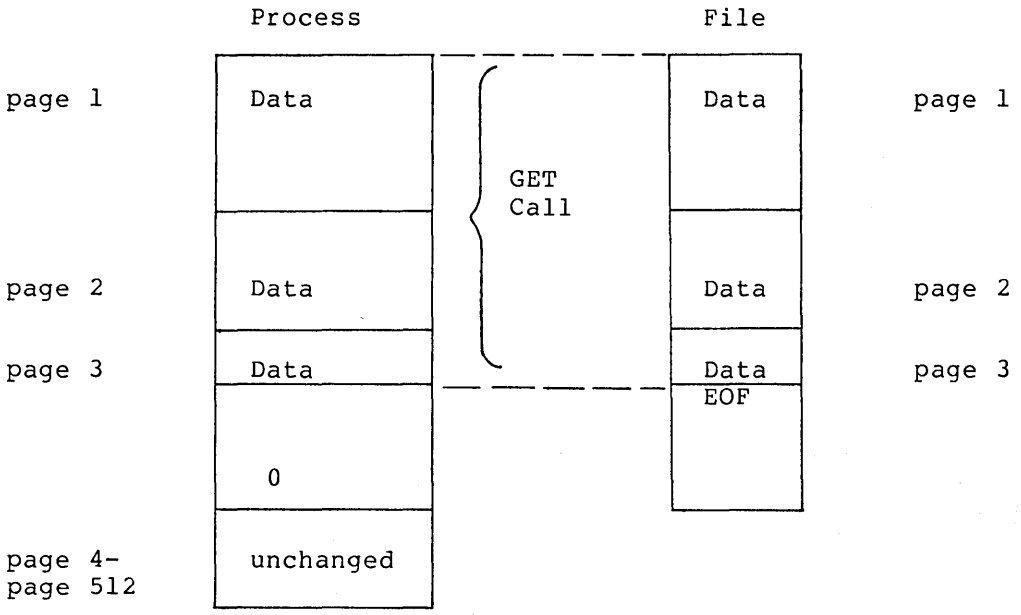
The GET monitor call accepts two words of arguments in AC1 and AC2. The first word specifies the handle of the desired process and the JFN of the desired file. The second word specifies where the pages from the file are to be placed in the address space of the process. Thus,

AC1: process handle in the left half, and JFN in the right half. If GT%ADR (bit 19) is on, AC2 is used for the memory limits. If GT%ADR is not on, all existing pages in the file are mapped into the process.

AC2: number of lowest page in the left half and number of highest page in the right half. These page numbers are for the address space of the process and are used to control the portions of memory that are loaded. These values are specified only if GT%ADR is on in AC1.

When the pages of the file are mapped into pages in the process' address space, the previous contents of the process pages are overwritten. Any full pages in the process that are not overwritten are unchanged. Any portions of process pages for which there is no data in the file are filled with zeros.

For example, a GET call executed for a file that contains 2 1/2 pages sets up the process' address space as shown in the following diagram.



After execution of the GET call, control returns to the user's program at the instruction following the call. If an error occurs, a software interrupt is generated, which the program can process via the software interrupt system.

5.6.2 PMAP Monitor Call

The PMAP monitor call is used to map pages from one process to the address space of a second process. Data is not actually transferred; only the contents of the page map of the second (i.e., destination) process are changed.

The PMAP monitor call accepts three words of arguments in AC1 through AC3. The first word contains the handle and page number of the first page to be mapped in the source process (i.e., the process whose pages are being mapped). The second word contains the handle and page number of the first page to be mapped in the destination process (i.e.,the process into which the pages are being mapped). The third word contains a count of the number of pages to map and bits indicating the access that the destination process will have to the pages mapped. Thus,

AC1: source process handle in the left half, and page number in the process in the right half.

AC2: destination process handle in the left half, and page number in the process in the right half.

AC3: count of number of pages to map and the access bits.

The count and access bits that can be specified in AC3 are described below.

Bit	Symbol	Meaning
0	PM%CNT	Repeat the mapping operation the number of times specified by the right half of AC3. The page numbers in both

		processes are incremented by 1 each time the operation is performed.
2	PM%RD	Allow read access to the page.
3	PM%WR	Allow write access to the page.
9	PM%CPY	Allow copy-on-write access to the page.
18-35		The number of times to repeat the mapping operation if bit 0 (PM%CNT) is set.

Upon successful execution of the PMAP call, addresses in the destination process actually refer to addresses in the source process. The contents of the destination page previous to the execution of the call have been deleted. The access requested in the PMAP call is granted if it does not conflict with the current access of the destination page (i.e., an AND operation is performed between the specified access and the current access). Control returns to the user's program at the instruction following the PMAP call. If an error occurs, a software interrupt is generated, which the program can process via the software interrupt system.

5.7 STARTING AN INFERIOR PROCESS

A program in an inferior process can be started in one of two ways. As mentioned in Section 5.5, the superior process can specify in the CFORK call the PC for the inferior process and start its execution. Alternatively, the superior process, after executing the CFORK call to create an inferior process, can execute the SFORK (Start Process) monitor call to start it.

The SFORK monitor call accepts two words of arguments in AC1 and AC2. The first word contains the handle of the desired process. The address of the PC word at which the process is to be started is in the second word. Thus,

AC1: process handle

AC2: address of inferior's PC

The process handle given in AC1 cannot refer to a superior process, to more than one process (e.g., .FHINF), or to a process that has already been started.

After execution of the SFORK call, control returns to the user's program at the instruction following the call. If an error occurs, a software interrupt is generated, which the program can process via the software interrupt system.

5.8 INFERIOR PROCESS TERMINATION

The superior process has one of two ways in which it can be notified when its inferiors terminate execution: via the software interrupt system or by executing the WFORK monitor call. An inferior process will terminate normally when it executes a HALTF monitor call.

Alternatively, the process will terminate abnormally when it executes an instruction that generates a software interrupt, such as an illegal instruction, and it has not activated the appropriate channel.

By activating channel .ICIFT (channel 19) for inferior process termination and enabling the software interrupt system, the superior process will receive an interrupt when one of its inferiors terminates. (Refer to Section 4.6 for information on activating channel .ICIFT.) The interrupt occurs when the first process terminates. Use of the interrupt system allows the superior to do other processing until an interrupt occurs, indicating that an inferior process has terminated.

In some cases, however, the superior cannot do additional processing until either a specific process or all of its inferior processes have completed execution. If this is the case, the superior process can execute the WFORK (Wait Process) monitor call. This call blocks the superior until one or all of its inferiors have terminated.

The WFORK monitor call accepts one argument in AC1, the handle of the desired process. This handle can be .FHINF (-4) to block the superior until all inferiors terminate, but cannot be a handle on a superior process.

After execution of the WFORK monitor call, control returns to the user's program at the instruction following the call, when the specified process or all of the inferior processes terminate. If an error occurs, it generates a software interrupt, which the program can process via the software interrupt system.

5.9 INFERIOR PROCESS STATUS

The superior process can obtain the status of one of its inferiors by executing the RFSTS (Read Process Status) monitor call. This call returns the status and PC words of the given inferior process.

The RFSTS monitor call accepts one argument in AC1, the handle of the desired process. This handle cannot refer to a superior process or to more than one process.

After execution of the RFSTS call, control returns to the user's program at the instruction following the call. If the RFSTS call is successful, AC1 contains the status word of the given process and AC2 contains the PC word. The status word is shown in Table 5-2.

Table 5-2
Process Status Word

Bit	Symbol	Meaning
0	RF%FRZ	The process is suspended (i.e., frozen). If this bit is not on, the process is not suspended.
1-17	RF%STS	The status of the process. Value Symbol Meaning 0 .RFRUN The process is runnable.

Table 5-2 (Cont.)
Process Status Word

Bit	Symbol	Meaning		
		1	.RFIO	The process is halted waiting for I/O
		2	.RFVPT	The process is halted by a HFORK or HALTF monitor call or was never started.
		3	.RFFPT	The process is halted by the occurrence of a software interrupt for which it was not prepared to handle. The right half of the status word contains the number of the channel on which the interrupt occurred.
		4	.RFWAT	The process is halted waiting for another process to terminate.
		5	.RFTIM	The process is halted for a specified amount of time.
18-35	RF%SIC	The channel number on which an interrupt occurred, which the process was not prepared to handle (see process status code .RFFPT above).		

If an error occurs during execution of the RFSTS call, a software interrupt is generated, which the program can process via the software interrupt system.

5.10 PROCESS COMMUNICATION

A superior process can communicate with its inferiors by sharing the same pages of memory. This sharing is accomplished with the CFORK (bit CR%MAP) or the PMAP monitor call. When the superior executes either of these calls, both the superior and the inferior share the same pages. Changes made to the shared pages by either process will be seen by the other process.

Alternatively, processes can communicate via the software interrupt system. The superior process can cause a software interrupt to be generated in an inferior process by executing the IIC (Initiate Interrupt on Channel) monitor call. For this type of communication to occur, the inferior's interrupt channels must be activated and its interrupt system enabled.

The IIC monitor call accepts two words of arguments in AC1 and AC2. The handle of the process to receive the interrupt is given in the right half of AC1. AC2 contains a 36-bit word, with each bit representing one of the 36 software channels. If a bit is on in AC2, a software interrupt is initiated on the corresponding channel. For example, if bit 5 is on in AC2, an interrupt is initiated on channel 5. Thus,

AC1: process handle in the right half

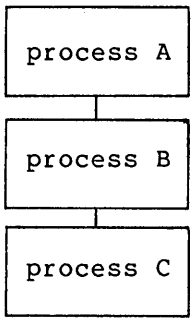
AC2: 36-bit word, with bit n on to initiate a software interrupt on channel n

The process handle given cannot refer to a superior process or to more than one process.

After execution of the IIC call, control returns to the user's program at the instruction following the call. If an error occurs, it generates a software interrupt, which the program can process via the software interrupt system.

5.11 DELETING AN INFERIOR PROCESS

A process is deleted from the job structure when the superior process executes the KFORK (Kill Process) monitor call. When a process is deleted, its address space, its handle, and any JFNs acquired by the process are released. If the process being deleted has processes inferior to it, the inferiors are also deleted. For example, in the structure:



if process A deletes process B by executing a KFORK call, process C is also deleted.

The KFORK monitor call accepts one argument in the right half of AC1, the handle of the process to be deleted. This handle cannot refer to a superior process, to more than one process (e.g., .FHINF), or to the process executing the call (i.e., .FHSLF). The RESET monitor call is used to reinitialize the current process; refer to Section 2.6.1.

After execution of the KFORK call, control returns to the user's program at the instruction following the call. If an error occurs, a software interrupt is generated, which the program can process via the software interrupt system.

5.12 PROCESS EXAMPLES

Example 1 - This program creates an inferior process to provide timing interrupts.

```
TITLE TIMINT - EXAMPLE OF USING AN INFERIOR PROCESS TO PROVIDE TIMING INTERRUPTS

        SEARCH MONSYM, MACSYM
        .REQUIRE SYS:MACREL

        T1=1
        T2=2
        T3=3
        T4=4
        P=17

START:  RESET                       ;RELEASE FILES, ETC.
        MOVE P,[IOWD 50,PDL]        ;INITIALIZE PUSH-DOWN LIST IN CASE OF ERRORS
        MOVX T1,CR%MAP              ;MAKE NEW PROCESS SHARE THIS PROCESS'S MEMORY
        CFORK                       ;CREATE A NEW PROCESS
         JSHLT                      ;UNEXPECTED ERROR.
        MOVEM T1,HANDLE             ;SAVE PROCESS HANDLE

; HERE TO START THE INFERIOR PROCESS

STPROC: SETZB T4,FLAG               ;INITIALIZE COUNTER AND FLAG
        MOVE T1,HANDLE              ;GET PROCESS HANDLE
        MOVEI T2,SLEEP              ;GET ADDRESS AT WHICH TO START NEW PROCESS
        SFORK                       ;START THE NEW PROCESS

; MAIN PROCESSING LOOP

LOOP:   AOS T4                      ;INCREMENT COUNTER
        SKIPN FLAG                  ;HAS TIME ELAPSED YET ?
        JRST LOOP                   ;NO, GO DO MORE PROCESSING

; HERE WHEN LOWER PROCESS HAS INTERRUPTED

        TMSG <
Counter has reached >               ;OUTPUT FIRST PART OF MESSAGE
        MOVX T1,.PRIOU              ;GET PRIMARY OUTPUT JFN
        MOVE T2,T4                  ;GET COUNTER VALUE
        MOVEI T3,^D10               ;USE DECIMAL RADIX
        NOUT                        ;OUTPUT CURRENT COUNTER VALUE
         JSERR                      ;UNEXPECTED ERROR
        TMSG <
>
        JRST STPROC                 ;CONTINUE COUNTING

; PROGRAM PERFORMED BY INFERIOR PROCESS TO WAIT FOR ONE-HALF MINUTE

SLEEP:  MOVX T1,^D30*^D1000         ;SLEEP FOR ONE-HALF MINUTE
        DISMS                       ;DISMISS FOR SPECIFIED TIME
        SETOM FLAG                  ;TELL SUPERIOR PROCESS 30 SECONDS HAVE ELASPED
        HALTF                       ;FINISHED

; CONSTANTS AND VARIABLES

PDL:    BLOCK 50
HANDLE: BLOCK 1                     ;PROCESS HANDLE
FLAG:   BLOCK 1

        END START
```

Example 2 - This program illustrates how an inferior process may be used as a source of timer interrupts. The main program increments a counter. It has an inferior process running for the sole purpose of timing 10 second intervals. Each time the inferior process has timed 10 seconds, it stops and interrupts the main program. The main program then reports how many more times it has incremented the counter since the last 10 second interrupt.

```
        SEARCH MONSYM, MACSYM
        .REQUIRE SYS:MACREL

        T1=1
        T2=2
        T3=3
        T4=4

START:  RESET                       ;RELEASE FILES, ETC.

; SET UP THE INTERRUPT SYSTEM

        MOVX T1,.FHSLF              ;GET OUR PROCESS HANDLE
        MOVE T2,[LEVTAB,,CHNTAB]    ;GET TABLE ADDRESSES
        SIR                         ;SET INTERRUPT TABLE ADDRESSES
        MOVX T2,1B<.ICIFT>          ;GET PROCESS-TERMINATION-CHANNEL BIT
        AIC                         ;ACTIVATE PROCESS TERMINATION CHANNEL
        EIR                         ;ENABLE THE SYSTEM.

; CREATE AND START THE INFERIOR PROCESS

        MOVX T1,CRZMAP+CRZST+SLEEP
        CFORK                       ;CREATE AND START TIMER AT "SLEEP"
         ERJMP [ JSHLT ]            ;UNEXPECTED ERROR.
        MOVEM T1,HANDLE             ;SAVE PROCESS HANDLE

;INITIALIZE THE COUNTER

STPROC: SETZB T4,OLDT4              ;CLEAR THE COUNTER

;MAIN LOOP OF PROGRAM WHICH JUST KEEPS COUNTING.  (REAL
;APPLICATION WOULD PRESUMABLY HAVE A MORE USEFUL MAIN PROGRAM.)

LOOP:   AOJA T4,LOOP                ;JUST KEEP INCREMENTING...

; HERE WHEN LOWER PROCESS HAS INTERRUPTED

PROINT: MOVEM 17,IACS+17            ;SAVE AC 17
        MOVEI 17,IACS               ;MAKE POINTER FOR REST OF ACS
        BLT 17,IACS+16              ;SAVE REST OF ACS
        TMSG <NUMBER OF COUNTS: >
        MOVEI T1,.FRIOU             ;GET PRIMARY OUTPUT JFN
        EXCH T4,OLDT4               ;SAVE NEW COUNTER VALUE.
        SUB T4,OLDT4                ;FIND NUMBER OF COUNTS SINCE LAST TIME
        MOVM T2,T4                  ;MAKE IT POSITIVE
        MOVEI T3,^D10               ;USE DECIMAL RADIX
        NOUT                        ;TYPE NUMBER OF COUNTS SINCE LAST TIME
         ERCAL [ JSERR              ;UNEXPECTED NOUT FAILURE
                 RET ]              ;RETURN
        TMSG <
>                                   ;END THE LINE
        MOVE T1,HANDLE              ;GET HANDLE ON TIMER PROCESS.
        MOVEI T2,SLEEP              ;GET THE PC WE WANT TO START IT AT.
        SFORK                       ;RESTART THE TIMER.
        MOVSI 17,IACS               ;GET POINTER TO SAVED ACS
        BLT 17,17                   ;RESTORE SAVED ACS
```

```
        DEBRK                       ;DISMISS INTERRUPT

;THE FOLLOWING LOOP IS EXECUTED AS A LOWER PROCESS TO DO THE
;TIMING.  IT SLEEPS FOR 10 SECONDS AND THEN STOPS.

SLEEP:  MOVX T1,^D10*^D1000         ;GET 10 SECONDS
        DISMS                       ;SLEEP
        HALTF                       ;STOP AND INTERRUPT THE MAIN PROGRAM

; CONSTANTS AND VARIABLES

CHNTAB: REPEAT ^D19, <EXP 0>        ;CHANNELS 0-18 ARE NOT USED
        1,,PROINT                   ;PROCESS TERMINATION INTERRUPT CHANNEL
        REPEAT ^D15,<EXP 0>         ;REMAINING CHANNELS ARE NOT USED
LEVTAB: RETPC1                      ;RETURN PC STORED AT RETPC1 FOR LEVEL 1
        0                           ;LEVEL 2 NOT USED
        0                           ;LEVEL 3 NOT USED
HANDLE: BLOCK 1                     ;PROCESS HANDLE
RETPC1: BLOCK 1                     ;RETURN PC STORED HERE ON INTERRUPTS
OLDT4:  BLOCK 1                     ;HOLDS TIMER VALUE AT LAST INTERRUPT
IACS:   BLOCK 20                    ;STORAGE FOR ACS DURING INTERRUPTS

        END START
```

Example 3 - This program creates an inferior process which waits until a line has been typed on the terminal.

```
TITLE FRKDOC - EXAMPLE OF USING AN INFERIOR PROCESS TO WAIT UNTIL A LINE IS TYPED

        SEARCH MONSYM, MACSYM
        .REQUIRE SYS:MACREL

        T1=1
        T2=2
        T3=3
        T4=4
        P=17

START:  RESET                           ;RELEASE FILES, ETC.
        MOVE P,[IOWD 50,PDL]            ;INITIALIZE PUSH-DOWN LIST IN CASE OF ERRORS
        MOVX T1,CRZMAP                  ;MAKE NEW PROCESS SHARE THIS PROCESS'S MEMORY
        CFORK                           ;CREATE A NEW PROCESS
         JSHLT                          ;UNEXPECTED ERROR.
        SETZB T4,FLAG                   ;INITIALIZE COUNTER AND FLAG
        MOVEI T2,GETCOM                 ;GET ADDRESS AT WHICH TO START NEW PROCESS
        SFORK                           ;START THE NEW PROCESS

; MAIN PROCESSING LOOP

LOOP:   AOS T4                          ;INCREMENT COUNTER
        SKIPN FLAG                      ;HAS A LINE BEEN INPUT YET ?
        JRST LOOP                       ;NO, GO DO MORE PROCESSING

; HERE WHEN INFERIOR PROCESS HAS INPUT A LINE OF TEXT

        TMSG <
Counter has reached >                   ;OUTPUT FIRST PART OF MESSAGE
        MOVX T1,.PRIOU                  ;GET PRIMARY OUTPUT JFN
        MOVE T2,T4                      ;GET COUNTER VALUE
        MOVEI T3,^D10                   ;USE DECIMAL RADIX
        NOUT                            ;OUTPUT CURRENT COUNTER VALUE
         JSERR                          ;UNEXPECTED ERROR
        TMSG <
Echo check: >                           ;OUTPUT FIRST PART OF MESSAGE
        HRROI T1,BUFFER                 ;GET POINTER TO BUFFER
        PSOUT                           ;OUTPUT TEXT JUST ENTERED
        HALTF                           ;STOP
        JRST START                      ;IN CASE PROGRAM CONTINUED

; PROGRAM PERFORMED BY INFERIOR PROCESS TO INPUT A LINE OF TEXT

GETCOM: HRROI T1,BUFFER                 ;GET POINTER TO TEXT BUFFER
        MOVEI T2,^D120                  ;GET COUNT OF MAX # OF CHARACTERS
        SETZM T3                        ;NO RETYPE BUFFER
        RDTTY                           ;READ A LINE FROM THE TERMINAL
         JSERR                          ;UNEXPECTED ERROR
        SETOM FLAG                      ;TELL SUPERIOR PROCESS A LINE HAS BEEN INPUT
        HALTF                           ;FINISHED

 ; CONSTANTS AND VARIABLES

 PDL:    BLOCK 50
 BUFFER: BLOCK 50
 FLAG:   BLOCK 1

         END START
```

CHAPTER 6

ENQUEUE/DEQUEUE FACILITY

6.1 OVERVIEW

Many times users are placed in situations where they must share files with other users. Each user wants to be quaranteed that while reading a file, other users are reading the same data and while writing a file, no users are also writing, or even reading, the same portion of the file.

Consider a data file used by members of an insurance company. When many agents are reading individual accounts from the data file, they can all access the file simultaneously because no one is changing any portion of the data. However, when an agent desires to modify or replace an individual account, that portion of the file should be accessed exclusively by that agent. None of the other agents wants to access accounts that are being changed until after the changes are made.

By using the ENQ/DEQ facility, cooperating users can insure that resources are shared correctly and that one user's modifications do not interfere with another user's. Examples of resources that can be controlled by this facility are devices, files, operations on files (e.g., READ, WRITE), records, and memory pages. This facility can be used for dynamic resource allocation, computer networks, and internal monitor queueing. However, control of simultaneous updating of files by multiple users is its most common application.

The ENQ/DEQ facility insures data integrity among processes only when the processes cooperate in their use of both the facility and the physical resource. Use of the facility does not prevent non-cooperating processes from accessing a resource without first enqueueing it. Nor does the facility provide protection from processes using it in an incorrect manner.

A resource is defined by the processes using it and not by the system. Because there is competition among processes for use of a resource, each resource is associated with a queue. This queue is the ordering of the requests for the resource. When a request for the resource is granted, a lock occurs between the process that made the request and the resource. For the duration of the lock, that process is the owner of the resource. Other processes requesting access to the resource are placed in the queue until the owner relinquishes the lock. However, there can be more than one owner of a resource at a time; this is called shared ownership (refer to Section 6.2).

Processes obtain access to a specific resource by placing a request in the queue for the resource. This request is generated by the ENQ monitor call. When finished with the resource, the process then issues the DEQ monitor call. This call releases the lock by removing the request from the queue and makes the resource available to the

next waiting process. This cycle continues until all requests in the queue have been satisfied.

6.2 RESOURCE OWNERSHIP

Ownership for a resource can be requested as either exclusive or shared. Exclusive ownership occurs when a process requests sole use of the resource. When a process is granted exclusive ownership, no other process will be allowed to use the resource until the owner relinquishes it. This type of ownership should be requested if the process plans on modifying the resource (e.g., the process is updating a record in a data file). Shared ownership occurs when a process requests a resource, specifying that it will share the use of the resource with other processes. When a process is given shared ownership, other processes also specifying shared ownership are allowed to simultaneously use the resource. Access to a resource should be shared as long as any one process is not modifying the resource.

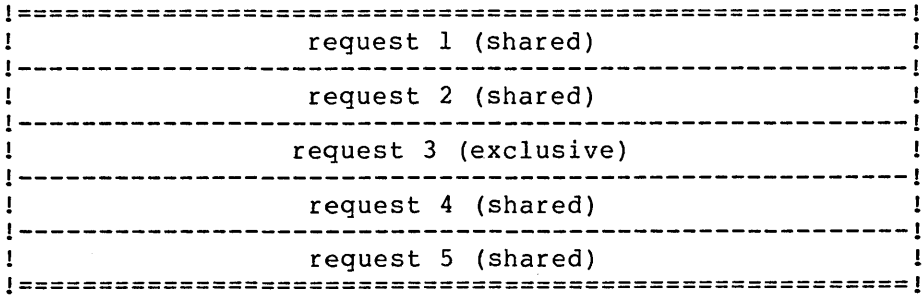
Two conditions determine when a lock to a resource is given to a process:

1. The position of the process' request in the queue for the resource.

2. The type of ownership specified by the process' request.

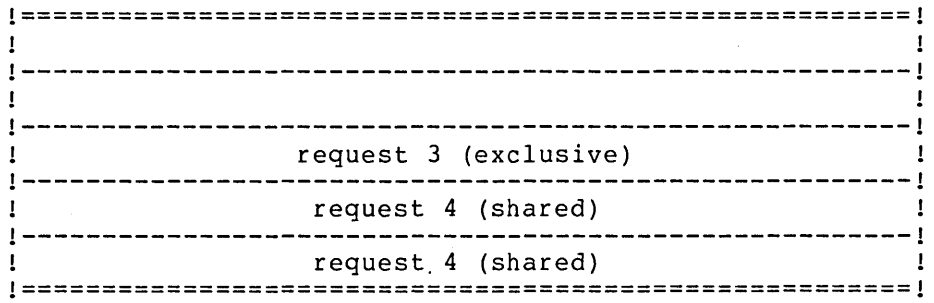
Because each resource has only one queue associated with it, requests for both exclusive and shared ownership of the resource are placed in the same queue. Requests are placed in the queue in the order in which the ENQ facility receives them, and the first request in the queue will be the first one serviced (except in the case of single requests for multiple resources; refer to Section 6.4.1). In other words, the ENQ facility processes requests on a first in, first out basis. If this first request is for shared ownership, that request will be serviced along with all following shared ownership requests up to but not including the first exclusive ownership request. If the first request is for exclusive ownership, no other processes are allowed use of the resource until the first process has released the lock.

Consider the following queue for a particular resource.



Request 1 will be serviced first because it is the first request in the queue. However, since this request is for shared ownership, request 2 can also be serviced. Request 3 cannot be serviced until the processes with request 1 and request 2 release the lock on the resource. Eventually the lock is released by the two processes, and

the first two requests are removed from the queue. The queue now has the following entries:



Request 3 is now first in the queue and is given a lock on the resource. Because the request is for exclusive ownership, no other requests will be serviced. Once the process associated with request 3 releases the lock, both request 4 and request 5 can be serviced because they both are for shared ownership.

6.3 PREPARING FOR THE ENQ/DEQ FACILITY

Before using the ENQ/DEQ facility, the user must obtain an ENQ quota from the system administrator and must obtain the name of the resource desired, the type of protection required, and the level number associated with the resource.

The ENQ quota indicates the total number of requests that can be outstanding for the user at any given time. Any request that would cause the quota to be exceeded results in an error. The user cannot use the ENQ facility if the quota is set to zero.

The resource name has a meaning agreed upon by all users of the specific resource and serves as an identifier of the resource. The system makes no association between the resource name and the physical resource itself; it is the responsibility of the user's process to make that association. The system merely uses the resource name to process requests and handles different resource names as requests for different resources.

The resource name has two parts. In most cases, the first part is the JFN of the file being accessed. Before using the ENQ facility, the user must initialize the file using the appropriate monitor calls (refer to Section 3.1). The second part of the name is a modifier, which is either a pointer to a string or a 33-bit user code. The string uniquely identifies the resource to all users. The pointer can either be a standard byte pointer or be in the form

```
-1,,ADR
```

where ADR is the location of the left-justified ASCIZ text string. The 33-bit user code similarly identifies the resource by representing an item such as a record number or block number. The ENQ facility considers these modifiers as logical strings and does not check for cooperation among the users. Thus, users must be careful when assigning these modifiers to prevent the occurrence of two different modifiers referring to the same resource.

The type of protection desired for the resource is indicated by the first part of the resource name. This part of the name can be one of four values. When the user specifies the JFN of the desired file, the file is subject to the standard access protection of the system. This is the most typical case. When the user specifies -1 instead of a JFN, it means that resources defined within a job are to be accessed only by processes of that job. Other jobs requesting resources of the same name are queued to a different resource. When the user specifies -2 instead of a JFN, it means that the resource can be accessed by any job on the system. A process must have bit SC%ENQ enabled in its capability word to specify this type of protection. If the user specifies -3 instead of a JFN, it means the same type of protection as that given when -2 is specified. However, this is reserved for the monitor and requires that the process have WHEEL or OPERATOR capability enabled. Quotas are not checked when -3 is given instead of a JFN.

In addition to specifying the resource name and type of protection, the user also assigns a level number to each resource. The use of level numbers prevents the occurrence of a deadly embrace situation: the situation where two or more processes are waiting for each to complete, but none of the processes can obtain a lock on the resource it needs for completion. This situation is represented by Figure 6-1.

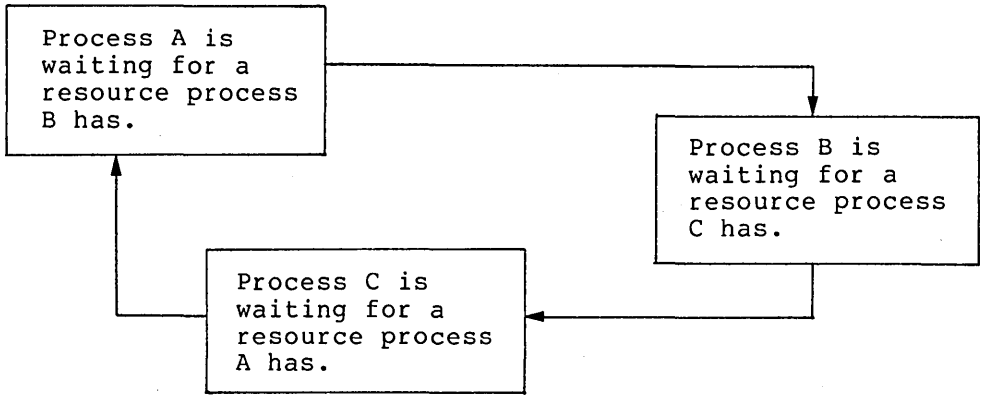


Figure 6-1 Deadly Embrace Situation

Each process is in the queue waiting for the resource it needs, but no request is being serviced because the desired resources are unavailable.

The use of level numbers forces cooperating processes to order their use of resources by requiring that processes request resources in an ascending numerical order and that all processes assign the same level number to a specific resource. This means that the order in which resources are requested is the same for all processes and therefore, requests for the first resource will always precede requests for the second one.

If both of the above requirements are not met, the process requesting the resource receives an error, unless the appropriate flag bit is set (refer to Section 6.4.1.2), and the request is not placed in the queue. Thus, instead of waiting for a resource it will never get, the process is informed immediately that the resource is not available.

6.4 USING THE ENQ/DEQ FACILITY

There are three monitor calls available for the ENQ/DEQ facility: ENQ, to request use of a resource; DEQ, to release a lock on a resource; and ENQC, to obtain information about the queues and to specify access to these queues.

6.4.1 Requesting Use Of A Resource

The user issues the ENQ monitor call to place a request in the queue associated with the desired resource. This call is used to specify the resource name, level number, and type of protection required.

A single ENQ monitor call can be used to request any number of resources. In fact, when desiring multiple resources, the user should request all of them in one call. This method of requesting resources guarantees that the user gets either none or all of the resources requested because the ENQ/DEQ facility never allocates only some of the resources specified in one call. Because all resources in a single call must be available at the same time, the first user requesting a resource (i.e., the first user in the queue for the resource) may not be the first user obtaining it if other resources in the request are currently not available.

A single call for multiple resources is not functionally the same as a series of single calls of those resources. In a single call, the entire request is rejected if an error is returned for one of the resources specified. In a series of single calls, each request that did not return an error will be queued.

The ENQ monitor call accepts two words of arguments in AC1 and AC2. The first word contains the code of the desired function, and the second contains the address of the argument block. Thus,

AC1: function code

AC2: address of argument block

6.4.1.1 ENQ Functions - The functions that can be requested in the ENQ call are described in Table 6-1.

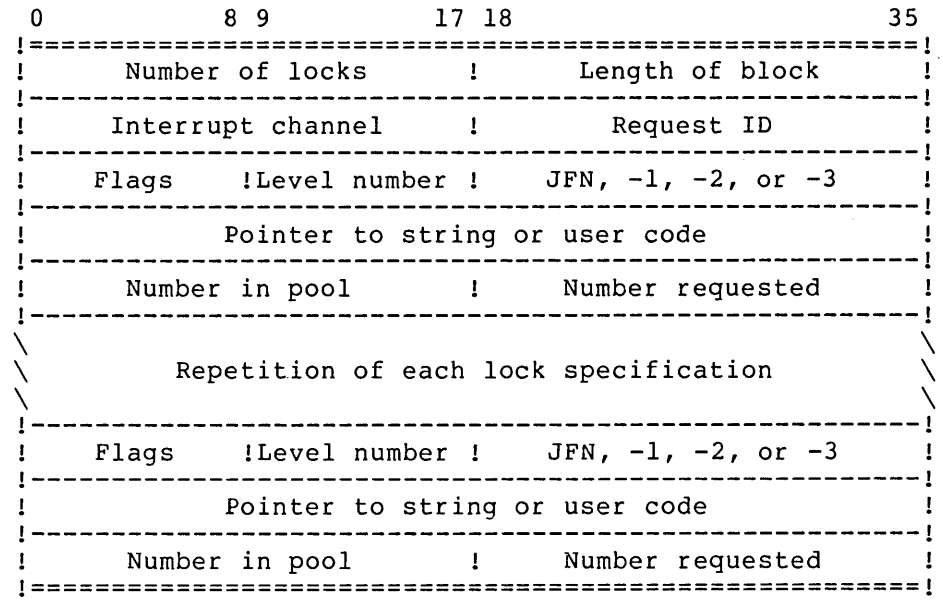
Table 6-1
ENQ Functions

Code	Symbol	Meaning
0	.ENQBL	Queue the requests and block the process until all requested locks are acquired. This function returns an error code only if the ENQ call is not correctly specified.
1	.ENQAA	Queue the requests and acquire the locks only if all requested resources are immediately available. If the resources are available, all will be allocated to the process. If any one of the resources is not available, no requests are queued, no locks are

Table 6-1 (Cont.)
ENQ Functions

Code	Symbol	Meaning
		acquired, and an error code is returned in AC1.
2	.ENQSI	Queue the requests for all specified resources. If all resources are available, this function is identical to the .ENQBL function. If all resources are not immediately available, the requests are queued, and a software interrupt is generated when all requested resources have been given to the process.
3	.ENQMA	Change the ownership access of a previously-queued request (refer to bit EN%SHR below). The access for each lock in this request is compared with the access for each lock in the request already queued. No action is taken if the two accesses are the same. If the access in this request is shared and the access in the previous request is exclusive, the ownership access is changed to shared access. Otherwise, an error is returned if: 1. The process tries to change the ownership access from shared to exclusive. If this is desired, the process should issue a DEQ monitor call for the shared request and then issue another ENQ monitor call for exclusive ownership. 2. Any one of the specified locks does not have a pending request. 3. Any one of the specified locks is a pooled resource (refer to Section 6.4.1.2). Each lock specified is checked, and the access is changed for all locks that were correctly given. On receiving an error, the process should issue the ENQC monitor call to determine the current state of each lock (refer to Section 6.4.3).

6.4.1.2 ENQ Argument Block - The format of the argument block is described below.



Word	Symbol	Meaning
0	.ENQLN	Number of locks being requested in the left half, and length of argument block (including this word) in the right half.
1	.ENQID	Number of software interrupt channel in the left half, and request ID in the right half.
2	.ENQLV	Flags and level number in the left half, and JFN, -1, -2 or -3 (refer to Section 6.3) in the right half.
3	.ENQUC	Pointer to string or 5B2+33-bit user code (refer to Section 6.3).
4	.ENQRS	Number of resources in the pool in the left half, and number of resources requested in the right half.

Words .ENQLV, .ENQUC, and .ENQRS (words 2 through 4) are repeated for each lock being requested. These three words are called the lock specification.

Software Interrupts

The software interrupt system is used in conjunction with the .ENQSI function (refer to Section 6.4.1.1). If all locks are not available when the user requests them, the .ENQSI function causes a software interrupt to be generated when the locks become available. The user specifies the software channel on which to receive the interrupt by placing the channel number in the left half of word .ENQID in the argument block.

When the user is waiting for more than one lock to become available, he will receive an interrupt when the last lock is available. If he desires to be informed as each lock becomes available, he can assign the locks to separate channels by issuing multiple ENQ calls. The availability of each lock will then be indicated by the occurrence of an interrupt on each channel.

When the user requests the .ENQSI function, he must initialize the interrupt system first or else an interrupt will not be generated when the locks become available (refer to Chapter 4).

Request ID

The 18-bit request ID is currently not used by the system, but is stored for use by the process. Thus, the process can supply an ID to use as identification for the request. This ID is useful on the .DEQID function of the DEQ monitor call (refer to Section 6.4.2.1).

Flags and Level Numbers

The left half of the first word of each lock specification (.ENQLV) is used for the following flags.

Bit	Symbol	Meaning
0	EN%SHR	Ownership for this resource is to be shared. If this bit is not on, ownership for this resource is to be exclusive.
1	EN%BLN	Ignore the level number associated with this resource. If this bit is set, sequencing errors in level numbers are not considered fatal, and execution of the call continues. On successful completion of the call, AC1 contains either an error code if a sequencing error occurred or zero if a sequencing error did not occur. WARNING A deadly embrace situation may occur when level numbers are not used. Use of these numbers guarantees that such a situation cannot arise; therefore, this bit should not be set.
2-8		Reserved for DEC.
9-17	EN%LVL	Level number associated with this resource. This number is specified by the user and must be agreed upon by all users of the resource. In order to eliminate a deadly embrace situation, users must request resources in numerically increasing order.

The request is not queued, and an error is given, if EN%BLN is not set and

1. The user requests a resource with a level number less than or equal to the highest numbered resource he has requested so far.

2. The level number of this request does not match the level number supplied in previous requests for this resource.

Pooled Resources

Word .ENQRS of each lock specification is used to allocate multiple copies from a pool of identical resources. Bit EN%SHR, indicating shared ownership, is meaningless for pooled resources because each resource in the pool can be owned by only one process at a time. A process can own one or more resources in the pool; however, it cannot own more than there are in the pool or more than there are unowned in the pool.

The left half of word .ENQRS contains the total number of resources existing in the pool. This number is previously agreed upon by all users of the pooled resource. The first user who requests the resource sets this number, and all subsequent requests must specify the same number or an error is given.

The right half of word .ENQRS contains the number of resources being requested by this process. This number must be greater than zero if a pool of resources exists and cannot be greater than the number in the left half. This means that if a pool of resources exists, the user must request at least one resource, but cannot request more than are in the pool.

Once the number of pooled resources is determined, the resources are allocated until the pool is depleted or until a request specifies more resources than are currently available. In the latter case, the user making the request is not given any resources until his entire request can be satisfied. Subsequent requests from other users are not granted until this request is satisfied even though there may be enough resources to satisfy these subsequent requests. As users release their resources, the resources are returned to the pool. When all resources have been returned, they cease to exist, and the next request completely redefines the number of resources in the new pool.

The system assumes that the resource is in a pool if the left half of word .ENQRS of the lock specification is nonzero. Thus the user should set the left half to zero if only one resource of a specific type exists. If this is the case, then the right half of this word is a number defining the group of users who can simultaneously share the resource. This means that when the resource is allocated to a user for shared ownership, only other users in the same group will be allowed access to the resource. The use of sharer groups restricts access to a resource to a set of processes smaller than the set for shared ownership (which is sharer group 0) but larger than the set for exclusive ownership. (Refer to Section 6.5 for more information on sharer groups).

6.4.2 Releasing A Resource

The user issues the DEQ monitor call to remove a request from the queue associated with a resource. The request is removed whether or not the user actually owns a lock on the resource or is only waiting in the queue for the resource.

The DEQ monitor call can be used to remove any number of requests from the queues. If one of the requests cannot be removed, the dequeueing procedure continues until all lock specifications have been processed. An error code is then returned for the last request found that could not be dequeued. The process can then execute the ENQC call (refer to Section 6.4.3) to determine the status of each lock. Thus, unlike the operation of the ENQ call, the DEQ call will dequeue as many resources as it can, even if an error is returned for one of the lock specifications in the argument block. However, when a user attempts to dequeue more pooled resources than he originally allocated, an error code is returned and none of the resources are dequeued.

The DEQ monitor call accepts two words of arguments in AC1 and AC2. The first word contains the code for the desired function, and the second word contains the address of the argument block. Thus,

AC1: function code

AC2: address of argument block

6.4.2.1 DEQ Functions - The DEQ functions are described in Table 6-2.

Table 6-2
DEQ Functions

Code	Symbol	Meaning
0	.DEQDR	Remove the specified requests from the queues. This function is the only one that requires an argument block.
1	.DEQDA	Remove all requests for this process from the queues. This action is taken on a RESET monitor call. An error code is returned if this process has not requested any resources (i.e., if this process has not issued an ENQ).
2	.DEQID	Remove all requests that correspond to the specified request identifier. When this function is specified, the user must place the 18-bit request ID in AC2 on the DEQ call. This function allows the user to release a class of locks in one call without itemizing each lock in an argument block. The function should be used when dequeueing in one call the same locks that were enqueued in one call. For example, with this function the user can specify the ID to be the same as the JFN used in the ENQ call and thus remove all locks to that file at once.

6.4.2.2 DEQ Argument Block - The format of the argument block for function .DEQDR is described below.

Word	Symbol	Meaning
0	.ENQLN	Number of locks being requested in the left half, and length of argument block (including this word) in the right half.
1	.ENQID	Number of software interrupt channel in the left half, and request ID in the right half.
2	.ENQLV	Flags and level number in the left half, and JFN, -1, -2 or -3 (refer to Section 6.3) in the right half.
3	.ENQUC	Pointer to string or 5B2+33-bit user code (refer to Section 6.3).
4	.ENQRS	Number of resources in the pool in the left half, and number of resources requested in the right half.

Words .ENQLV, .ENQUC, and .ENQRS (words 2 through 4) are repeated for each request being dequeued. These three words are called the lock specification.

6.4.3 Obtaining Information About The Resources

The user issues the ENQC monitor call to obtain information about the current status of the given resources. This call can also be used by privileged users to perform various utility functions on the queue structure. The format of the ENQC call is different for these two uses. (Refer to the DECsystem-20 Monitor Calls Reference Manual for the explanation of the privileged use of the ENQC call.)

The ENQC monitor call accepts three words of arguments in AC1 through AC3:

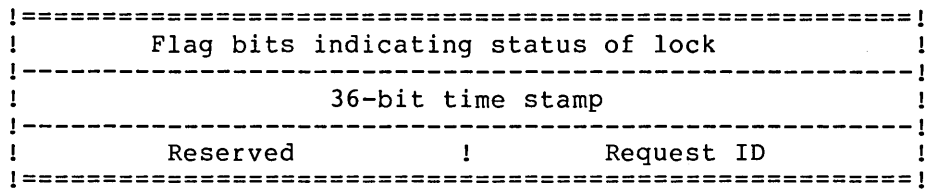
AC1: function code (.ENQCS)

AC2: address of argument block

AC3: address of area to receive status information

The format of the argument block is identical to the format of the ENQ and DEQ argument blocks. The area in which the status is to be returned should be three times as long as the number of locks specified in the argument block.

On successful execution of the ENQC call, the current status of each lock specified is returned as a 3-word entry. This 3-word entry has the following format.



The following flag bits are defined.

Bit	Symbol	Meaning
0	EN%QCE	An error has occurred in the corresponding lock request. Bits 18-35 contain the appropriate error code.
1	EN%QCO	The process issuing the ENQC call is the owner of this lock.
2	EN%QCQ	The process issuing the ENQC call is in the queue waiting for this resource. This bit will be on when EN%QCO is on because a request remains in the queue until a DEQ call is given.
3	EN%QCX	The lock has been allocated for exclusive ownership. When this bit is off, there is no way of determining the number of sharers of the resource.
4	EN%QCB	The process issuing the ENQC call is in the queue waiting for exclusive ownership to the resource. This bit will be off if EN%QCQ is off.
5-8		Reserved for DEC.
9-17	EN%LVL	The level number of the resource.
18-35	EN%JOB	The number of the job that owns the lock. For locks with shared ownership, this value will be the job number of one of the owners. However, this value will be the current job's number if the current job is one of the owners. If this lock is not owned, the value is -1. If EN%QCE is on, this field contains the appropriate error code.

The 36-bit time stamp indicates the last time a process locked the resource. The time is in the universal date-time standard. If no one currently has a lock on the resource, this word is zero.

The request ID returned in the right half of the third word is either the request ID of the current process if that process is in the queue or the request ID of the owner of the lock.

6.5 SHARER GROUPS

Processes can specify the sharing of resources by using sharer group numbers (refer to Section 6.4.1.2). The use of sharer groups restricts the ownership for a resource to a set of processes smaller than the set for shared ownership but larger than the set for exclusive ownership.

Sharer group number 0 is used to indicate the group of all cooperating processes of the resource. This group number is assumed when no group is specified in the ENQ call. To restrict use of the resource, a group number other than 0 must be explicitly specified in the call.

Consider the following example. The resource is the WRITE operation on a file. There are four types of uses of this resource as shown in Figure 6-2.

Process' own use of the resource / Other process' use of the resource	WRITE	Not allowed to write
WRITE	1 Shared, group 0	2 No need to use ENQ/DEQ
Not allowed to write	3 Exclusive	4 Shared, group 1

Figure 6-2 Use of Sharer Groups

In block 1 of the figure, the process owning the lock wishes to allow all cooperating processes to also lock the resource (i.e., to perform the WRITE operation). Therefore, in the ENQ call, the process specifies the resource can be locked by all cooperating processes. In block 2 of the figure, the process does not plan on locking the resource and does not care if other processes lock it. Thus, there is no need for the process to use the ENQ/DEQ facility. In block 3 of the figure, the process desires to lock the resource exclusively and does not want other processes to lock it. Thus, the process obtains exclusive ownership for the resource. In block 4 of the figure, the process does not want to lock the resource immediately but also does not want other processes to lock it because it soon plans to request a lock on the resource. If the process were the only one requesting this type of use, exclusive ownership would be sufficient, because the resource would be unavailable to others as long as the process owned the lock. However, if other processes desire this same type of use, exclusive ownership is not sufficient, because once one process releases the lock, another process with a different type of use could

obtain its own lock. Thus, in this example, sharer group 1 is defined to include all processes with the same type of use (i.e., all processes who do not want to lock the resource immediately but also do not want other processes to lock it). This elimates the problem of another user obtaining the resource for a different type of use.

Sharer group 0 should be sufficient for most uses of the ENQ/DEQ facility. Additional groups should only be needed in those situations where a subset of the cooperating processes must have a specific use of a resource, as in the above example.

6.6 AVOIDING DEADLY EMBRACES

Processes can interact in many undesirable ways if improper communication occurs among the processes or if resources are incorrectly shared. An example of one undesirable situation is the occurrence of a deadly embrace: when two processes are waiting for each other to complete but neither one can gain access to the resource it needs for completion. This situation can be avoided when processes consider the following guidelines.

1. Processes should request resources at the time they need them. If possible, processes should request resources one at a time and release each resource before requesting the next one.

2. Processes should request shared ownership whenever possible. However, the process should not request shared ownership if it plans on modifying the resource.

3. When a process needs more than one resource, it should request these resources in one ENQ call instead of multiple calls for each resource. The process should also release the entire set of resources at once with a single DEQ call.

4. When the use of one resource depends on the use of a second one, the process should define the two resources as one in the ENQ and DEQ calls. However, there is no protection of the resources if they are also requested separately.

5. Occasionally processes use a set of resources and require a lock on the second resource while retaining the lock on the first. In this case, the order in which the locks are obtained should be the same for all users of the set of resources. The same ordering of locks is accomplished by the processes assigning level numbers to each resource. The requirements that processes request resources in ascending numerical order and that all processes use the same level number for a specific resource ensure that a deadly embrace situation will not occur.

CHAPTER 7

INTER-PROCESS COMMUNICATION FACILITY

7.1 OVERVIEW

The Inter-Process Communication Facility (IPCF) allows communication among jobs and system processes. This communication occurs when processes send and receive information in the form of packets. Each sender and receiver has a Process I. D. (PID) assigned to it for identification purposes.

When the sender sends a packet of information to another process, the packet is placed into the receiver's input queue. The packet remains in the queue until the receiver checks the queue and retrieves the packet. Instead of periodically checking its input queue, the receiver can enable the software interrupt system (refer to Chapter 4) to generate an interrupt when a packet is placed in its input queue.

The <SYSTEM>INFO process is the information center for the Inter-Process Communication Facility. This process performs system functions related to PIDs and names, and any process can request these functions by sending <SYSTEM>INFO a packet.

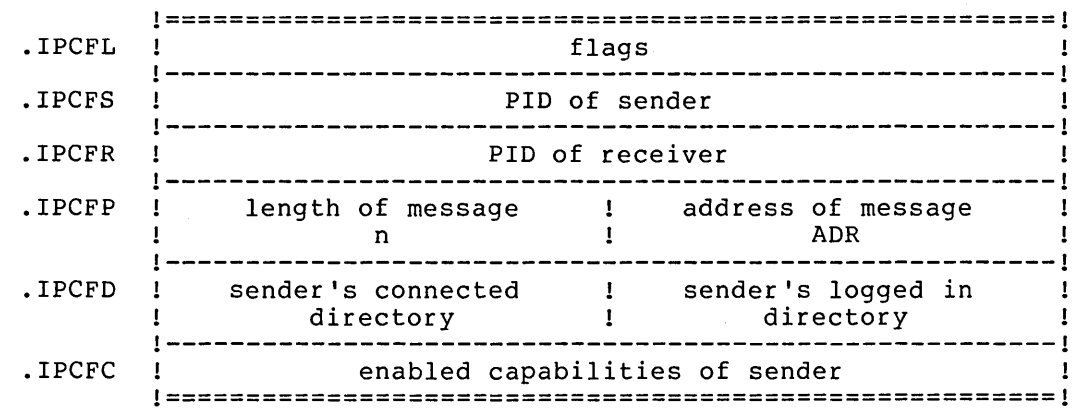
7.2 QUOTAS

Before using IPCF, the user must obtain two quotas from the system administrator: a send packet quota and a receive packet quota. These quotas designate, on a per process basis, the number of sends and receives that can be outstanding at any one time. For example, if the process has a send quota of two and it has sent two packets, it cannot send any more until at least one packet has been retrieved by its receiver. A send packet quota of two and a receive packet quota of five are assumed as the standard quotas. If these quotas are zero, the process cannot use IPCF.

7.3 PACKETS

Information is transferred in the form of packets. Each packet is divided into two portions: a packet descriptor block of four to six words and a packet data block the length of the message. The format of the packet is shown in Figure 7-1.

Packet Descriptor Block



Packet Data Block

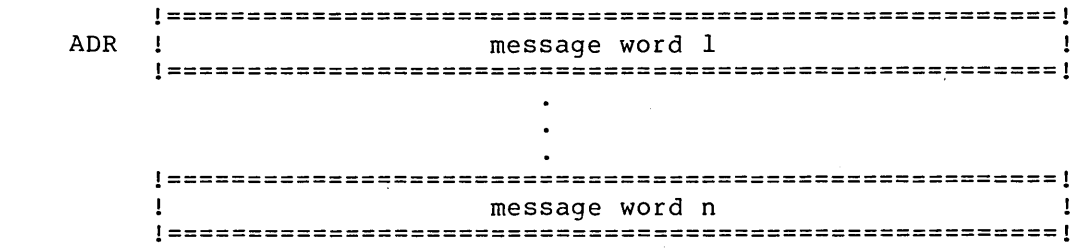


Figure 7-1 IPCF Packet

7.3.1 Flags

There are two types of flags that can be set in word .IPCFL of the packet descriptor block. The flags in the left half of the word are instructions to IPCF for packet communication, and the flags in the right half are descriptions of the data message. The flags in the right half are returned as part of the associated variable (refer to Section 7.4.2). The packet descriptor block flags are described in Table 7-1.

Table 7-1
Packet Descriptor Block Flags

Bit	Symbol	Meaning
0	IP%CFB	Do not block the process if there are no messages in the queue. If this bit is on, the process receives an error if there are no messages.
1	IP%CFS	Use the PID obtained from the address in word .IPCFS of the packet descriptor block as the sender's PID.

Table 7-1 (Cont.)
Packet Descriptor Block Falgs

Bit	Symbol	Meaning
2	IP%CFR	Use the PID obtained from the address in word .IPCFR of the packet descriptor block as the receiver's PID.
3	IP%CFO	Allow the process one send above the send quota. (The standard send quota is two.)
4	IP%TTL	Truncate the message if it is longer than the area reserved for it in the packet data block. If this bit is not on, the process receives an error if the message is too long.
5	IP%CPD	Create a PID to use as the sender's PID. The PID created is returned in word .IPCFS of the packet descriptor block.
6	IP%JWP	Make the PID created be permanent until the job logs out (if both bits IP%CPD and IP%JWP are on). Make the PID created be temporary until the process executes a RESET monitor call (if bit IP%CPD is on and bit IP%JWP is not on). If bit IP%CPD is not on, bit IP%JWP is ignored.
7	IP%NOA	Do not allow other processes to use the PID created when bit IP%CPD is on. If bit IP%CPD is not on, bit IP%NOA is ignored.
8-17		Reserved for DEC.
18	IP%CFP	The packet is privileged. This bit can be set only by a process with WHEEL capability enabled. Refer to the DECsystem-20 Monitor Calls Reference Manual for a description of this bit.
19	IP%CFV	The packet is a page of 512 (decimal) words of data.
20-23		Reserved for DEC.
24-29	IP%CFE	Field for error code returned from <SYSTEM> INFO. Code Symbol Meaning 15 .IPCPI insufficient privileges 16 .IPCUF invalid function 67 .IPCSN <SYSTEM>INFO needs name 72 .IPCFF <SYSTEM>INFO free space exhausted 74 .IPCBP PID has no name or is invalid

Table 7-1 (Cont.)
Packet Descriptor Blocks Flags

Bit	Symbol	Meaning
		75 .IPCDN duplicate name has been specified 76 .IPCNN unknown name has been specified 77 .IPCEN invalid name has been specified
30-32	IP%CFC	System and sender code. This code can be set only by a process with WHEEL capability enabled, but the monitor will return the code so a nonprivileged process can examine it. Code Symbol Meaning 1 .IPCCC Sent by <SYSTEM>IPCF 2 .IPCCF Sent by system-wide <SYSTEM>INFO 3 .IPCCP Sent by receiver's <SYSTEM>INFO
33-35	IP%CFM	Field for special messages. This code can be set only by a process with WHEEL capability enabled, but the monitor will return the code so that a nonprivileged process can examine it. Code Symbol Meaning 1 .IPCFN Process' input queue contains a packet that could not be delivered to intended PID.

7.3.2 PIDS

Any process that wants to send or receive a packet must obtain a PID. The process can obtain a PID by sending a packet to <SYSTEM>INFO requesting that a PID be assigned. The process must also include a symbolic name that is to be associated with the assigned PID.

The symbolic name can be a maximum of 29 characters and can contain any characters as long as it is terminated by a zero word. There should be mutual understanding among processes as to the symbolic names used in order to initiate communication. Once the name is defined, any process referring to that name must specify it exactly character for character.

Before a process can send a packet, it must know the receiver's symbolic name or PID. If only the receiver's name is known, the sender must ask <SYSTEM>INFO for the PID associated with the name, since all communication is via PIDs.

The association between a PID and a name is broken:

1. On a RESET monitor call.
2. When the process is killed or the job logs off the system.
3. When a request to disassociate the PID from the name is made to <SYSTEM>INFO.

<SYSTEM>INFO will not allow a name already associated with a PID to be assigned again unless the owner of the name makes the request. Nor will <SYSTEM>INFO assign a PID once it has been used. This action protects against messages being sent to the wrong receiver by accident.

The PIDs of the sender and the receiver are indicated by words .IPCFS and .IPCFR, respectively, of the packet descriptor block.

7.3.3 Length And Address Of Packet Data Block

Word .IPCFP of the packet descriptor block contains the length and the beginning address of the message. The length specified is one of two types, depending on the type of message (refer to Section 7.3.5). If the message is a short-form message, the length is the actual word length of the message. If the message is a long-form message, the length is 1000 (octal), i.e., one page.

The address specified is either an address or a page number, depending on the type of message (refer to Section 7.3.5). When a message is sent, it is taken from this address. When a message is received, it is placed in this address.

7.3.4 Directories And Capabilities

Words .IPCFD and .IPCFC describe the sender at the time the message was sent and are used by the receiver to validate messages sent to it. These two words are not used when a message is sent, and if the sender of the packet supplies them, they are ignored. However, when a message is received, if the receiver of the packet has reserved space for these words in the packet descriptor block, the system supplies the appropriate values of the sender of the packet. The receiver of the packet does not have to reserve these words if it is not interested in knowing the sender's directories and capabilities.

7.3.5 Packet Data Block

The packet data block contains the message being sent or received. The message can be either a short-form message or a long-form message.

A short-form message is one to n words long, where n is defined by the installation. (Usually, n is assumed to be 10 words.) When a short-form message is sent or received, word .IPCFP of the packet descriptor block contains the actual word length of the message in the left half and the address of the first word of the message in the right half. A process always uses the short form when sending messages to <SYSTEM>INFO.

A long-form message is one page in length (512 decimal words). When a long-form message is sent or received, word .IPCFP of the packet descriptor block contains 1000 (octal) in the left half and the page number of the message in the right half. To send and receive a long-form message, both the sender and receiver must have bit IP%CFV (bit 19) set in the first word of the packet descriptor block, or else an error code is returned.

7.4 SENDING AND RECEIVING MESSAGES

To send a message, the sending process must set up the first four words of the packet descriptor block. The process then executes the MSEND monitor call. After execution of this call, the packet is sent to the intended receiver's input queue.

To receive a message, the receiving process must also set up the first four words of the packet descriptor block. The last two words for the directories and capabilities of the sender can be supplied, and the system will fill in the appropriate values. The process then executes the MRECV monitor call. After execution of this call, a packet is retrieved from the receiver's input queue. The input queue is emptied on a first-message-in, first-message-out basis.

7.4.1 Sending A Packet

The MSEND monitor call is used to send a message via IPCF. Messages are in the form of packets of information and can be sent to a specified PID or to the system process <SYSTEM>INFO. Refer to Section 7.5 for information on sending messages to <SYSTEM>INFO.

The MSEND call accepts two words of arguments. The length of the packet descriptor block is given in AC1, and the beginning address of the packet descriptor block is given in AC2. Thus,

AC1: length of packet descriptor block. The length cannot be less than 4.

AC2: address of packet descriptor block

The packet descriptor block consists of the following four words:

.IPCFL	Flags
.IPCFS	Sender's PID
.IPCFR	Receiver's PID
.IPCFP	Pointer to packet data block containing the message being sent.

Refer to Section 7.3 for the details on the packet descriptor and packet data blocks.

The flags that are meaningful when sending a packet are described below. Refer to Table 7-1 for the complete list of flag bits.

Table 7-2
Flags Meaningful on a MSEND Call

Bit	Symbol	Meaning
1	IP%CFS	The sender's PID is given in word .IPCFS of the packet descriptor block.
2	IP%CFR	The receiver's PID is given in word .IPCFR of the packet descriptor block.
3	IP%CFO	Allow the sender to send one message above its send quota.
5	IP%CPD	Create a PID for the sender and return it in word .IPCFS of the packet descriptor block. The PID created is to be permanent and useable by other processes according to the setting of bits IP%JWP and IP%NOA.
6	IP%JWP	The PID created is to be job wide and permanent until the job logs out. If this bit is not on, the PID created is to be temporary until the process executes the RESET monitor call.
7	IP%NOA	The PID created is not to be used by other processes.
18	IP%CFP	The message being sent is privileged (refer to the DECsystem-20 Monitor Calls Reference Manual).
19	IP%CFV	The message being sent is a long-form message (i.e., a page). The page the message is being sent to cannot be a shared page; it must be a private page.

When bit IP%CFS is on in the flag word, the sender's PID is taken from word .IPCFS of the packet descriptor block. This word is zero if bit IP%CPD is on in the flag word, indicating that a PID is to be created for the sender. In this case, the PID created is returned in word .IPCFS.

When bit IP%CFR is on in the flag word, the receiver's PID is taken from word .IPCFR of the packet descriptor block. If this word is 0, then the receiver of the message is <SYSTEM>INFO. Refer to Section 7.5 for information on sending messages to <SYSTEM>INFO.

On successful execution of the MSEND monitor call, the packet is sent to the receiver's input queue. Word .IPCFS of the packet descriptor block is updated with the sender's PID. Execution of the user's program continues at the second location after the MSEND call.

If execution of the MSEND call is not successful, the message is not sent, and an error code is returned in AC1. The execution of the user's program continues at the instruction following the MSEND call.

7.4.2 Receiving A Packet

The MRECV monitor call is used to retrieve a message from the process' input queue. Before a process can retrieve a message, it must know if the message is a long-form message and also must set up a packet descriptor block.

The MRECV monitor call accepts two words of arguments. The length of the packet descriptor block is given in AC1, and the beginning address of the packet descriptor block is given in AC2. Thus,

AC1: length of packet descriptor block. The length cannot be less than 4.

AC2: address of packet descriptor block

The packet descriptor block can consist of the following six words. The last two words are optional, and if supplied by the receiver, the values of the sender will be filled in by the system.

.IPCFL	Flags
.IPCFS	Sender's PID
.IPCFR	Receiver's PID
.IPCFP	Pointer to packet data block where the message is to be placed.
.IPCFD	Connected and logged-in directories of the sender.
.IPCFC	Enabled capabilities of the sender.

Refer to Section 7.3 for the details on the packet descriptor and packet data blocks.

The flags that are meaningful when receiving a packet are described below. Refer to Table 7-1 for the complete list of flag bits.

Table 7-3
Flags Meaningful on a MRECV Call

Bit	Symbol	Meaning
0	IP%CFB	If there are no packets in the receiver's input queue, do not block the process and return an error code if the queue is empty. If this bit is not on, the process waits until a packet arrives, if the queue is empty.
2	IP%CFR	The receiver's PID is given in word .IPCFR of the packet descriptor block.
4	IP%TTL	Truncate the message if it is larger than the space reserved for it in the packet data block. If this bit is not on and the message is too large, an error code is returned and no message is received.
19	IP%CFV	The message is expected to be a long-form message (i.e., a page). The page the message is being stored into cannot be a shared page; it must be a private page.

The information in word .IPCFS is not supplied by the receiver when the MRECV call is executed. The system fills in the PID of the sender of the packet when the packet is retrieved.

Word .IPCFR is supplied by the receiver. If bit IP%CFR is on in the flag word, then the PID receiving the packet is taken from word .IPCFR of the packet descriptor block. If bit IP%CFR is not on in the flag word, then word .IPCFR contains either -1, to receive a packet for any PID belonging to this process, or -2, to receive a packet for any PID belonging to this job. When -1 or -2 is given, packets are not received in any particular order except that packets from a specific PID are received in the order in which they were sent. Any other values in this word cause an error code to be returned.

The information in words .IPCFD and .IPCFC is also not supplied by the receiver. If these two words have been specified by the receiver, the system fills in the information when the packet is retrieved. Word .IPCFD contains the sender's connected directory in the left half and the sender's logged-in directory in the right half. Word .IPCFC contains the enabled capabilities of the sender. These words describe the sender at the time the message was sent.

On successful execution of the MRECV monitor call, the packet is retrieved and placed into the packet data block as indicated by word .IPCFP of the packet descriptor block. AC1 contains the length of the next packet in the queue in the left half and flags from the next packet in the right half (see below). This word returned in AC1 is called the associated variable of the next packet in the queue. If there is not another packet in the queue, AC1 contains zero. Execution of the user's program continues at the second instruction after the MRECV call.

The flags returned in the right half of AC1 on successful execution of the MRECV monitor call are described below.

Bit	Symbol	Meaning
30-32	IP%CFC	System and sender code, set only by a privileged process. The packet was sent by <SYSTEM>IPCF if the code is 1(.IPCCC). The packet was sent by the system-wide <SYSTEM>INFO if the code is 2(.IPCCF). The packet was sent by the receiver's <SYSTEM>INFO if the code is 3(.IPCCP).
33-35	IP%CFM	Field for return of special messages. If the field contains 1(.IPCFN), then the process' input queue contains a packet that was sent to another PID, but was returned to the sender because it could not be delivered.

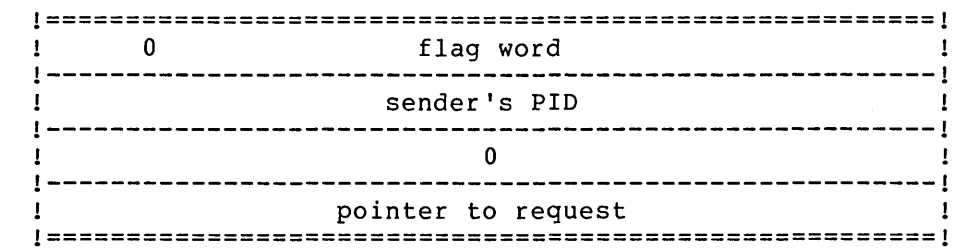
If execution of the MRECV call is not successful, a packet is not retrieved, and an error code is returned in AC1. The execution of the user's program continues at the instruction following the MRECV call.

7.5 SENDING MESSAGES TO <SYSTEM>INFO

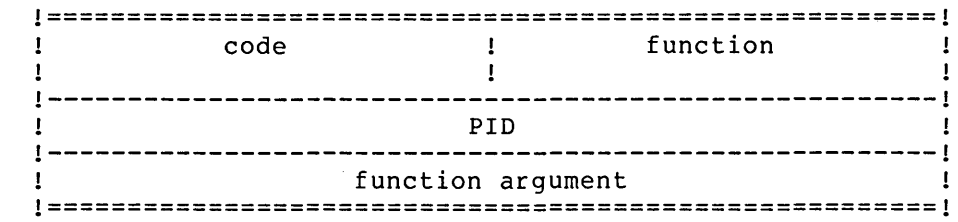
The <SYSTEM>INFO process is the central information utility for IPCF. It performs functions associated with names and PIDs, such as, assigning a PID or a name or returning a name associated with a PID.

A process can request functions to be performed by <SYSTEM>INFO by executing the MSEND monitor call (refer to Section 7.4.1). The message portion of the packet (i.e., the packet data block) sent to <SYSTEM>INFO contains the request being made. In other words, the total request to <SYSTEM>INFO is a packet consisting of a packet descriptor block and a packet data block containing the request.

Packet Descriptor Block



Packet Data Block



Refer to Section 7.4.1 for the descriptions of the words in the packet descriptor block. The receiver's PID (word .IPCFR) is 0 when sending a packet to <SYSTEM>INFO.

7.5.1 Format Of <SYSTEM>INFO Requests

As mentioned previously, the packet data block (i.e., the message portion) of the packet contains the request to <SYSTEM>INFO.

The first word (word .IPCI0) contains a user-defined code in the left half and the function being requested in the right half. The user-defined code is used to associate the response from <SYSTEM>INFO with the correct request. The functions that the process can request of <SYSTEM>INFO are described in Table 7-4.

The second word (word .IPCI1) contains a PID associated with a process that is to receive a duplicate of any response from <SYSTEM>INFO. If this word is zero, the response from <SYSTEM>INFO is sent only to the process making the request.

The third word (word .IPCI2) contains the argument for the function specified in the right half of word .IPCI0. The argument is different depending on the function being requested. The arguments for the functions are described in Table 7-4.

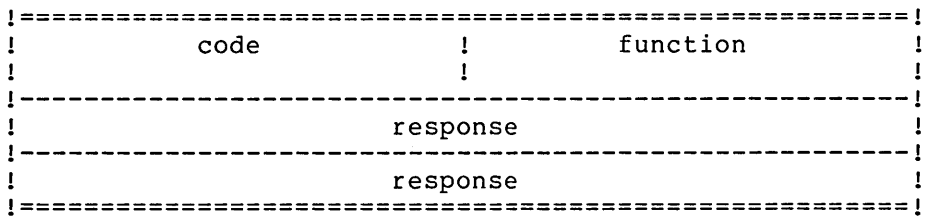
Table 7-4
<SYSTEM>INFO Functions and Arguments

Function	Argument	Meaning
.IPCIW	name	Return the PID associated with the given name (refer to Section 7.3.2 for the description of the name).
.IPCIG	PID	Return the name associated with the given PID.
.IPCII	name in ASCIZ	Assign the given name to the PID associated with the process making the request. The PID is permanent if IP%JWP was set in the flag word when the PID was originally created (refer to Table 7-1).

7.5.2 Format Of <SYSTEM>INFO Responses

Responses from <SYSTEM>INFO are in the form of a packet sent to the process that made the request. A copy of the response is sent to the PID given in word .IPCI1, if any.

The message portion (i.e., the packet data block) of the packet contains the response from <SYSTEM>INFO. The format of this response is



The first word (word .IPCI0) contains the user-defined code in the left half and the function that was requested in the right half. These values are copied from the values given in the request.

The second and third words (words .IPCI1 and .IPCI2) contain the response from the function requested of <SYSTEM>INFO. The response is different depending on the function requested. The responses from the functions are described in Table 7-5.

Table 7-5
<SYSTEM>INFO Responses

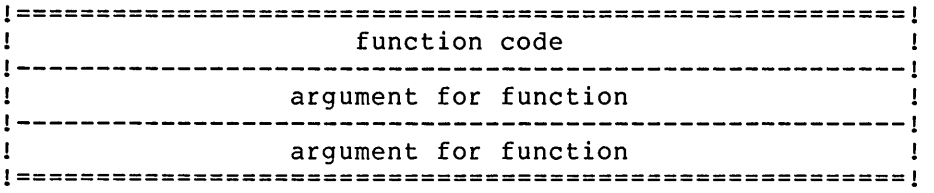
Function Requested	Response
.IPCIW	The PID associated with the name given in the request is returned in word .IPCI1.
.IPCIG	The name associated with the PID given in the request is returned in word .IPCI1.
.IPCII	No response is returned.

7.6 PERFORMING IPCF UTILITY FUNCTIONS

A process can request various functions to be performed by executing the MUTIL monitor call. Some of these functions are enabling and disabling PIDs, creating and deleting PIDs, and returning quotas. Several of the functions that can be requested are privileged functions. These are described in the DECsystem-20 Monitor Calls Reference Manual.

The MUTIL monitor call accepts two words of argument. The length of the argument block is given in AC1, and the beginning address of the argument block is given in AC2.

The argument block has the following format:



The arguments are different, depending on the function being requested. Any values resulting from the function requested are returned in the argument block, starting at the second word.

Table 7-6 describes the functions that can be requested, the arguments for the functions, and the values returned from the functions.

Table 7-6
MUTIL Functions

Function	Meaning
.MUENB	Allow the PID given to receive packets. If the process executing the call is not the owner of the PID, the process must be privileged. Argument PID Value Returned None
.MUDIS	Disable the PID given from receiving packets. If the process executing the call is not the owner of the PID, the process must be privileged. Argument PID Value Returned None

Table 7-6 (Cont.)
MUTIL Functions

Function	Meaning
.MUGTI	Return the PID associated with <SYSTEM>INFO. Argument PID or job number Value Returned PID of <SYSTEM>INFO
.MUDES	Delete the PID given. The process executing the call must own the PID being deleted. Argument PID to be deleted Value Returned None
.MUCRE	Create a PID for the process or job given. If the job number given is not that of the process executing the call, the process must be privileged. The flag bits that can be specified are IP%JWP and IP%NOA (refer to Table 7-1 for their descriptions). Argument flag bits in the left half, and process handle or job number in the right half Value Returned PID that was created
.MUFOJ	Return the number of the job associated with the PID given. Argument PID Value Returned Job number associated with PID given
.MUFJP	Return all PIDs associated with the job given. Argument job number or PID belonging to the job Values Returned Two-word entries for each PID belonging to the job. The first word of the entry is the PID, and the second word has bits IP%JWP and IP%NOA set if appropriate (refer to Table 7-1 for the descriptions of these bits). The list of entries returned is terminated by a zero word.

Table 7-6 (Cont.)
MUTIL Functions

Function	Meaning
.MUFSQ	Return the send quota and the receive quota for the PID given. Argument PID Values Returned Send quota in bits 18-26 and receive quota in bits 27-35.
.MUFFP	Return all PIDs associated with the process of the PID given. Argument PID Values Returned Two-word entries for each PID belonging to the process. The first word of the entry is the PID, and the second word has bits IP%JWP and IP%NOA set if appropriate (refer to Table 7-1 for the descriptions of these bits). The list of entries returned is terminated by a zero word.
.MUFPQ	Return the maximum number of PIDs allowed for the job given. Argument Job number or PID belonging to the job Value Returned Number of PIDs allowed for the job given
.MUQRY	Return the packet descriptor block for the next packet in the queue of the PID given. Argument PID, -1 to return the next descriptor block for the process, or -2 to return the next descriptor block for the job Values Returned Packet descriptor block of next packet in queue.
.MUAPF	Associate the PID given with the process given. Arguments PID process handle Value Returned None

Table 7-6 (Cont.)
MUTIL Functions

Function	Meaning
.MUPIC	Place the PID given on the software channel given in order to cause an interrupt to be generated when a packet is received in the input queue of the PID given. Argument PID channel number, or -1 to remove the given PID from its current channel Value Returned None
.MUMPS	Return the maximum packet size for the PID given. Argument PID Value Returned Maximum packet size for PID

On successful completion of the MUTIL monitor call, the function requested is performed, and any value is returned in the argument block. Execution of the user's program continues at the second location following the MUTIL call.

If execution of the MUTIL monitor call is not successful, no requested function is performed and an error code is returned in AC1. Execution of the user's program continues at the location following the MUTIL call.

APPENDIX A

ERROR CODES AND MESSAGE STRINGS

Many monitor calls return an error number (usually in the right half of AC1) on a failure return. This error number indicates the reason that the call could not perform its intended function. The error number is associated with a unique error symbol and message string, all three of which are defined in the MONSYM file. The ERSTR monitor call can be used to translate the returned number into its corresponding message string. Refer to the DECsystem-20 Monitor Calls Reference Manual for the description of this call.

Symbol	Code	Message String
LGINX1	600010	Invalid account identifier
LGINX2	600011	Directory is "files-only" and cannot be logged in to
LGINX3	600012	Internal format of directory is incorrect
LGINX4	600013	Invalid password
LGINX5	600014	Job is already logged in
LOUTX1	600035	Illegal to specify job number when logging out own job
LOUTX2	600036	No such job
CACTX1	600045	Invalid account identifier
CACTX2	600046	Job is not logged in
EFCTX1	600050	WHEEL or OPERATOR capability required
EFCTX2	600051	Entry cannot be longer than 64 words
EFCTX3	600052	Fatal error when accessing FACT file
GJFX1	600055	Desired JFN invalid
GJFX2	600056	Desired JFN not available
GJFX3	600057	No JFN available
GJFX4	600060	Invalid character in filename
GJFX5	600061	Field cannot be longer than 39 characters
GJFX6	600062	Device field not in a valid position
GJFX7	600063	Directory field not in a valid position
GJFX8	600064	Directory terminating delimiter is not preceded by a valid beginning delimiter
GJFX9	600065	More than one name field is not allowed
GJFX10	600066	Generation number is not numeric
GJFX11	600067	More than one generation number field is not allowed
GJFX12	600070	More than one account field is not allowed
GJFX13	600071	More than one protection field is not allowed
GJFX14	600072	Invalid protection
GJFX15	600073	Invalid confirmation character
GJFX16	600074	No such device
GJFX17	600075	No such directory
GJFX18	600076	No such filename
GJFX19	600077	No such file type
GJFX20	600100	No such generation number
GJFX21	600101	File was expunged
GJFX22	600102	Job Storage Block full
GJFX23	600103	Directory full
GJFX24	600104	File not found

GJFX27	600107	File already exists (new file required)
GJFX28	600110	Device is not on-line
GJFX29	600111	Device is not available to this job
GJFX30	600112	Account is not numeric
GJFX31	600113	Invalid wildcard designator
GJFX32	600114	No files match this specification
GJFX33	600115	Filename was not specified
GJFX34	600116	Invalid character "?" in file specification
GJFX35	600117	Directory access privileges required
OPNX1	600120	File is already open
OPNX2	600121	File does not exist
OPNX3	600122	Read access required
OPNX4	600123	Write access required
OPNX5	600124	Execute access required
OPNX6	600125	Append access required
OPNX7	600126	Device assigned to another job
OPNX8	600127	Device is not on-line
OPNX9	600130	Invalid simultaneous access
OPNX10	600131	Entire public disk full
OPNX12	600133	List access required
OPNX13	600134	Invalid access requested
OPNX14	600135	Invalid mode requested
OPNX15	600136	Read/write access required
OPNX16	600137	File has bad index block
OPNX17	600140	No room in job for long file page table
OPNX18	600141	Reserved
OPNX19	600142	Reserved
OPNX20	600143	Reserved
OPNX21	600144	Reserved
OPNX22	600145	Reserved
DESX1	600150	Invalid source/destination designator
DESX2	600151	Terminal is not available to this job
DESX3	600152	JFN is not assigned
DESX4	600153	Invalid use of terminal designator or string pointer
DESX5	600154	File is not open
DESX6	600155	Device is not a terminal
DESX7	600156	JFN cannot refer to output wildcard designators
DESX8	600157	File is not on disk
CLSX1	600160	File is not open
CLSX2	600161	File cannot be closed by this process
RJFNX1	600165	File is not closed
RJFNX2	600166	JFN is being used to accumulate filename
RJFNX3	600167	JFN is not accessible by this process
DELFX1	600170	Delete access required
SFPTX1	600175	File is not open
SFPTX2	600176	Illegal to reset pointer for this file
SFPTX3	600177	Invalid byte number
CNDIX1	600200	Incorrect password
CNDIX3	600202	Invalid directory number
CNDIX5	600204	Job is not logged in
SFBSX1	600210	Illegal to change byte size for this opening of file
SFBSX2	600211	Invalid byte size
IOX1	600215	File is not opened for reading
IOX2	600216	File is not opened for writing
IOX3	600217	File is not open for random access
IOX4	600220	End of file reached
IOX5	600221	Device or data error
IOX6	600222	Illegal to write beyond absolute end of file
PMAPX1	600240	Invalid access requested
PMAPX2	600241	Invalid use of PMAP
SPACX1	600245	Invalid access requested
FRKHX1	600250	Invalid process handle
FRKHX2	600251	Illegal to manipulate a superior process
FRKHX3	600252	Invalid use of multiple process handle

FRKHX4	600253	Process is running
FRKHX6	600255	All relative process handles in use
SPLFX1	600260	Process is not inferior or equal to self
SPLFX2	600261	Process is not inferior to self
SPLFX3	600262	New superior process is inferior to intended inferior
GTABX1	600267	Invalid table index
GTABX2	600270	Invalid table number
GTABX3	600271	GETAB privileges required
RUNTX1	600273	Invalid process handle -3 or -4
STADX1	600275	WHEEL or OPERATOR capability required
STADX2	600276	Invalid date or time
ASNDX1	600300	Device is not assignable
ASNDX2	600301	Illegal to assign this device
ASNDX3	600302	No such device
ATACX1	600320	Invalid job number
ATACX2	600321	Job already attached
ATACX3	600322	Incorrect user number
ATACX4	600323	Incorrect password
ATACX5	600324	This job has no controlling terminal
STDVX1	600332	No such device
DEVX1	600335	Invalid device designator
DEVX2	600336	Device already assigned to another job
DEVX3	600337	Device is not on-line
MNTX1	600345	Invalid directory format
MNTX2	600346	Device is not on-line
MNTX3	600347	Device is not mountable
TERMX1	600350	Invalid terminal code
TLNKX1	600351	Illegal to set remote to object before object to remote
ATIX1	600352	Invalid channel number
ATIX2	600353	Control-C capability required
TLNKX2	600356	Link was not received within 15 seconds
TLNKX3	600357	Links full
TTYX1	600360	Not a terminal or no such terminal
RSCNX1	600361	Overflowed rescan buffer, input string truncated
CFRKX3	600363	Insufficient resources available
KFRKX1	600365	Illegal to kill top level process
KFRKX2	600366	Illegal to kill self
RFRKX1	600367	Processes are not frozen
HFRKX1	600370	Illegal to halt self with HFORK
GFRKX1	600371	Invalid process handle
GETX1	600373	Invalid save file format
GETX2	600374	System Special Pages Table full
SFRVX1	600377	Invalid position in entry vector
NOUTX1	600407	Radix is not in range 2 to 10
NOUTX2	600410	Column overflow
IFIXX1	600414	Radix is not in range 2 to 10
IFIXX2	600415	First character is not a digit
IFIXX3	600416	Overflow (number is greater than 2**35)
GFDBX1	600424	Invalid displacement
GFDBX2	600425	Invalid number of words
GFDBX3	600426	List access required
CFDBX1	600430	Invalid displacement
CFDBX2	600431	Illegal to change specified bits
CFDBX3	600432	Write or owner access required
CFDBX4	600433	Invalid value for specified bits
DUMPX1	600440	Command list error
DUMPX2	600441	JFN is not open in dump mode
DUMPX3	600442	Address error (too big or crosses end of memory)
DUMPX4	600443	Access error (cannot read or write data in memory)
RNAMX1	600450	Files are not on same device
RNAMX2	600451	Destination file expunged
RNAMX3	600452	Write or owner access to destination file required
RNAMX4	600453	Insufficient resources to rename file
BKJFX1	600454	Illegal to back up terminal pointer twice

TIMEX1	600460	Time cannot be greater than 24 hours
ZONEX1	600461	Time zone out of range
ODTNX1	600462	Time zone must be USA or Greenwich
DILFX1	600464	Invalid date format
TILFX1	600465	Invalid time format
DATEX1	600466	Year out of range
DATEX2	600467	Month is not less than 12
DATEX3	600470	Day of month too large
DATEX4	600471	Day of week is not less than 7
DATEX5	600472	Date out of range
DATEX6	600473	System date and time are not set
SMONX1	600516	WHEEL or OPERATOR capability required
SACTX1	600530	File is not on multiple-directory device
SACTX2	600531	Job Storage Block full
SACTX3	600532	Directory requires numeric account
SACTX4	600533	Write or owner access required
GACTX1	600540	File is not on multiple-directory device
GACTX2	600541	File expunged
FFUFX1	600544	File is not open
FFUFX2	600545	File is not on multiple-directory device
FFUFX3	600546	No used page found
DSMX1	600555	File(s) not closed
RDDIX1	600560	Illegal to read directory for this device
SIRX1	600570	Table address is not greater than 20
SSAVX1	600600	Illegal to save files on this device
SSAVX2	600601	Page count is not less than or equal to 1000
SEVEX1	600610	Entry vector is not less than 777
WHELX1	600614	WHEEL or OPERATOR capability required
CAPX1	600615	WHEEL or OPERATOR capability required
PEEKX2	600617	Read access failure on monitor page
CRDIX1	600620	WHEEL or OPERATOR capability required
CRDIX2	600621	Illegal to change number of old directory
CRDIX3	600622	Job Storage Block full
CRDIX4	600623	Sub index full
CRDIX5	600624	Directory name not given
CRDIX7	600626	File(s) open in directory
GTDIX1	600640	WHEEL or OPERATOR capability required
GTDIX2	600641	No such directory number
FLINX1	600650	First character is not blank or numeric
FLINX2	600651	Number too small
FLINX3	600652	Number too large
FLINX4	600653	Invalid format
FLOTX1	600660	Column overflow in field 1 or 2
FLOTX2	600661	Column overflow in field 3
FLOTX3	600662	Invalid format specified
HPTX1	600670	Undefined clock number
FDFRX1	600700	Not a multiple-directory device
FDFRX2	600701	No such directory number
RNAMX5	600750	Destination file is not closed
RNAMX6	600751	Destination file has bad page table
RNAMX7	600752	Source file expunged
RNAMX8	600753	Write or owner access to source file required
RNAMX9	600754	Source file is empty
RNMX10	600755	Source file is not closed
RNMX11	600756	Source file has bad page table
RNMX12	600757	Illegal to rename to self
GJFX36	600760	Internal format of directory is incorrect
ILINS1	600770	Undefined operation code
ILINS2	600771	Undefined JSYS
ILINS3	600772	UUO simulation facility not available
CRLNX1	601000	Logical name is not defined
INLNX1	601001	Index is beyond end of logical name table
LNSTX1	601002	No such logical name
MLKBX1	601003	Lock facility already in use

MLKBX2	601004	Too many pages to be locked
MLKBX3	601005	Page is not available
MLKBX4	601006	Illegal to remove previous contents of user map
VBCX1	601007	Display data area not locked in core
RDTX1	601010	Invalid string pointer
GFKSX1	601011	Area too small to hold process structure
GTJIX1	601013	Invalid index
GTJIX2	601014	Invalid terminal line number
GTJIX3	601015	Invalid job number
IPCFX1	601016	Length of packet block cannot be less than 4
IPCFX2	601017	No message for this PID
IPCFX3	601020	Data too long for user's buffer
IPCFX4	601021	Receiver's PID invalid
IPCFX5	601022	Receiver's PID disabled
IPCFX6	601023	Send quota exceeded
IPCFX7	601024	Receiver quota exceeded
IPCFX8	601025	IPCF free space exhausted
IPCFX9	601026	Sender's PID invalid
IPCF10	601027	WHEEL capability required
IPCF11	601030	WHEEL or IPCF capability required
IPCF12	601031	No free PID's available
IPCF13	601032	PID quota exceeded
IPCF14	601033	No PID's available to this job
IPCF15	601034	No PID's available to this process
IPCF16	601035	Receive and message data modes do not match
IPCF17	601036	Not enough arguments
IPCF18	601037	Invalid MUTIL JSYS function
IPCF19	601040	No PID for [SYSTEM] INFO
IPCF20	601041	Invalid process handle
IPCF21	601042	Invalid job number
IPCF22	601043	Invalid software channel
IPCF23	601044	[SYSTEM] INFO already exists
IPCF24	601045	Invalid message size
IPCF25	601046	PID does not belong to this job
IPCF26	601047	PID does not belong to this process
IPCF27	601050	PID is not defined
IPCF28	601051	PID not accessible by this process
IPCF29	601052	PID already being used by another process
IPCF30	601053	Job is not logged in
GNJFX1	601054	No more files in this specification
ENQX1	601055	Invalid function code
ENQX2	601056	Level number too small
ENQX3	601057	Request and lock level numbers do not match
ENQX4	601060	Number of pool and lock resources do not match
ENQX5	601061	Lock already requested
ENQX6	601062	Requested locks are not all locked
ENQX7	601063	No ENQ on this lock
ENQX8	601064	Invalid access change requested
ENQX9	601065	Invalid number of blocks specified
ENQX10	601066	Invalid argument block length
ENQX11	601067	Invalid software interrupt channel number
ENQX12	601070	Invalid number of resources requested
ENQX13	601071	Indirect or indexed byte pointer not allowed
ENQX14	601072	Invalid byte size
ENQX15	601073	ENQ/DEQ capability required
ENQX16	601074	WHEEL or OPERATOR capability required
ENQX17	601075	Invalid JFN
ENQX18	601076	Quota exceeded
ENQX19	601077	String too long
ENQX20	601100	Locked JFN cannot be closed
ENQX21	601101	Invalid job number or job not logged in
IPCF31	601102	Invalid page number
IPCF32	601103	Page is not private
PMAPX3	601104	Illegal to move shared page into file

PMAPX4	601105	Illegal to move file page into process
PMAPX5	601106	Illegal to move special page into file
PMAPX6	601107	Disk quota exceeded
SNOPX1	601110	WHEEL or OPERATOR capability required
SNOPX2	601111	Invalid function code
SNOPX3	601112	.SNPLC function must be first
SNOPX4	601113	Only one .SNPLC function allowed
SNOPX5	601114	Invalid page number
SNOPX6	601115	Invalid number of pages to lock
SNOPX7	601116	Illegal to define breakpoints after inserting them
SNOPX8	601117	Breakpoints is not set on instruction
SNOPX9	601120	No more breakpoints allowed
SNOP10	601121	Breakpoints already inserted
SNOP11	601122	Breakpoints not inserted
SNOP12	601123	Invalid format for program name symbol
SNOP13	601124	No such program name symbol
SNOP14	601125	No such symbol
SNOP15	601126	Not enough free pages for snooping
SNOP16	601127	Multiply defined symbol
IPCF33	601130	Invalid index into system PID table
SNOP17	601131	Breakpoint already defined
OPNX23	601132	Disk quota exceeded
GJFX37	601133	Input deleted
CRLNX2	601134	WHEEl or OPERATOR capability required
INLNX2	601135	Invalid function code
LNSTX2	601136	Invalid function code
ALCX1	601137	Invalid function code
ALCX2	601140	WHEEL or OPERATOR capability required
ALCX3	601141	Device is non-assignable
ALCX4	601142	Invalid job number
ALCX5	601143	Device not available
SPLX1	601144	Invalid function code
SPLX2	601145	Invalid argument block length
SPLX3	601146	Invalid device designator
SPLX4	601147	WHEEL or OPERATOR capability required
SPLX5	601150	Illegal to specify 0 as generation number for first file
CLSX3	601151	File still mapped
CRLNX3	601152	Invalid function code
ALCX6	601153	Device assigned to user job, but will be given to allocator when released
CKAX1	601154	Not enough arguments
CKAX2	601155	Invalid directory number
CKAX3	601156	Invalid access code
TIMX1	601157	Invalid function code
TIMX2	601160	Invalid process handle
TIMX3	601161	Time limit already set
TIMX4	601162	Illegal to clear time limit
SNOP18	601163	Data page is not private or copy-on-write
GJFX38	601164	File not found because output-only device was specified
GJFX39	601165	Logical name loop detected
CRDIX8	601166	Invalid directory number specified
CRDIX9	601167	Invalid format directory file encountered
CRDI10	601170	Maximum directory number exceeded; index table needs expanding
DELDX1	601171	WHEEL or OPERATOR capability required
DELDX2	601172	Invalid directory number
GACTX3	601173	Bad block type in directory
DIAGX1	601174	Invalid function
DIAGX2	601175	Device is not assigned
DIAGX3	601176	Too few arguments
DIAGX4	601177	Invalid device type
DIAGX5	601200	WHEEL, OPERATOR, or MAINTENANCE capability required
DIAGX6	601201	Invalid channel command list

DIAGX7	601202	Illegal to do I/O across page boundary
DIAGX8	601203	No such device
DIAGX9	601204	Unit does not exist
DIAG10	601205	TU16 does not exist
SYEX1	601206	Unreasonable SYSERR block size
SYEX2	601207	No buffer space available for SYSERR
MTOX1	601210	Invalid function
IOX7	601211	No room in Job Storage Block
IOX8	601212	Monitor internal error
MTOX5	601213	Invalid hardware data mode for magnetic tape
DUMPX5	601214	No-wait dump mode not supported for this device
DUMPX6	601215	Dump mode not supported for this device
IOX9	601216	Function legal for sequential write only
CLSX4	601217	Device still active
MTOX2	601220	Record size was not set before I/O was done
MTOX3	601221	Function not legal in dump mode
MTOX4	601222	Invalid record size
MTOX6	601223	Invalid magnetic tape density
OPNX25	601224	Device is write locked
GJFX40	601225	Undefined attribute in file specification
MTOX7	601226	WHEEL or OPERATOR capability required
LOUTX3	601227	WHEEL or OPERATOR capability required
LOUTX4	601230	LOG capability required
CAPX2	601231	WHEEL or OPERATOR capability required
SSAVX3	601232	No job storage available
SSAVX4	601233	Directory area of EXE file is more than one page
TDELX1	601234	Table is empty
TADDX1	601235	Table is full
TADDX2	601236	Entry is already in table
TLUKX1	601237	Internal format of table is incorrect
IOX10	601240	Record is longer than user requested
CNDIX2	601241	WHEEL or OPERATOR capability required
CNDIX4	601242	Invalid job number
CNDIX6	601243	Job is not logged in
SJBX1	601244	Invalid function code
SJBX2	601245	Invalid magnetic tape density
SJBX3	601246	Invalid magnetic tape data mode
TMONX1	601247	Invalid TMON function
SMONX2	601250	Invalid SMON function
SJBX4	601251	No such job
SJBX5	601252	Job is not logged in
SJBX6	601253	WHEEL or OPERATOR capability required
GTJIX4	601254	No such job
ILINS4	601255	UUO simulation is disabled
ILINS5	601256	DMS facility is not available
COMNX1	601257	Invalid COMND function code
COMNX2	601260	Field too long for internal buffer
COMNX3	601261	Command too long for internal buffer
COMNX4	601262	Invalid character in input
PRAX1	601263	Invalid PRARG function code
PRAX2	601264	No room in monitor data base for argument block
COMNX5	601265	Invalid string pointer argument
COMNX6	601266	Problem in indirect file
COMNX7	601267	Error in command
PRAX3	601270	PRARG argument block too large
CKAX4	601271	File is not on disk
GACCX1	601272	Invalid job number
GACCX2	601273	No such job
MTOX8	601274	Argument block too long
DBRKX1	601275	No breaks in progress
SJPRX1	601276	Nonexistent job
GJFX41	601277	File name must not exceed 6 characters
GJFX42	601300	File type must not exceed 3 characters
GACCX3	601301	Confidential Information Access Capability required

TIMEX2	601302	Downtime cannot be more than 7 days in the future
DELFX2	601303	File cannot be deleted because it is currently open
DELFX3	601304	System scratch area depleted; file not deleted
DELFX4	601305	Directory symbol table could not be rebuilt
DELFX5	601306	Directory symbol table needs rebuilding
DELFX6	601307	Internal format of directory is incorrect
DELFX7	601310	FDB formatted incorrectly; file not deleted
DELFX8	601311	FDB not found; file not deleted
FRKHX7	601312	Process page cannot exceed 777
DIRX1	601313	Invalid directory number
DIRX2	601314	Not enough internal system resources to open directory file
DIRX3	601315	Internal format of directory is incorrect
UFPGX1	601316	File is not open for write
LNGFX1	601317	Page table does not eXist and file not open for write
IPCF34	601320	Cannot receive into an existing page
COMNX8	601321	Number base out of range 2-10
MTOX9	601322	Output still pending
MTOX10	601323	VFU or RAM file cannot be OPENed
MTOX11	601324	Data too large for buffers
MTOX12	601325	Input error or not all data read
MTOX13	601326	Argument block too small
MTOX14	601327	Invalid PSI channel
SAVX1	601330	Illegal to save files on this device
MTOX15	601331	Device does not have Direct Access (programmable) VFU
LPINX1	601333	Invalid unit number
LPINX2	601334	WHEEL or OPERATOR capability required
LPINX3	601335	Illegal to load RAM or VFU while device is OPEN
MTOX17	601336	Device is off-line
LGINX6	601337	No more jobs available for logging-in
DESX9	601340	Illegal operation for this device

INDEX

Monitor Calls
User's Guide
DEC-20-OMUGA-A-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you require a written reply, please check here. ☐

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754