IBM Advanced Interactive Executive
for the Personal System/2
(AIX PS/2)
Interface Library Reference
Version 1.1

Document Number SC23-2051-0

IBM Advanced Interactive Executive
for the Personal System/2
(AIX PS/2)

Interface Library Reference

Version 1.1

Document Number SC23-2051-0

Edition Notice

First Edition (March 1989)

The information in this manual applies to IBM AIX VS Pascal (Program Number 5713-AEZ) and IBM AIX VS FORTRAN (Program Number 5713-AFA) for use with Version 1.1 of the IBM AIX PS/2 Operating System (Program Number 5713-AEQ), and it applies to all releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Products are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM dealer or your IBM marketing representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 35RB, 36 Apple Ridge Road, Danbury, Connecticut 06810. IBM may use or distribute, in any way it believes appropriate and without incurring any obligation to the sender, whatever information it receives in this manner.

About this Book

This reference book contains information about the library of system calls available with IBM AIX VS Pascal and IBM AIX VS FORTRAN as implemented for use with the IBM AIX PS/2 Operating System.

Subtopics

Who Should Read this Book

This book is intended for programmers wishing to use AIX system subroutines in their own VS Pascal or VS FORTRAN application programs. It assumes familiarity with Pascal or FORTRAN and with either AIX or UNIX ( ) System V commands and system calls. For AIX PS/2 publications that deal with VS Pascal, VS FORTRAN, and AIX, see "Related Publications" on page PREFACE.4.

Note: Neither VS FORTRAN nor VS Pascal supports multibyte characters. Programs written in these languages can only process single-byte characters like ASCII.

It is recommended that such programs not be used with AIX system Release 1.3.

( ) UNIX is a registered trademark of UNIX System Laboratories, Inc. in the U.S.A. and other countries.

How to Use This Book

The information in this reference is divided into two sections and six appendixes. For an overview of the book and of the major functions available through the interface library, read the first section--Introduction to the Interface Library--which begins on page 1.0. This section also contains additional information of interest to programmers using the library.

The second section--System Calls--which begins on page 2.0, contains the bulk of the reference material in this book. Most of the system calls in the interface library are described individually in separate subsections. In some instances, however, two or more related system calls are described in a single subsection. Subsections are alphabetically ordered by system-call name. All system calls are listed in the Table of Contents (individually or grouped) and in the Index. In addition, each descriptive subsection carries as a running title the name(s) of the system call(s) discussed in that section.

The appendixes contain information about error codes and messages, Pascal definitions and declarations, and two important system subroutines: **ftok** and **perror**.

Highlighting

This book uses several typographic conventions in its descriptions of the various system calls.

System-call names appear in the descriptive text in **UPPERCASE BOLDFACE**.

Program variables appear in the descriptive text in **lowercase italics**.

Constants appear in the descriptive text in UPPERCASE LETTERS

The syntax descriptions near the beginning of each subsection appea in a monospace typeface that suggests a **computer printout**.

The same **"example"** typeface is used to present the example programs at the end of each subsection.

In the brief descriptions preceding each example program, doubl quotation marks around one or more characters (for example, "s1") identifies a variable name arbitrarily picked for purposes of the particular example.

In the few direct references to an AIX system subroutine, the name o the subroutine appears in **lowercase boldface**.

Related Publications

You may want to refer to the following IBM AIX publications for additional information:

AIX Operating System Commands Reference, SC23-2025, lists and describes the AIX Operating System commands.

AIX Operating System Programming Tools and Interfaces, SC23-2029, describes the programming environment of the AIX Operating System and includes information about the use of operating system tools to develop, compile, and debug programs.

AIX Operating System Technical Reference, Volumes 1 and 2 (SC23-2032 and SC23-2033) describes the system calls and subroutines a programmer would use to write application programs. This book also provides information about the AIX Operating System file system, special files, miscellaneous files, and the writing of device drivers.

VS Pascal User's Guide, SC23-2053, describes how to develop and execute VS Pascal programs. This book also describes the procedures for compiling and executing programs that contain sections of code written in VS FORTRAN and C.

VS Pascal Reference, SC23-2054, describes the statements, data structures, and other features of the Pascal programming

VS FORTRAN User's Guide, SC23-2049, describes how to develop and execute VS FORTRAN programs. This book also describes the procedures for compiling and executing programs that contain sections of code written in VS Pascal and C.

VS FORTRAN Reference, SC23-2050, describes the statements, data structures, and other features of the FORTRAN 77 programming language.

Table of Contents

1.0 Introduction to the Interface Library

Subtopics

1.1 What It Is

The VS Language/Operating System Interface Library is an application-program interface that provides access to the system calls of the AIX Operating System from programs written either in AIX VS Pascal or in AIX VS FORTRAN. These system calls, which are a part of the AIX Operating System, invoke a variety of system routines whose functions include file maintenance, input and output, and interprocess (1) communication.

Note: Neither VS FORTRAN nor VS Pascal supports multibyte characters. Programs written in these languages can only process single-byte characters like ASCII.

It is recommended that such programs not be used with AIX Release 1.3. Nevertheless, this manual can provide valuable information on AIX systems calls for someone who wants to use them in C programs.

Information on using system calls in a C program can be found in:

AIX Technical Reference

AIX Programming Tools and Interfaces

(1) As used in this manual, the term *process* refers to a program running under the AIX Operating System, together with the environment it runs in.

1.2 What You Need

The AIX Operating System installed on your PS/

AIX PS/2 VS Pascal or AIX PS/2 VS FORTRAN installed according to th directions given in the Program Directory that accompanied the language.

1.3 What It Does

The VS Language/Operating System Interface Library makes it easy to use the AIX system calls directly from programs written in VS Pascal or VS FORTRAN by changing the calls's associated data structures, naming conventions, and data types to conform to those required by the system. The Interface Library takes care of many of the details of interfacing to the actual system calls without the need for C-language or assembly-language "wrappers."

1.4 How This Manual is Organized

The system-call descriptions are listed alphabetically by system-call name, beginning on page 2.1. Information on a particular call can be found by looking for the call name in the Table of Contents. (Because some sections describe more than one system call, the listing in the Table of Contents is not perfectly alphabetical, though all of the calls are listed.) Individual calls can also be found by consulting the Index, either under the name of the individual system call or under one of the following functional categories.

- process contro
- process identificatio
- process trackin
- input and outpu
- file maintenanc
- signal
- semaphore
- message
- shared memor
- socket
- system utilitie

The calls are grouped by functional category as follows:

Subtopics

1.4.1 Process Control

BRK, SBRK (change data-segment space allocation)

EXECL, EXECLE, EXECLP (execute a program)

EXECV, EXECVE, EXECVP (execute a program)

EXIT, _EXIT (terminate a process)

FORK (create a new process)

NICE (set a process priority)

PLOCK (lock or unlock a process, text, or data)

WAIT, WAIT3 (wait for a child process to terminate)

1.4.2 Process Identification

GETDTABLESIZE (get size of process-descriptor table)

GETGROUPS (get a group access list)

GETHOSTID (get the host-machine identifier)

GETHOSTNAME (get the host-machine name)

GETLOCAL (get the alias for <LOCAL>)

GETPGRP, GETPID, GETPPID (get a process-group or process identifier)

GETUID, GETGID, GETEUID, GETEGID (get a user or a group identifier)

SETGROUPS (set a group access list)

SETHOSTID (set the host-machine identifier)

SETHOSTNAME (set the host-machine name)

SETLOCAL (set the alias for <LOCAL>)

SETPGRP, SETPGID (set a process group ID)

SETUID, SETGID (set user or group identifiers)

ULIMIT (get and set process limits)

USRINFO (get and set user information)

1.4.3 Process Tracking

ACCT (turn accounting process on or off)

PROFIL (generate a time profile)

PTRACE (trace the execution of a child process)

TIMES (get the processing times)

1.4.4 Input-Output

ACCESS (check file-access permissions)

CLOSE (close a file)

CREAT (create a new file)

DUP, DUP2 (generate a second file-descriptor)

FABORT (cancel changes to a file)

FCLEAR (clear space in a file)

FSYNC, FCOMMIT (write to permanent storage)

FTRUNCATE (truncate a file)

IOCTL (control the input and output of a device)

LOCKF (lock or unlock a region of a file)

LSEEK (set a read or write pointer)

OPEN (open a file for reading or writing)

PIPE (create an interprocess channel)

READ, READX (read from a file)

READV (read output into multiple buffers)

SELECT (check I/O status of descriptors and message queues)

WRITE, WRITEX (write to a file)

WRITEV (write input from multiple buffers)

Note: READV and WRITEV are not available in FORTAN.

1.4.5 File Maintenance

CHDIR (change the default directory)

CHHIDDEN (convert a directory)

CHMOD (change file-access permissions)

CHOWN, CHOWNX (change file ownership)

CHROOT (change a root directory)

FCNTL (control an open-file descriptor)

FABORT (cancel a change to a file)

LINK (link to a file)

MKDIR (create a directory)

MKNOD (create a directory or a special file)

MOUNT, UMOUNT (mount or unmount a file system)

READLINK (read the value of a symbolic link)

RENAME (rename a directory or file)

RMDIR (remove a directory)

STATX, FSTATX, STAT, FSTAT, LSTAT, FULLSTAT, FFULLSTAT (return the status of a file)

SYMLINK (create a symbolic link to a file)

SYNC (update a file system)

UMASK (set and get a file-creation-mode mask)

UNLINK (delete a directory entry)

USTAT (get file-system information)

UTIME (set the file times)

UTIMES (set the file times)

1.4.6 Signals

ALARM (schedule an alarm signal)

KILL, KILLPG (send a signal to a process or process group)

PAUSE (wait for a signal)

SIGACTION (specify a response to a signal)

SIGBLOCK (block one or more signals)

SIGNAL (specify the process response to a signal)

SIGPAUSE (release a blocked signal and wait for an interrupt)

SIGPROCMASK (set the signal mask of the current process)

SIGSETMASK (set the signal mask of the current process)

SIGSTACK (define an alternate stack)

SIGSUSPEND (reset the signal mask and wait for an interrupt)

SIGVEC (select signal-handling facilities)

Note: SIGACTION, SIGSTACK, and SIGVEC are not available in FORTRAN.

1.4.7 Semaphores

SEMCTL (invoke semaphore-control operations)

SEMGET (get or create a semaphore-set identifier)

SEMOP (perform semaphore operations)

1.4.8 Messages

MSGCTL (invoke message-control operations)

MSGGET (get or create a message queue)

MSGRCV, MSGXRCV (read and store a message)

MSGSND (send a message to a queue)

RECV, RECVMSG, RECVFROM (receive a message from a socket)

SEND, SENDTO, SENDMSG (send a message from a socket)

1.4.9 Shared Memory

SHMAT (attach a shared-memory segment or mapped file)

SHMCTL (invoke shared-memory-control operations)

SHMDT (detach a shared-memory or mapped-file segment)

SHMGET (get a shared-memory-segment identifier)

1.4.10 Sockets

ACCEPT (accept a connection to a socket)

BIND (assign a name to a socket)

CONNECT (make a connection between two sockets)

GETPEERNAME (get the name of a connected socket)

GETSOCKNAME (get the name of a connected socket)

GETSOCKOPT (get the socket options)

LISTEN ("listen" for a connection to a socket)

SETSOCKOPT (set a socket's options)

SHUTDOWN (disable sending or receiving functions)

SOCKET (create a socket)

SOCKETPAIR (create a pair of connected sockets)

1.4.11 System Utilities

ADJTIME (synchronize the system clock)

DISCLAIM ("disclaim" the content of an area of memory)

GETITIMER (get the value of an internal timer)

GETTIMEOFDAY (get the current time)

GETXVERS (return the UNIX version string)

REBOOT (restart the operating system)

SETITIMER (set the value of an internal timer)

SETTIMEOFDAY (set the current time)

SETXVERS (set the UNIX version string)

STIME (set the system clock)

TIME (get the system time)

UNAME, UNAMEX (get the name of the current operating system)

1.5 The ftok System Subroutine

The Interface Library gives the programmer access to AIX Operating System calls from VS Pascal or VS FORTRAN. An exception is **ftok**, an AIX Operating System subroutine that is often used by Pascal procedures and FORTRAN subroutines of the kind shown in the program examples elsewhere in this manual. For your convenience, therefore, a description of the **ftok** subroutine is given in Appendix E.

1.6 Using the Interface Library with VS Pascal

Before you can use the Interface Library with a VS Pascal program, you must do two things:

1. Declare the constants, data types, and external functions that will be used by the program.

   For your convenience, these declarations are provided in include files (see Appendixes B, C, D). The type declarations include those for the parameters and return values that appear in the descriptions of the calls. For purposes of illustration, predefined constants, types, and functions listed in the include files are also used in the programming examples.

2. Link the Interface Library to the program, using the **cc** utility.

Once these requirements are satisfied, you can use any number of AIX system calls in your program. For information concerning these calls, see "Related Publications" on page PREFACE.4.

Subtopics

1.6.1 Declarations

The Interface Library provides three files that can be used for making Pascal declarations:

1. constants:

   /usr/include/ailpconsts.inc

2. data types:

   /usr/include/ailtypes.inc

3. external functions:

   /usr/include/aildefs.inc

To include any of these files in a VS Pascal program, use the %include compiler directive (see *VS Pascal User's Guide*). For the contents of the include files, see Appendixes B, C, and D. The following program illustrates how these files are used.

```
program aildemo;

const
  %include /usr/include/ailpconsts.inc

type
  %include /usr/include/ailtypes.inc
  usrary = packed array[1..INFSIZ] of char;
  usrptr = @usrary;

%include /usr/include/aildefs.inc

function p_usrinf (cmd : integer; buf : usrptr;
                  count : integer) : integer; external;

procedure call1;
var
  red : unam;
  blue : integer;

begin
  blue := p_uname (red);
  writeln (red.sysname)
end;

procedure call2;
var
  blue, red : integer;
  yellow : usrptr;

begin
   new (yellow);
  blue := p_usrinf (GETINF, yellow, INFSIZ);
  for red := 1 to blue do
    write (yellow@[red]);
  writeln
end;
```

```
begin
  call1;
  call2
end.
```

1.6.2 Linkage

You must link the Interface Library (/lib/libvspil.a) to your program. For example, to compile the aildemo program (assume the file name is aildemo.p), you would type the following command:

```
cc -o aildemo aildemo.p -lm -lvspil -lvssys -lc
```

1.7 Using the Interface Library with VS FORTRAN

Before you can use the Interface Library with a VS FORTRAN program, you must do two things:

1. First, declare the constants that will be used by the program, so that it can be compiled.

2. Second, link the Interface Library to the program, using the **cc** utility.

Once these requirements are satisfied, you can use any number of AIX system calls in your program. For information concerning these calls, see "Related Publications" on page PREFACE.4.

Subtopics

1.7.1 Declarations

The Interface Library provides one file that can be used for making FORTRAN declarations:

(/usr/include/ailfconsts.inc)

To include this file in your program, use the INCLUDE compiler directive (see *VS FORTRAN User's Guide*). For a description of the contents of the file, see Appendix B.

The program on the next page illustrates how this file is used.

```
      PROGRAM AILDEMO
      CALL FIRST
      CALL SECOND
      END

      SUBROUTINE FIRST
      CHARACTER*9 RED(5)
      INTEGER BLUE, UNAME
      BLUE = UNAME (RED)
      PRINT *, RED(1)
      END

      SUBROUTINE SECOND
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER RED, BLUE, USRINF
      CHARACTER*INFSIZ YELLOW
      BLUE = USRINFO (GETINF, YELLOW, INFSIZ)
      WRITE  *, YELLOW
      END
```

1.7.2 Linkage

You must link the Interface Library (/lib/libvsfil.a) to your program. For example, to compile the AILDEMO program (assume the file name is aildemo.f), you would type the following command:

```
cc -o aildemo aildemo.f -lm -lvsfil -lvsfor -lvssys -lc
```

1.8 Return Values, Error Codes, and Error Messages

Most of the AIX system calls from Pascal and FORTRAN return a value. See the individual system-call descriptions for details regarding these values.

A return value of -1 indicates that an error has occurred. When a system call generates an error, an error code is set in the external variable **errno**. Two routines are available for retrieving this value:

1. A call to the p_ercode function in Pascal or the ERCODE subroutine in FORTRAN returns the value of the external variable **errno**.

2. A call to the **perror** system subroutine prints out an error message (for a description of **perror**, see Appendix F).

2.0 System Calls

Each system-call description in this section summarizes the function of the system routine being called, the syntax of the call, its parameters, and any return values. It also contains examples of a call made from both VS Pascal and VS FORTRAN programs.

Each description contains the first five subsections listed below, and occasionally the sixth.

Description briefly describes the function of the system routine that is being called.

Syntax shows the correct coding required for making a given system call from Pascal and FORTRAN programs.

Parameters briefly defines the function and type (for example, integer) of any parameters required by a given system call.

Return Values briefly describes the value returned by a given system call when it has been successfully completed and when it has failed.

Examples contains short examples of Pascal and FORTRAN coding that invoke the system call or calls described in the section.

Notes provides, where it is appropriate, additional information of importance to the programmer. ("Notes" also appear occasionally in other parts of a descriptive section, but they are not then displayed as subsection headings, though they are printed in bold-faced type.

Subtopics

2.1 ACCEPT accept a connection to a socket

Description

The **ACCEPT** system call extracts the first connection from the queue of pending connections, creates a new connection with the same properties as **s**, and allocates a new file descriptor to that socket.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_accept (s, addr, addrlen)                                      ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FACCEPT (S, ADDR1, ADDR2, ADDRLEN)                               ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

s
is the descriptor of a socket that was created with a **SOCKET** system call, was bound to an address with a **BIND** system call, and is "listening" for connections after a **LISTEN** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

addr, ADDR1, ADDR2
are result parameters that receive the address of the connecting entity as it is known to the communications layer. The exact format of **addr** is determined by the *domain* in which the communication occurs.

In Pascal, **addr** is of type sockaddrptr (declared in the include file ailtypes.inc).

In FORTRAN, **addr1** is of type INTEGER and corresponds to sockaddr.sa_family in Pascal.

In FORTRAN, **addr2** is of type CHARACTER*14 and corresponds to sockaddr.sa_data in Pascal.

addrlen
initially contains the amount of space pointed to by the "addr" parameters. On return, it contains the actual length of the address returned.

In Pascal, **addrlen** is of type integer.

In FORTRAN, **addrlen** is of type INTEGER.

Return Values

The value returned upon successful completion of the call is the nonnegative socket-descriptor of the accepted socket. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **ACCEPT** system routine, which in these examples receives in the variable "green", the nonnegative socket-descriptor of the accepted socket.

Pascal

```
procedure accept1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  addrlen, s, green : integer;
  addr : sockaddrptr;

%include /usr/include/aildefs.inc

begin
  new (addr);
  s := p_socket(PF_UNIX, SOCK_STREAM, 0);
  if (s = -1) then showerror;
  addrlen := 20;
  green := p_accept (s, addr, addrlen);
  writeln ('Accept returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE ACCEPT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FACCEPT, FSOCKET, ADDR1, S, GREEN
      CHARACTER*14 ADDR2
      S = FSOCKET (PFUNIX, SKSTRM, 0)
      IF (S .EQ. -1) CALL ERRORS

      GREEN = FACCEPT (S, ADDR1, ADDR2, 20)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.2 ACCESS check file accessibility

Description

The **ACCESS** system call checks a file's accessibility against a specified access mode.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_access (path, amode);                                          ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FACCESS (PATH, AMODE)                                            ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

path
is the name of the file to be checked.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

amode
is the access mode of the file specified by **path**. The parameter value is that of one of the parameter options or is constructed from two or more of those options by logical ORing. The options are defined as constants in the Pascal and FORTRAN constants include files.

F_OK searches for a file

X_OK tests for execute permission

W_OK tests for write permission

R_OK tests for read permission

Note: In FORTRAN, the underscore is omitted (for example, "FOK").

Note: Specifying access mode 0 (zero) tests whether the directories leading to a file can be searched and whether the file exists.

In Pascal, **amode** is a variable or constant of type integer.

In FORTRAN, **amode** is a variable or constant of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **ACCESS** system routine. The accessibility of the file specified by **path** ("blue") is tested. The specified file is found and tested for execution, write, and read permissions as specified by the ORed value 7, defined in the variable "red".

Pascal

```
procedure access1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green, red : integer;
  blue : st80;

%include /usr/include/aildefs.inc

begin
  red := X_OK + W_OK + R_OK;
  blue := '/tmp/myfile';
  green := p_access (blue, red);
  writeln (green);
end;
```

FORTRAN

```
      SUBROUTINE ACCESS1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FACCESS, GREEN, RED
      CHARACTER*80 BLUE
      RED = XOK + WOK + ROK;
      BLUE = '/tmp/myfile '
      GREEN = FACCESS (BLUE, RED)
      PRINT *, GREEN
      END
```

2.3 ACCT turn process accounting on or off

Description

The **ACCT** call writes records in a specified "accounting file" whenever a process is terminated. Records of the terminated process are appended to the accounting file.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_acct (path);                                                     ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FACCT (PATH)                                                       ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

path

is the name of the file to which all accounting records are written. Passing the file name as an argument in the call activates the accounting function. Passing a null string turns the accounting function off.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **ACCT** routine. The accounting function is turned on and the records are appended to the files specified by **path**. The return value of the call is in "blue".

Pascal

```
  procedure acct1;

  const
    %include /usr/include/ailpconsts.inc
```

```
type
  %include /usr/include/ailtypes.inc
var
  blue : integer;
  red : st80;

%include /usr/include/aildefs.inc

begin
  red := '/tmp/acct';
  blue := p_acct (red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE ACCT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FACCT, BLUE
      CHARACTER*80 RED
      RED = '/tmp/acct '
      BLUE = FACCT (RED)
      PRINT *, BLUE
      END
```

2.4 ADJTIME synchronize the system clock

Description
The **ADJTIME** system call makes small adjustments to the system time (as returned by the **GETTIMEOFDAY** call), advancing or slowing it by a specified amount.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                      ¦
¦   p_adjtime (delta, olddelta);                                       ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                      ¦
¦   FADJTIME (DELTA, OLDDELTA)                                         ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

delta
specifies the amount of time (in seconds and microseconds) by which the system time is to be adjusted. If the value specified is negative, the system clock is slowed down by advancing the time at less than the normal rate until synchronization is achieved.

In Pascal, **delta** is of type timeval.

In FORTRAN **delta** is an array containing two elements of type INTEGER.

olddelta
returns the number of seconds and microseconds to adjust the time from the earlier call.

In Pascal, **olddelta** is of type timeval.

In FORTRAN **olddelta** is an array containing two elements of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **ADJTIME** system routine.

Pascal

```
procedure adjtime1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  delta, olddelta : timeval;

%include /usr/include/aildefs.inc

begin
  delta.tv_sec := 20;
  delta.tv_usec := 30;
  green := p_adjtime (delta, olddelta);
    writeln ('Adjtime returned: ', green: 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE ADJTIME1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FADJTIME, DELTA(2), OLDDELTA(2), GREEN
      DELTA(1) = 20
      DELTA(2) = 30
      GREEN = FADJTIME (DELTA, OLDDELTA)
      IF (GREEN .EQ. -1) THEN
        PRINT *, 'ADJTIME: ERROR'
        CALL ERRORS
      ELSE
        PRINT *, 'ADJTIME: OK'
      ENDIF
      END
```

2.5 ALARM schedule an alarm signal

Description
The **ALARM** system call sends a **SIGALARM** signal to the calling process in a specified number of seconds. In effect, it sets an "alarm" clock. Unless caught or ignored, the signal terminates the calling process.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_alarm (sec);                                                     ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+

+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FALARM (SEC)                                                       ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

sec
is the number of seconds before the alarm signal is sent to the calling process (see **Notes** at the end of this section).

In Pascal, **sec** is of type usign.

In FORTRAN, **sec** is of type INTEGER.

Return Values
The return value of this call is the amount of clock time remaining from the previous **ALARM** call. The return value is the amount of time that previously remained on the alarm clock of the calling process before it is reset to the new time (see **Notes**).

In Pascal, the return value is of type usign

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **ALARM** system routine, which in these examples instructs the alarm clock to signal the calling process after 100 seconds have elapsed.

Pascal

```
procedure alarm1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : usign;
```

```
  %include /usr/include/aildefs.inc

  begin
    red := 100;
    blue := p_alarm (red);
    writeln (blue);
  end;
```

FORTRAN

```
        SUBROUTINE ALARM1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FALARM, BLUE, RED
        RED = 100
        BLUE = FALARM (RED)
        PRINT *, BLUE
        END
```

Notes

Because Pascal and FORTRAN lack the facilities for handling unsigned 4-byte integers, the programmer must convert parameter values of type usign that fall in the range

2 147 483 648 through 4 294 067 295

To use a parameter value in this range, subtract 4 294 067 296 from that value before issuing the call (the result will always be negative).

2.6 BIND bind a name to a socket

Description
The **BIND** system call assigns a name to a socket.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_bind (s, name, namelen)                                        ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FBIND (S, FAMILY, NAME, NAMELEN)                                 ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

s
is the descriptor of a socket that was created with a **SOCKET** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER, corresponding to sockaddr.sa_familyt in Pascal.

name
is a unique name to be assigned to the socket.

In Pascal, **name** is of type sockaddrptr (declared in the include file ailtypes.inc).

In FORTRAN, **name** is of type CHARACTER*14 and corresponds to sockaddr.sa_data in Pascal.

family
is the address family specified in the **SOCKET** system call.

Used only in FORTRAN, **family** is of type INTEGER and corresponds to sockaddr.sa_family in Pascal.

namelen
is the length of the **name** parameter. On return, it contains the actual length of the address returned.

In Pascal, **namelen** is of type integer.

In FORTRAN, **namelen** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **BIND** system routine, which in these examples assigns the name 'socket' to socket descriptor "s".

Pascal

```
procedure bind1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  namelen, s, green : integer;
  name : sockaddrptr;

%include /usr/include/aildefs.inc

begin
  new (name);
  s := p_socket (PF_UNIX, SOCK_STREAM, 0);
  if (s = -1) then showerror;
  name^.sa_data := 'socket';
  name^.sa_family := PF_UNIX;
  namelen := 10;
  green := p_bind (s, name, namelen);
  writeln ('Bind returned: ', green : 2);
  if (green = -1) then showerror;
  green := p_unlink(name);
end;
```

FORTRAN

```
      SUBROUTINE BIND1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FBIND, FSOCKET, FUNLINK, S, GREEN
      CHARACTER*14 NAME
      S = FSOCKET (PFUNIX, SKSTRM, 0)
      IF (S .EQ. -1) CALL ERRORS
      NAME = 'SOCKET '
      GREEN = FBIND (S, PFUNIX, NAME, 10)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      GREEN = FUNLINK (NAME)
      END
```

Notes

Sockets in the AF_UNIX address family create a name in the file system name space that must be deleted by the calling process (using **UNLINK**) when it is no longer needed.

2.7 BRK, SBRK change data-segment space allocation

Description
The **BRK** and **SBRK** system calls dynamically change the amount of space allocated to the data segment of the calling process.

The **BRK** system call sets the breakpoint value to that specified in the call and changes the space allocation accordingly.

The **SBRK** system call adds to the breakpoint value the number of bytes specified in the call and changes the space allocation accordingly.

Syntax

Pascal

```
p_brk (endds);

p_sbrk (incr);
```

FORTRAN

```
FBRK (ENDDS)

FSBRK (INCR)
```

Parameters

endds
is used only with the **BRK** call. It specifies the new breakpoint that is to be set.

In Pascal, **endds** is of type integer.

In FORTRAN, **endds** is of type INTEGER.

incr
is used only with the **SBRK** call. It specifies the number of bytes to be added to or subtracted from the space allocated to the program data segment.

In Pascal, **incr** is of type integer.

In FORTRAN, **incr** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the **BRK** call.

The previous break value is returned upon successful completion of the **SBRK** call.

The value -1 is returned and an error code set in **errno** if either call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow issue an **SBRK** system call to add 1000 bytes to the data segment of the calling program. The return value is in the variable "blue".

Pascal

```
procedure sbrk1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;

%include /usr/include/aildefs.inc

begin
  red := 1000;
  blue := p_sbrk (red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE SBRK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSBRK, BLUE, RED
      RED = 1000
      BLUE = FSBRK (RED)
      PRINT *, BLUE
      END
```

2.8 CHDIR change the current directory

Description

The **CHDIR** system call replaces the current working directory with the directory specified in the call. The current working directory is the starting point for searches when "/" is not specified.

Syntax

```
+--- Pascal --------------------------------------------------------+
¦                                                                   ¦
¦   p_chdir (path);                                                 ¦
¦                                                                   ¦
¦                                                                   ¦
+-------------------------------------------------------------------+
```

```
+--- FORTRAN -------------------------------------------------------+
¦                                                                   ¦
¦   FCHDIR (PATH)                                                   ¦
¦                                                                   ¦
¦                                                                   ¦
+-------------------------------------------------------------------+
```

Parameters

path
is the name of the directory that becomes the current working directory when the call is issued. Assigning "dot dot" (..) to this variable specifies the parent of the current directory.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values

The value 0 is returned when the directory is changed. The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **CHDIR** system routine. The directory specified in the call is /usr/games, which becomes the current working directory. The return value of the call is in the variable "folio". When the calling program terminates, the directory from which that program was executed once again becomes the current working directory.

Pascal

```
  procedure chdir1;

  const
    %include /usr/include/ailpconsts.inc
  type
```

```
  %include /usr/include/ailtypes.inc
var
  folio : integer;
  red : st80;

%include /usr/include/aildefs.inc

begin
  red := '/usr/games';
  folio := p_chdir (red);
  writeln (folio);
end;
```

FORTRAN

```
      SUBROUTINE CHDIR1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER  FCHDIR, FOLIO
      CHARACTER*80 RED
      RED = '/usr/games '
      FOLIO = FCHDIR (RED)
      PRINT *, FOLIO
      END
```

2.9 CHHIDDEN convert a hidden or normal directory

Description

The **CHHIDDEN** system call allows a super-user to convert a normal directory to a hidden one and vice versa.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ----------------------------------------------------------+
|                                                                      |
|   p_chhidden (dirname, hideflag);                                    |
|                                                                      |
|                                                                      |
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
|                                                                      |
|   FCHHIDDEN (DIRNAME, HIDEFLAG);                                     |
|                                                                      |
|                                                                      |
+----------------------------------------------------------------------+
```

Parameters

dirname
is the name of the directory to be converted.

In Pascal, **dirname** is a string variable or constant of type st80.

In FORTRAN, **dirname** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

hideflag
determines the "direction" of the conversion. A nonzero value converts a normal directory to a hidden one. The value 0 converts a hidden directory to a normal one.

In Pascal, **hideflag** is of type integer.

In FORTRAN, **hideflag** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **CHHIDDEN** system routine, which in these examples makes the directory /bushel/light/hide a hidden directory (by adding an '@' at the end of the directory name). Upon successful completion of the system call, the directory is made "unhidden" by calling **CHHIDDEN** again, with **hideflag** set to 0.

Pascal

```
procedure chhidden1;

const
  %include/usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
  p1 : st80;

%include /usr/include/aildefs.inc

begin
  p1 := '/bushel/light/hide';
  green := p_mkdir (p1, 128);
  green := p_chhidden (p1, 5);
  writeln ('Chhidden returned: ', green : 2);
  if (green = -1 ) then showerror;
  green := p_chhidden (p1, 0);
  end;
```

FORTRAN

```
      SUBROUTINE CHHIDDEN1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FCHHIDDEN, FMKDIR, GREEN
      P1 = 'bushel/light/hide '
      GREEN = FMKDIR (P1, 128)
      GREEN = FCHHIDDEN (P1, 5)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      GREEN = FCHHIDDEN (P1, 0)
      END
```

2.10 CHMOD change file-access permissions

Description

The **CHMOD** system call changes the access permissions, or *access mode*, of a specified file.

Note: Only the owner of a file and the super-user can change the access mode of that file.

Syntax

```
+--- Pascal -----------------------------------------------------------+
|                                                                      |
|   p_chmod (path, mode);                                              |
|                                                                      |
|                                                                      |
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
|                                                                      |
|   FCHMOD (PATH, MODE)                                                |
|                                                                      |
|                                                                      |
+----------------------------------------------------------------------+
```

Parameters

path
is the name of the file whose access mode is being changed.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

mode
is the new access mode for the file specified by **path**. The parameter value is either one of the parameter options shown here or it is constructed by logically ORing two or more of those options. The options are defined as constants in the Pascal and FORTRAN constants include files (Appendix B).

Constant	Access Attribute
ISUID	set user ID on execution
ISGID	set group ID on execution
ISVTX	save text image after execution
ENFMT	enables enforcement mode record locking
IRUSR	read by owner
IWUSR	write by owner
IXUSR	execute file (or search directory) by owner

IRGRP read by group

IWGRP write by group

IXGRP execute by group

IROTH read by others

IWOTH write by others

IXOTH execute by others

In Pascal, **mode** is of type integer.

In FORTRAN, **mode** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **CHMOD** system routine, which in these examples changes the access mode of "anyfile" to "read by others," specified by the attribute IROTH of the **mode** parameter ("red"). The file affected is assumed to be a valid file owned by the issuer of the call.

Pascal

```
procedure chmod1;

const
  %include/usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;
  green : st80;

%include /usr/include/aildefs.inc

begin
  red := IROTH;
  green := 'anyfile';
  blue := p_chmod ('anyfile', red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE CHMOD1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FCHMOD, BLUE, RED
```

```
CHARACTER*80 GREEN
RED = IROTH
GREEN = 'anyfile '
BLUE = FCHMOD (GREEN, RED)
PRINT *, BLUE
END
```

2.11 CHOWN, CHOWNX change ownership of a file

Description

The **CHOWN** and **CHOWNX** system calls change the ownership of a specified file by changing the user and group IDs. The **CHOWNX** system call, however, can specify that one of the IDs remain unchanged.

Note: Only the owner of a file and the super-user can use these system calls to change the ownership of that file.

Syntax

Pascal

```
p_chown (path, owner, group);

p_chownx (path, owner, group, tflag);
```

FORTRAN

```
FCHOWN (PATH, OWNER, GROUP)

FCHOWNX (PATH, OWNER, GROUP, TFLAG)
```

Parameters

path
is the name of the file whose owner and group IDs are being changed.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

owner
is the user ID of the new owner of the file specified by **path**.

In Pascal, **owner** is of type integer.

In FORTRAN, **owner** is of type INTEGER.

group
is the group ID of the new owner of the file specified by **path**.

In Pascal, **group** is of type integer.

In FORTRAN, **group** is of type INTEGER.

tflag
is a variable or constant, used only in the **CHOWNX** call, that specifies which of the two IDs is to remain unchanged. The options are defined in the Pascal and FORTRAN constants include files.

T_OWNER_AS_IS ignores the ID specified in the **owner** parameter.

T_GROUP_AS_IS ignores the ID specified in the **group** parameter.

Note: In FORTRAN, the underscore is omitted (for example, "TOWNERASIS").

In Pascal, **tflag** is of type integer.

In FORTRAN, **tflag** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **CHOWN** system routine, which in these examples assigns the ownership of "myfile" to the owner of root. The file affected is assumed to be a valid file owned by the issuer of the call.

Pascal

```
procedure chown1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red : integer;
  yellow : st80;

%include /usr/include/aildefs.inc

begin
  red := 0;
  green := 0;
  yellow := 'myfile';
  blue := p_chown ('myfile', red, green);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE CHOWN1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FCHOWN, BLUE, GREEN, RED
      CHARACTER*80 YELLOW
      RED = 0
      GREEN = 0
      YELLOW = 'myfile '
      BLUE = FCHOWN (YELLOW, RED, GREEN)
      PRINT *, BLUE
```

```
END
```

2.12 CHROOT change the root directory

Description

The **CHROOT** system call changes a specified directory to the effective root directory (the starting point when searching for pathnames that begin with "/").

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_chroot (path);                                                 ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FCHROOT (PATH)                                                   ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

path
is the name of the directory that will be used as the home directory for file names beginning with "/".

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **CHROOT** system routine, which in these examples makes /usr/include the effective root directory for the life of the calling process.

Pascal

```
procedure chroot1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
```

```
    red : integer;
    blue : st80;

  %include /usr/include/aildefs.inc

  begin
    blue := '/usr/include';
    red := p_chroot (blue);
    writeln (red);
  end;
```

FORTRAN

```
        SUBROUTINE CHROOT1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FCHROOT, RED
        CHARACTER*80 BLUE
        BLUE = '/usr/include '
        RED = FCHROOT (BLUE)
        PRINT *, RED
        END
```

2.13 CLOSE close a file

Description
The **CLOSE** system call closes a specified file.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_close (fildes);                                                 ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FCLOSE (FILDES)                                                   ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

fildes
is a descriptor returned by a **CREAT**, **DUP**, **DUP2**, **FCNTL**, **OPEN**, or **PIPE**, system call.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **OPEN** and **CLOSE** system routines. The **OPEN** call returns a file descriptor in the variable "red". This descriptor is used to close the same file.
Pascal

```
procedure close1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;

%include /usr/include/aildefs.inc

begin
  red := p_open ('/tmp/anyfile', RDONLY, 0);
  blue := p_close (red);
```

```
    writeln (blue);
  end;
```

FORTRAN

```
        SUBROUTINE CLOSE1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FCLOSE, FOPEN, BLUE, RED
        RED = FOPEN ('/tmp/anyfile ', RDONLY, 0)
        BLUE = FCLOSE (RED)
        PRINT *, BLUE
        END
```

2.14 CONNECT initiate a connection to a socket

Description

The **CONNECT** system call makes a connection to a specified "peer" socket if that socket is of type SOCK_DGRAM. If the socket is of type SOCK_STREAM, then this system call attempts to make a connection to another socket.

Syntax

```
+--- Pascal ------------------------------------------------------------+
|                                                                        |
|   p_connect (s, name, namelen)                                         |
|                                                                        |
|                                                                        |
+------------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
|                                                                        |
|   FCONNECT (S, NAME1, NAME2, NAMELEN)                                  |
|                                                                        |
|                                                                        |
+------------------------------------------------------------------------+
```

Parameters

s
is the descriptor of a socket that was created with a **SOCKET** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

name
specifies the socket to which a connection is to be made. Each communication space interprets this parameter in its own way.

In Pascal, **name** is of type sockaddrptr (declared in the include file ailtypes.inc).

In FORTRAN, **name1** is of type INTEGER and corresponds to sockaddr.sa_family in Pascal.

In FORTRAN, **name2** is of type CHARACTER*14 and corresponds to sockaddr.sa_data in Pascal.

namelen
is the length of the **name** parameter.

In Pascal, **namelen** is of type integer.

In FORTRAN, **namelen** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **CONNECT** system routine, which in these examples connects "s" and "s1". Socket "s" of type SOCK_DGRAM is created with a **SOCKET** system call. Another socket "s1" has been created and then bound to name "socket" with a **BIND** system call.

Pascal

```
procedure connect1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  namelen, s, s1, green : integer;
  name : sockaddrptr;

%include /usr/include/aildefs.inc

begin
  new (name);
  s := p_socket (PF_UNIX, SOCK_DGRAM, 0);
  if (s = -1) then showerror;
  s1 := p_socket (PF_UNIX, SOCK_DGRAM, 0);
  name^.sa_family := PF_UNIX;
  name^.sa_data := 'socket';
  namelen := 16;
  green := p_bind (s1, name, namelen);
  green := p_connect (s, name, namelen);
  writeln ('Connect returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE CONNECT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FCONNECT, FBIND, FSOCKET, NAME1, S, GREEN, S1
      CHARACTER*14 NAME2
      S = FSOCKET ( PFUNIX, SKDGRAM, 0 )
      S1 = FSOCKET (PFUNIX, SKDGRAM, 0)
      NAME2 = 'SOCKET '
      NAME1 = PFUNIX
      GREEN = FBIND (S1, NAME1, NAME2, 16)
      GREEN = FCONNECT (S, NAME1, NAME2, 16)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.15 CREAT create a new file

Description

The **CREAT** system call creates a new file or calls up an existing file in preparation for rewriting.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_creat (path, mode);                                               ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FCREAT (PATH, MODE)                                                 ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

path
is the name of the file being created or rewritten.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

mode
is the access mode of the file being created or rewritten. (For a list of **modes** see **CHMOD** on page 2.10.)

In Pascal, **mode** is of type integer.

In FORTRAN, **mode** is of type INTEGER.

Return Values

The return value is the file descriptor of the file created. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **CREAT** system routine. The variable "green" defines the **path** parameter. The Pascal and FORTRAN constants include files contain definitions of constants for the modes available in **CREAT**. File /tmp/test.1 is given owner read permissions as specified by the variable "red".

Pascal

```
procedure creat1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;
  green : st80;

%include /usr/include/aildefs.inc

begin
  red := IREAD;
  green := '/tmp/test.1';
  blue := p_creat (green, red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE CREAT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FCREAT, BLUE, RED
      CHARACTER*80 GREEN
      RED = IREAD
      GREEN = '/tmp/test.1 '
      BLUE = FCREAT (GREEN, RED)
      END
```

Notes

For additional information about the CREAT system call, refer to the **umask** command in *AIX Operating System Commands Reference*, which explains the interaction between the current-file-creation mask and the **mode** parameter.

2.16 DISCLAIM "disclaim" the contents of an area of memory

Description

The **DISCLAIM** system call marks an area of memory as containing data that is no longer needed. This system call cannot be used on memory that has been mapped to a file by the **SHMAT** system call.

Syntax

Pascal

```
p_disclaim (addr, length, flag)
```

Pascal external function definition

```
p_disclaim (addr: memptr; length, flag : usign) : integer;
                                                  external;
```

FORTRAN

```
FDISCLAIM (ADDR, LENGTH, FLAG)
```

Parameters

addr
points to the beginning of the memory area to be disclaimed.

In Pascal, **addr** is a pointer of type memptr (memptr is a pointer to a user-defined area of any data type.

In FORTRAN, **addr** is a user-defined area of any type.

length
specifies the number of bytes of memory to be disclaimed.

In Pascal, **length** is of type usign.

In FORTRAN, **length** is of type INTEGER;

flag
specifies that each memory location in the address range is to be set to 0 (zero). This flag must have the value specified by ZERO_MEM (ZEROMEM in FORTRAN).

In Pascal, **flag** is of type integer.

In FORTRAN, **flag** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **DISCLAIM** system routine, which in these examples disclaims the content of 10 bytes of memory in a character array ("yellow" or "ADDR"), and in effect frees that amount of memory for other use.

Pascal

```
procedure disclaim1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
  myaray = packed array[1..10] of char;
  myptr = @myaray;

var
  i : integer;
  yellow : myptr;

%include /usr/include/aildefs.inc

function p_disclaim (addr : myptr; length, flag : usign) :
                                                  integer; external

begin
  new(yellow);
  green := p_disclaim (yellow, 10, ZERO_MEM);
  writeln ('Disclaim returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE DISCLAIM1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FDISCLAIM, GREEN
      CHARACTER*80 ADDR
      GREEN = FDISCLAIM (ADDR, 10, ZEROMEM)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

Notes
Because Pascal and FORTRAN lack the facilities for handling unsigned 4-byte integers, the programmer must convert parameter values of type usign that fall in the range

2 147 483 648 through 4 294 067 295

To use a parameter value in this range, subtract 4 294 067 296 from that value before issuing the call (the result will always be negative).

2.17 DUP, DUP2 return a second file-descriptor

Description

The **DUP** and **DUP2** system calls create a second descriptor for a specified open file.

The **DUP** system call returns a new file descriptor for the specified file.

The **DUP2** system call returns a new file descriptor in one of the parameters.

The descriptor that is to be "duplicated" must be an existing descriptor returned by a **CREAT**, **DUP**, **DUP2**, **FCNTL**, **OPEN**, **PIPE**, **SOCKET**, or **SOCKETPAIR** system call. The new file descriptor is synonymous with the existing one (that is, the new descriptor points to the same file).

Syntax

Pascal

```
p_dup (fildes);

p_dup2 (oldfd, newfd);
```

FORTRAN

```
FDUP (FILDES)

FDUP2 (OLDFD, NEWFD)
```

Parameters

fildes
is the file descriptor to be duplicated by the **DUP** system call.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** of type INTEGER.

oldfd
is the file descriptor to be duplicated by the **DUP2** system call.

In Pascal, **oldfd** is of type integer.

In FORTRAN, **oldfd** of type INTEGER.

newfd
is the new file-descriptor generated by the **DUP2** system call.

In Pascal, **newfd** is of type integer.

In FORTRAN, **newfd** of type INTEGER.

Return Values

The return value is the new file-descriptor. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow make calls to the **DUP** system routine, which returns a file descriptor in the variable "blue". The Pascal and FORTRAN constants include files contain definitions of constants for the modes available in **OPEN**.

Pascal

```
procedure dup1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;

%include /usr/include/aildefs.inc

begin
  red := p_open ('/usr/include/ailtypes.inc', RDONLY, 0);
  blue := p_dup (red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE DUP1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FDUP, FOPEN, BLUE, RED
      RED = FOPEN ('/usr/include/ailtypes.inc ', RDONLY, 0)
      BLUE = FDUP (RED)
      PRINT *, BLUE
      END
```

2.18 EXECL, EXECLE, EXECLP execute a program

Description

The **EXEC** system call, in all its forms, executes a new program in the calling process. The call does not create a new process but overlays the current program with a new one.

The three **EXEC** calls described in this section pass a maximum of four arguments to a specified executable file. This restriction on the number of arguments is what distinguishes these three system calls from those described in the next section.

The **EXECLE** call differs from the other two in having an **envp** parameter.

The **EXECLP** call is issued with the same arguments as **EXECL**, but it duplicates the shell actions in searching for an executable file in a list of directories.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                      ¦
¦   p_execl (path, arg0, arg1, arg2, arg3);                            ¦
¦                                                                      ¦
¦   p_execle (path, arg0, arg1, arg2, arg3, envp);                     ¦
¦                                                                      ¦
¦   p_execlp (filenm, arg0, arg1, arg2, arg3);                         ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                      ¦
¦   FEXECL (PATH, ARG0, ARG1, ARG2, ARG3)                              ¦
¦                                                                      ¦
¦   FEXECLE (PATH, ARG0, ARG1, ARG2, ARG3, ENVP)                       ¦
¦                                                                      ¦
¦   FEXECLP (FILENM, ARG0, ARG1, ARG2, ARG3)                           ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

path
is the explicit path (location) of the file to be executed. This parameter is used in the **EXECL** and **EXECLE** calls.

In Pascal, **path** is of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

filenm
is the name of the file to be executed. This parameter is used in the **EXECLP** call, which will search for the specified file only in the current and default directories.

In Pascal, **filenm** is of type st80.

In FORTRAN, **filenm** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

arg0, arg1, arg2, and **arg3**
are string variables or constants. They hold the arguments to be passed to the file specified by **filenm** or **path**. The value of **arg0** must be **filenm** or the last attribute of **path**.

In Pascal, each **arg** is of type st80. If fewer than four arguments are required, the remaining strings must be nil strings.

In FORTRAN, each **arg** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space. If fewer than four arguments are required, the remaining strings must each contain one, and only one, blank.

envp
is a parameter used only in **EXECLE** (and **EXECVE**; see next section). It is an 80-element array that holds the attributes of the execution environment of the calling process. Each element is an 80-byte character string.

In Pascal, **envp** is a variable of type pasargv. The terminating string in the array must be a nil string.

In FORTRAN, **envp** is an array of strings of type CHARACTER*80. The terminating character of a string must be a blank space. The terminating string in the array must contain one, and only one, blank.

Note: For details of this parameter, see the **sh** command in *AIX Operating System Commands Reference*.

Return Values
There is no return value from a successful **EXEC** call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **EXECL** system routine, which prints the current date.

Pascal

```
procedure execl1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  merlin : integer;
  arg0, arg1, arg2, arg3, path : st80;
```

```
%include /usr/include/aildefs.inc

begin
  path := '/bin/sh';
  arg0 := 'sh';
  arg1 := '-c';
  arg2 := 'date';
  arg3 := '';
  merlin := p_execl (path, arg0, arg1, arg2, arg3)
end;
```

FORTRAN

```
      SUBROUTINE EXECL1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FEXECL, MERLIN
      CHARACTER*80 ARG0, ARG1, ARG2, ARG3, PATH
      PATH = '/bin/sh '
      ARG0 = 'sh '
      ARG1 = '-c '
      ARG2 = 'date '
      ARG3 = ' '
      MERLIN = FEXECL (PATH, ARG0, ARG1, ARG2, ARG3)
      END
```

(*) The **EXECV**, **EXECVE**, and **EXECVP** calls are described in the next subsection (page 2.19).

2.19 EXECV, EXECVE, EXECVP execute a program

Description

The three **EXEC** system calls described in this section can pass a maximum of 80 arguments to a specified executable file (in contrast to the maximum of four arguments that can be passed by the **EXEC** routines described in the preceding section).

The **EXECVE** call differs from the other two in having an **envp** parameter.

The **EXECVP** call is issued with the same arguments as **EXECV**, but it duplicates the shell actions in searching for an executable file in a list of directories.

Syntax

Pascal

```
p_execv (path, args);

p_execve (path, args, envp);

p_execvp (filenm, args);
```

FORTRAN

```
FEXECV (PATH, ARGS)

FEXECVE (PATH, ARGS, ENVP)

FEXECVP (FILENM, ARGS)
```

Parameters

path
is the explicit path (location) of the file to be executed. This parameter is used in the **EXECV** and **EXECVE** calls.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

filenm
is the name of the file to be loaded and executed. This parameter is used in the **EXECVP** call, which searches for the specified file only in the current and default directories.

In Pascal, **filenm** is a string variable or constant of type st80.

In FORTRAN, **filenm** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

args
is an array of strings. It holds any arguments to be passed to the file specified by **filenm** or **path**. The first element of the array should be **filenm** or the last attribute of **path**.

In Pascal, **args** is a variable of type pasargv declared in the ailtypes.inc file. The terminating string must be a nil string.

In FORTRAN, **args** is a string variable or constant of type CHARACTER*80. The terminating character of a string must be a blank space. The terminating string must contain one, and only one, blank.

envp
is a parameter used only in **EXECVE** (and **EXECLE**, described in the preceding section). It is an 80-element array that holds the attributes of the execution environment of the calling process. (Each element is an 80-byte character string.)

In Pascal, **envp** is a variable of type pasargv, declared in the types file. The terminating string in the array must be a nil string.

In FORTRAN, **envp** is an array of strings of type CHARACTER*80. The terminating character of a string must be a blank space. The terminating string in the array must contain one, and only one, blank space.

For details of this parameter, see the description of the **sh** command in *&AIX Commands Reference*.

Return Values
There is no return value from a successful **EXEC** call. The value -1 is returned and an error code set in **errno** if the call fails.

If **EXECVP** is called to execute a shell command file and it is impossible to execute that file, the values of args[0] and args[1] are modified before the return.

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **EXECV** system routine, which will produce a listing of the current working directory (see **Notes**).

Pascal

```
procedure execvp1;

const
  %include /usr/include/ailfconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  merlin : integer;
  name : st80;
  args : pasargv;

%include /usr/include/aildefs.inc
```

```
  begin
    name := 'examp';
    args[1] := 'examp';
    args[2] := '-x';
    args[3] := -F';
    args[4] := '-f';
    args[5] := '';
    merlin := p_execvp (name, args)
  end;
```

FORTRAN

```
        SUBROUTINE EXECVP1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FEXECPV, MERLIN
        CHARACTER*80 ARGS(80), NAME
        NAME = 'examp '
        ARGS(1) = 'examp '
        ARGS(2) = '-x '
        ARGS(3) = '-F '
        ARGS(4) = '-f '
        ARGS(5) = ' '
        MERLIN = FEXECVP (NAME, ARGS)
        END
```

Notes

The executable file 'examp' must be in the current directory before these examples will work.

2.20 EXIT, _EXIT terminate a process

Description

The **EXIT** system call is the standard means of terminating a process.

The **_EXIT** call terminates a process without performing any of the clean-up operations performed by the **EXIT** routine.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                        ¦
¦   p_exit (status);                                                     ¦
¦                                                                        ¦
¦   p__exit (status);                                                    ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                        ¦
¦   FEXIT (STATUS)                                                       ¦
¦                                                                        ¦
¦   FEEXIT (STATUS)                                                      ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

Parameters

status
is the termination status returned to the parent process.

In Pascal, **status** is of type integer.

In FORTRAN, **status** is of type INTEGER.

Return Values

There is no return value from a successful **EXIT** or **_EXIT** call.

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **EXIT**, **FORK**, and **WAIT** system routines. Both create a child process, which issues the **EXIT** call. The parent process executes a **WAIT** call, and the parameter of that call ("green") receives the low-order eight bits of the value that the child passes to the **EXIT** routine. It is this value that is printed.

Pascal

```
  procedure exit1;

  const
    %include /usr/include/ailpconsts.inc
  type
    %include /usr/include/ailtypes.inc

  var
```

```
  blue, green, red, yellow : integer;

%include /usr/include/aildefs.inc

begin
  green := p_fork;
  if green = 0 then
     blue := p_exit (red);
  yellow := p_wait (green);
  writeln ('status  ', green);
end;
```

FORTRAN

```
      SUBROUTINE EXIT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FEXIT, FFORK, FWAIT, BLUE, GREEN, RED, YELLOW
      GREEN = FFORK ()
      IF (GREEN .EQ. 0) THEN
         BLUE = FEXIT (RED)
      ENDIF
      YELLOW = FWAIT (GREEN)
      PRINT *, 'STATUS ', GREEN
      END
```

2.21 FABORT abort the changes to a file

Description

The **FABORT** system call cancels data changes made to a specified file. The file must be open for write or read/write at the time the call is made. If no changes have been made since the file was last written to storage, the call has no effect.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_fabort (fildes);                                                 ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+

+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FFABORT (FILDES)                                                   ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

fildes
is the descriptor of a file that has been opened for write or read/write.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **FABORT** system routine, which in these examples cancels changes made to the file /usr/include/junk since the last time it was filed.

Pascal

```
procedure fabort1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red : integer;

%include /usr/include/aildefs.inc
```

```
begin
  red := p_open ('/usr/include/junk', WRONLY, 0);
  blue := p_fcommit (red);

{ The file can be changed between these two calls }.

  green := p_fabort (red);
  writeln ('Fabort returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE FABORT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FFABORT, FFCOMMIT, FOPEN, BLUE, RED, YELLOW
      RED = FOPEN ('/usr/include/junk ', WRONLY, 0)
      BLUE = FFCOMIT (RED)

C     THE FILE CAN BE CHANGED BETWEEN THESE TWO CALLS.

      BLUE = FFABORT (RED)
      IF (BLUE .EQ. -1) PRINT *, 'FABORT: ERROR'
      IF (BLUE .NE. -1) PRINT *, 'FABORT: OK'
      END
```

2.22 FCLEAR clear space in a file

Description

The **FCLEAR** system call clears space (makes a "hole") in a file by writing binary zeros to a specified number of bytes in that file. This "zeroing" process begins at the current position of the seek pointer of the file specified in the call.

Syntax

```
+--- Pascal -------------------------------------------------------------+
¦                                                                        ¦
¦   p_fclear (fildes, nbytes);                                           ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

```
+--- FORTRAN ------------------------------------------------------------+
¦                                                                        ¦
¦   FFCLEAR (FILDES, NBYTES)                                             ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

Parameters

fildes
is the descriptor of the file in which space is being cleared.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

nbytes
is a constant or a variable specifying the number of bytes to be zeroed. If this number falls within a certain range, the programmer will have to use a conversion formula to obtain the proper value for **nbytes** (see **Notes**).

In Pascal **nbytes** is of type usign.

In FORTRAN **nbytes** is of type INTEGER.

Return Values

The return value is **nbytes**. If this value falls within a certain range, the programmer will have to use a conversion formula to obtain the actual number (see **Notes**).

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **FCLEAR** system routine, which overwrites the specified open file /tmp/junk with 200 null characters.

Pascal

```
  procedure fclear1;

  const
    %include /usr/include/ailpconsts.inc
```

```
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;

%include /usr/include/aildefs.inc

begin
  red := p_open ('/tmp/junk', WRONLY, 0);
  blue := p_fclear (red, 200);
  writeln (blue);
end;
```

FORTRAN

```
        SUBROUTINE FCLEAR1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FFCLEAR, FOPEN, BLUE, RED
        RED = FOPEN ('/tmp/junk ', WRONLY, 0)
        BLUE = FFCLEAR (RED, 200)
        PRINT *, BLUE
        END
```

Notes

Because Pascal and FORTRAN lack the facilities for handling unsigned 4-byte integers, the programmer must convert parameter values of type usign that fall in the range

2 147 483 648 through 4 294 067 295

To use a parameter value in this range, *subtract* 4 294 067 296 from the parameter value (the result will always be negative) before issuing the call.

Conversely, if the return value is a negative number, *add* 4 294 067 296 to that number to obtain the correct value.

2.23 FCNTL control an open-file descriptor

Description

The **FCNTL** system call performs various control operations on an open-file descriptor.

Syntax

```
+--- Pascal ---------------------------------------------------------+
|                                                                     |
|   p_fcntl (fildes, cmd, arg);                                       |
|                                                                     |
|                                                                     |
+---------------------------------------------------------------------+

+--- Pascal external function declaratios ---------------------------+
|                                                                     |
|   function p_fcntl (fildes, cmd : int; var arg : integer) : integer;|
|                                                           external; |
|                                                                     |
|   or                                                                |
|                                                                     |
|   function p_fcntl (fildes, cmd : int; var arg : flockrec) : integer;|
|                                                            external;|
|                                                                     |
|                                                                     |
+---------------------------------------------------------------------+

+--- FORTRAN --------------------------------------------------------+
|                                                                     |
|   FFCNTL (FILDES, CMD, ARG)                                         |
|                                                                     |
|                                                                     |
+---------------------------------------------------------------------+
```

Parameters

fildes

is a descriptor returned by a **CREAT**, **DUP**, **DUP2**, **FCNTL**, **OPEN**, **PIPE**, **SOCKET**, or **SOCKETPAIR** system call.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

cmd

is a variable or constant specifying the operation to be performed. The options are defined as constants in the Pascal and FORTRAN constants include files.

F_DUPFD returns a new file descriptor.

F_GETFD returns the value of the close-on-exec flag associated with the file descriptor **fildes.**

F_SETFD sets the close-on-exec flag associated with **fildes** to the value of the low-order bit of **arg**.

F_GETFL gets the file status flags of the file descriptor. **fildes**.

F_SETFL sets the file status flags to the value of **arg**.

F_GETLK gets the first blocking file lock.

F_SETLK sets or clears a file lock.

F_SETLKW waits, if necessary, to set or clear a file lock.

F_GETOWN gets the process ID or process-group ID set to receive signals.

F_SETOWN sets the process ID or process-group ID set to receive signals.

Note: In FORTRAN, the underscore is omitted (for example, "FDUPFD").

In Pascal, **cmd** is of type integer.

In FORTRAN, **cmd** is of type INTEGER.

arg
varies according to the **cmd** parameter.

In Pascal, **arg** is of type integer for all values of **cmd** except F_GETLK, F_SETLK, and F_SETLK. For these values, **arg** is of type flockrec. Possible values for the l_type field are:

F_RDLCK = 1
F_WRLCK = 2
F_UNLCK = 3

In FORTRAN, **arg** is of type integer for all values of **cmd** except F_GETLK, F_SETLK, and F_SETLK. For these values, **arg** is of type INT*2 ARG(Possible values for **arg**[1] are:

FRDLCK = 1
FWRLCK = 2
FUNLCK = 3

Return Values
The value returned varies according to the command option specified in the **cmd** parameter of the call:

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **FCNTL** system routine, which in these examples opens the file /usr/include/ailtypes.inc for reading and writing. The file descriptor returned by the **OPEN** call is used for the **fildes** parameter ("blue") in **FCNTL**; the **cmd** parameter ("red") instructs the system to return the file-status flags of the file descriptor. This is the value printed out.

Pascal

```
procedure fcntl1;
```

```
const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red, yellow : integer;

%include /usr/include/aildefs.inc

function p_fcntl (fildes, cmd : int; var arg : integer) : integer; external

begin
  red := F_GETFL;
  green := 0;
  blue := p_open ('/usr/include/ailtypes.inc', 2, 0);
  yellow := p_fcntl (blue, red, green);
  writeln (yellow);
end;
```

FORTRAN

```
      SUBROUTINE FCNTL1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FFCNTL, FOPEN, BLUE, GREEN, RED, YELLOW
      RED = FGETFL
      GREEN = 0
      BLUE = FOPEN ('/usr/include/ailtypes.inc ', 2, 0)
      YELLOW = FFCNTL (BLUE, RED, GREEN)
      PRINT *, YELLOW
      END
```

2.24 FORK create a process

Description

The **FORK** system call creates a new process whose memory image is a copy of the memory image of the process that issued the **FORK** call.

Syntax

```
+--- Pascal -----------------------------------------------------------+
|                                                                       |
|   p_fork;                                                             |
|                                                                       |
|                                                                       |
+-----------------------------------------------------------------------+

+--- FORTRAN ----------------------------------------------------------+
|                                                                       |
|   FFORK ()                                                            |
|                                                                       |
|                                                                       |
+-----------------------------------------------------------------------+
```

Parameters

This system call has no parameters.

Return Values

Upon successful completion, **FORK** returns the value 0 to the child process and the process ID of the child to the parent. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **FORK** system routine to create a new process. The process ID of the child is returned to the parent process in the variable "blue", and the value 0 to the child process. Therefore both 0 and the process ID of the child are printed out.

Pascal

```
procedure fork1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue : integer;

%include /usr/include/aildefs.inc

begin
  blue := p_fork;
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE FORK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FFORK, BLUE
      BLUE = FFORK ()
      PRINT *, BLUE
      END
```

2.25 FSYNC, FCOMMIT write to permanent storage

Description

FSYNC and **FCOMMIT** are synonymous system calls that write all modified data in a specified open file to permanent storage.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_fsync (fildes);                                                   ¦
¦                                                                       ¦
¦   p_fcommit (fildes);                                                 ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FFFSYNC (FILDES)                                                    ¦
¦                                                                       ¦
¦   FFCOMMIT (FILDES)                                                   ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

fildes
is the descriptor of an open file.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **FSYNC** system routine, which writes changes in a specified file to permanent storage.

Pascal

```
  procedure fsync1;

  const
    %include /usr/include/ailpconsts.inc
  type
    %include /usr/include/ailtypes.inc
  var
    blue, red, yellow : integer;
```

```
%include /usr/include/aildefs.inc

begin
  red := p_open ('/tmp/junk', WRONLY, 0);
  blue := p_fsync (red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE FSYNC1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FFFSYNC, FOPEN, BLUE, RED
      RED = FOPEN ('/tmp/junk ', WRONLY, 0)
      BLUE = FFFSYNC (RED)
      PRINT *, BLUE
      END
```

2.26 FTRUNCATE truncate a file

Description
The **FTRUNCATE** system call counts a specified number of bytes from the beginning of a specified file and then deletes all the remaining bytes.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_ftruncate (fildes, len);                                         ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FFTRUNCATE (FILDES, LEN)                                           ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

fildes
is the descriptor of an open file.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

len
is the number of bytes to be left in the truncated file, counting from the first byte. (See **Notes**.)

In Pascal, **len** is of type usign.

In FORTRAN, **len** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **FTRUNCATE** system routine, which in these examples truncates the file /tmp/xxx (assuming that it exists) to a length of 100 bytes as specified by the **len** parameter ("blue").

Pascal

```
  procedure ftruncate1;

  const
    %include /usr/include/ailpconsts.inc
```

```
  type
    %include /usr/include/ailtypes.inc
  var
    blue, red, yellow : integer;
    orange : st80;

  %include /usr/include/aildefs.inc

  begin
    orange := '/tmp/xxx';
    blue := 100;
    red := p_open (orange, WRONLY, 0);
    yellow := p_ftruncate (red, blue);
    writeln (yellow);
  end;
```

FORTRAN

```
        SUBROUTINE FTRUNCATE1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FFTRUNCATE, FOPEN, BLUE, RED, YELLOW
        CHARACTER*80 ORANGE
        ORANGE = '/tmp/xxx '
        BLUE = 100
        RED = FOPEN (orange, WRONLY, 0)
        YELLOW = FFTRUNCATE (RED, BLUE)
        PRINT *, YELLOW
        END
```

Notes

Because Pascal and FORTRAN lack the facilities for handling unsigned 4-byte integers, the programmer must convert parameter values of type usign that fall in the range

2 147 483 648 through 4 294 067 295

To use a parameter value in this range, subtract 4 294 067 296 from that value before issuing the call (the result will always be negative).

2.27 GETDTABLESIZE get the size of a process-descriptor table

Description

The **GETDTABLESIZE** system returns the size of the process-descriptor table, which has at least 20 slots for each process. In AIX the value returned is 200.

Syntax

Pascal

```
p_getdtablesize;
```

FORTRAN

```
FGETDTABLESIZE ()
```

Parameters

This system call has no parameters.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **GETDTABLESIZE** system routine.

Pascal

```
procedure getdtablesize1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
begin
  green := p_getdtablesize;
  writeln ('tablesize is ', green);
  if green = -1 then
    writeln ('Getdtablesize: ERROR')
  else
    writeln ('Getdtablesize: OK');
end;
```

FORTRAN

```
SUBROUTINE GETDTABLESIZE1
INCLUDE (/usr/include/ailfconsts.inc)
INTEGER FGETDTABLESIZE, GREEN
GREEN = FGETDTABLESIZE()
IF (GREEN .EQ. -1) THEN
  PRINT *, 'GETDTABLESIZE: ERROR'
  CALL ERRORS
ELSE
  PRINT *, 'GETDTABLESIZE: OK'
ENDIF
END
```

2.28 GETGROUPS get a group access list

Description

The **GETGROUPS** system call gets the group access list of the current process and stores it in an array specified in the call.

Syntax

```
+--- Pascal ---------------------------------------------------------+
|                                                                    |
|   p_getgroups (ngrps, gidset);                                     |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
|                                                                    |
|   FGETGROUPS (NGRPS, GIDSET)                                       |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
```

Parameters

ngrps
is the number of entries that can be stored in the array specified by the **gidset** parameter.

In Pascal, **ngrps** is of type integer.

In FORTRAN, **ngrps** is of type INTEGER.

gidset
is an array in which the requested list items will be put. The maximum number of elements the array may hold is equal to the constant NGROUP defined in the Pascal and FORTRAN constants include files.

In Pascal, **gidset** is of type intngroup. (Getptr is a pointer to a user-defined integer array.)

In FORTRAN, **gidset** is a user-defined array of type INTEGER.

Return Values

The value returned upon successful completion of the call is the number of elements stored in the group access list. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **GETGROUPS** system routine, which in these example returns a number that is equal to the number of elements in the array specified by the variable "red".

Pascal

```
procedure getgroups1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green : integer;
  red : intngroup;

begin
  green := 20;
  blue := p_getgroups (green, red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE GETGROUPS1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FGETGROUPS, BLUE, GREEN, RED(20)
      GREEN = 20
      BLUE = FGETGROUPS (GREEN, RED)
      PRINT *, BLUE
      END
```

2.29 GETHOSTID get a host ID

Description
The **GETHOSTID** system returns an integer identifier for the current host.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_gethostid;                                                       ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FGETHOSTID ()                                                      ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters
This system call has no parameters.

Return Values
The identifier for the current host is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow return the host ID in the variable "green".

Pascal

```
procedure gethostid1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;

%include /usr/include/aildefs.inc

begin
  green := p_gethostid;
  writeln ('Gethostid returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
SUBROUTINE GETHOSTID1
INCLUDE (/usr/include/ailfconsts.inc)
INTEGER FGETHOSTID, GREEN
GREEN = FGETHOSTID ()
PRINT *, GREEN
IF (GREEN .EQ. -1) CALL ERRORS
END
```

2.30 GETHOSTNAME get a local host name

Description

The **GETHOSTNAME** system returns the name of the current host.

Syntax

```
+--- Pascal -------------------------------------------------------------+
¦                                                                        ¦
¦   p_gethostname (name, namelen);                                       ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

```
+--- FORTRAN ------------------------------------------------------------+
¦                                                                        ¦
¦   FGETHOSTNAME (NAME, NAMELEN)                                         ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

Parameters

name
receives the name of the host machine.

In Pascal, **name** is of type st80.

In FORTRAN, **name** is of type CHARACTER*80.

namelen
is the length of the name parameter.

In Pascal, **namelen** is of type integer.

In FORTRAN, **namelen** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page return the name of the current host in the variable **name**.

Pascal

```
  procedure gethostname1;

  const
    %include /usr/include/ailpconsts.inc
  type
    %include /usr/include/ailtypes.inc
  var
    green, namelen : integer;
```

```
    name : st80;

  %include /usr/include/aildefs.inc

  begin
    namelen := 20;
    green := p_gethostname (name, namelen);
    writeln ('Gethostname returned: ', green : 2);
    if (green = -1) then showerror;
  end;
  end;
```

FORTRAN

```
        SUBROUTINE GETHOSTNAME1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FGETHOSTNAME, NAMELEN, GREEN
        CHARACTER*80 NAME
        NAMELEN = 20
        GREEN = FGETHOSTNAME (NAME, NAMELEN)
        PRINT *, GREEN
        IF (GREEN .EQ. -1) CALL ERRORS
        END
```

2.31 GETITIMER get the current value of an internal timer

Description

The **GETITIMER** system call returns the value of the internal timer specified in the call.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                     ¦
¦   p_getitimer (which, value);                                       ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                     ¦
¦   FGETITIMER (WHICH, VALUE)                                         ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

which
specifies one of the following timers:

ITIMER_REAL the timer decrements in real time.

ITIMER_VIRTUAL the timer decrements in process virtual time (it runs only when the process is executing).

ITIMER_PROF the timer decrements both in process virtual time and when the operating system is executing on behalf of the process.

Note: In FORTRAN, the underscore is omitted (for example, "ITIMERREAL").

In Pascal, **which** is of type integer.

In FORTRAN, **which** is of type INTEGER.

value
is a variable in which the time is returned when the call is executed.

In Pascal, **value** is of type itimerval, declared in the include file ailtypes.inc.

In FORTRAN, **value** is an array of four integers, or INTEGER VALUE(4).

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **GETITIMER** system routine, which in these examples get the current value of the ITIMER_REAL timer. This value is returned in the variables "vvalue" (Pascal) and "VAL" (FORTRAN).

Pascal

```
procedure getitimer1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  which : integer;
  vvalue : itimerval;

%include /usr/include/aildefs.inc

begin
  new(vvalue);
  which := ITIMER_REAL;
  green := p_getitimer (which, vvalue);
  writeln ('Getitimer returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE GETITIMER1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FGETITIMER, VAL(4), GREEN
      GREEN = FGETITIMER (ITIMERREAL, VAL)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.32 GETLOCAL get the alias for <LOCAL>

Description
The **GETLOCAL** system call gets the alias for <LOCAL>.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_getlocal (localname, maxlength)                                  ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FGETLOCAL (LOCALNAME, MAXLENGTH)                                   ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

localname
receives the pathname for <LOCAL>.

In Pascal, **localname** is of type st80.

In FORTRAN, **localname** is of type CHARACTER*80.

maxlength
is the maximum length of the **localname** buffer.

In Pascal, **maxlength** is of type integer.

In FORTRAN, **maxlength** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **GETLOCAL** system routine, and the alias for <LOCAL> is placed in **buf**.

Pascal

```
procedure getlocal1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
```

```
    buf : st80;

  %include /usr/include/aildefs.inc

  begin
    green := p_getlocal (buf, 50);
    writeln ('Alias for local is ', buf);
    writeln ('Getlocal returned: ', green : 2);
    if (green = -1) then showerror;
  end;
```

FORTRAN

```
        SUBROUTINE GETLOCAL1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FGETLOCAL, GREEN
        CHARACTER BUF(80)
        PRINT *, 'Calling Getlocal'
        GREEN = FGETLOCAL (BUF, 20)
        PRINT *, GREEN
        IF (GREEN .EQ. -1) CALL ERRORS
        END
```

2.33 GETPEERNAME get the name of a "peer" socket

Description

The **GETPEERNAME** system call returns the name of the "peer" connected to the socket specified in the call.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_getpeername (s, name, namelen)                                   ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FGETPEERNAME (S, NAME1, NAME2, NAMELEN)                            ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

s
: is the descriptor of a socket that was created with a **SOCKET** or **SOCKETPAIR** system call.

    In Pascal, **s** is of type integer.

    In FORTRAN, **s** is of type INTEGER.

name
: receives the name of the peer upon completion of the call.

    In Pascal, **name** is of type sockaddrptr (declared in the include file ailtypes.inc).

    In FORTRAN, **name1** is of type INTEGER and corresponds to sockaddr.sa_family in Pascal.

    In FORTRAN, **name2** is of type CHARACTER*14 and corresponds to sockaddr.sa_data in Pascal.

namelen
: is the length of the **name** parameter. It should be initialized to indicate the amount of space pointed to by **name**. It receives the actual size of the peer name upon completion of the call.

    In Pascal, **namelen** is of type integer.

    In FORTRAN, **namelen** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **GETPEERNAME** system routine, which in these examples returns (in the variable "name1"), the name associated with socket "sv[1]" (previously created and bound to the name "sockname" with a **BIND** system call).

Pascal

```
procedure getpeername1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  namelen, blue, gray : integer;
  name, name1 : sockaddrptr;
  sv : int2;

%include /usr/include/aildefs.inc

begin
  new (name);
  new(name1);
  namelen := 16;
  green:=p_socketpair(PF_UNIX, SOCK_DGRAM, 0, sv);
  name^.sa_data := 'abc';
  name^.sa_family := PF_UNIX;
  green := p_unlink('abc');
  gray := p_bind (sv[2], name, namelen);
  green := p_getpeername (sv[1], name1, namelen);
  if (green <> -1) then
    writeln('Getpeername returned : OK')
  else
    writeln('Getpeername returned : ERROR');
  if (green = -1) then showerror;
  green:=p_unlink ('abc');
  green:=p_shutdown (sv[1], 2);
  green:=p_shutdown (sv[2], 2);
end;
```

FORTRAN

```
      SUBROUTINE GETPEERNAME1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FGETPEERNAME, FBIND, FSHUTDOWN, FSOCKETPAIR, FUNLINK
      INTEGER GREEN, LEN, SV(2)
      CHARACTER*14 NAME, NAME1
      PROT = 0
      GREEN = FSOCKETPAIR (PFUNIX, SKDGRAM, 0, SV)
      NAME = 'BNAME '
      GREEN = FUNLINK ('BNAME ')
      GREEN = FBIND (SV(1), PFUNIX, NAME, 16)
      LEN = 16
      NAME2 = 'SOCKNAME'
      GREEN = FGETPEERNAME (SV(2), PFUNIX, NAME1, LEN)
```

```
IF (GREEN .LE. -1) THEN
  PRINT *, 'GETPEERNAME : ERROR'
ELSE
  PRINT *, 'GETPEERNAME : OK'
ENDIF
GREEN = FUNLINK ('BNAME ')
GREEN = FSHUTDOWN (SV(1), 2)
GREEN = FSHUTDOWN (SV(2), 2)
END
```

2.34 GETPGRP, GETPID, GETPPID get a process-group or process identifier

Description

The **GET** system calls described in this and the following section return the ID of a group, process, or user.

GETPGRP returns the process group ID of the calling process.

GETPID returns the process ID of the calling process and is often used to generate uniquely named temporary files.

GETPPID returns the process ID of the parent process.

Syntax

Pascal

```
p_getpgrp;

p_getpid;

p_getppid;
```

FORTRAN

```
FGETPGRP ()

FGETPID ()

FGETPPID ()
```

Parameters

These system calls have no parameters.

Return Values

The return value of each of the three calls is a particular ID (see description above).

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **GETPID** system routine, which returns the process ID in the variable "blue".

Pascal

```
procedure getpid1;

const
  %include /usr/include/ailpconsts.inc
type
```

```
  %include /usr/include/ailtypes.inc
var
  blue : integer;

%include /usr/include/aildefs.inc

begin
  blue := p_getpid;
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE GETPID1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FGETPID, BLUE
      BLUE = FGETPID ()
      PRINT *, BLUE
      END
```

2.35 GETSOCKNAME get a socket name

Description

The **GETSOCKNAME** system returns the current name of the socket specified in the call.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_getsockname (s, name, namelen)                                   ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FGETSOCKNAME (S, NAME1, NAME2, NAMELEN)                            ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

s
is the descriptor of a socket that was created with a **SOCKET** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

name
receives the name of the socket upon completion of the call.

In Pascal, **name** is of type sockaddrptr (declared in the include file ailtypes.inc).

In FORTRAN, **name1** is of type INTEGER and corresponds to sockaddr.sa_family in Pascal.

In FORTRAN, **name2** is of type CHARACTER*14 and corresponds to sockaddr.sa_data in Pascal.

namelen
is the length of the **name** parameter. It should be initialized to indicate the amount of space pointed to by **name**. It receives the actual size of the socket name upon completion of the call.

In Pascal, **namelen** is of type integer.

In FORTRAN, **namelen** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **GETSOCKNAME** system routine, which in these examples returns in the variable "name1" the name 'sockname', which was bound to socket "s&cdq, with a **BIND** system call.

Pascal

```
procedure getsockname1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  namelen, s, green : integer;
  name, name1 : sockaddrptr;

%include /usr/include/aildefs.inc

begin
  new (name);
  name^.sa_data := 'sockname';
  name^.sa_family := PF_UNIX;
  s := p_socket (PF_UNIX, SOCK_STREAM, 0);
  if (s = -1) then showerror;
  new (name1);
  namelen := 16;
  green := p_bind (s, name, namelen);
  green := p_getsockname (s, name1, namelen);
  writeln ('Getsockname returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE GETSOCKNAME1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FGETSOCKNAME, FBIND, FSOCKET, S, NAMELEN
      INTEGER, GREEN, NAME1, RC
      CHARACTER*14 NAME, NAME2
      S = FSOCKET (PFUNIX, SKSTRM, 0)
      NAME2 = 'sockname '
      NAMELEN = 16
      NAME1 = PFUNIX
      RC = FBIND (S, NAME1, NAME2, NAMELEN)
      IF (S .EQ. -1) CALL ERRORS
      GREEN = FGETSOCKNAME (S, NAME1, NAME, NAMELEN)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.36 GETSOCKOPT get socket options

Description

The **GETSOCKOPT** system gets the options associated with a specified socket. These options may exist at multiple protocol levels, and are always present at the uppermost socket level.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

Pascal

```
p_getsockopt (s, level, optname, optval, optlen)
```

FORTRAN

```
FGETSOCKOPT (S, LEVEL, OPTNAME, OPTVAL, OPTLEN)
```

Parameters

s
is the descriptor of a socket that was created with a **SOCKET** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

level
level at which the desired option resides. To manipulate options at the socket level, specify the level as SOL_SOCKET.

In Pascal, **level** is of type integer.

In FORTRAN, **level** is of type INTEGER;

optname
is the option name, passed uninterpreted to the appropriate protocol module for interpretation. The socket-level options are:

SO_DEBUG turns on recording of debugging information.

SO_REUSEADDR allows local address reuse.

SO_KEEPALIVE keeps connections alive.

SO_DONTROUTE does not apply routing on outgoing messages.

SO_LINGER lingers on a **CLOSE** system call if data is present.

SO_OOBINLINE leaves received out-of-band data in line.

SO_SNDBUF sends buffer size.

SO_RCVBUF receives buffer size.

SO_ERROR gets error status.

SO_TYPE gets socket type.

SO_BROADCAST requests permission to transmit broadcast messages.

Note: In FORTRAN, the underscore is omitted (for example, "SODEBUG").

In Pascal, **optname** is of type integer.

In FORTRAN, **optname** is of type INTEGER.

optval
points to a buffer, in which the option values are returned by the system call.

In Pascal, **optval** is of type st80.

In FORTRAN, **optval** is of type CHARACTER*80.

optlen.
specifies the length of the buffer pointed to by **optval**.

In Pascal, **optlen** is of type integer.

In FORTRAN, **optlen** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **GETSOCKOPT** system routine, which in these examples returns the options associated with socket "s".

Pascal

```
procedure getsockopt1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  level, optlen, optname, s, green : integer;
  optval : st80;

%include /usr/include/aildefs.inc

begin
```

```
    s := p_socket (PF_UNIX, SOCK_STREAM, 0);
    level := SOL_SOCKET;
    optlen := 80;
    green := p_getsockopt (s, level, optname, optval, optlen);
    writeln ('Getsockopt returned: ', green : 2);
    if (green = -1) then showerror;
  end;
```

FORTRAN

```
        SUBROUTINE GETSOCKOPT1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FGETSOCKOPT, FSOCKET, LEVEL, OPTLEN, OPTNAME, S, GREEN
        CHARACTER*80 OPTVAL
        PRINT *, 'Calling Getsockopt'
        S = FSOCKET (PFUNIX, SKSTRM, 0)
        IF (S .EQ. -1) CALL ERRORS
        LEVEL = SOLSOCKET
        OPTLEN = 80;
        GREEN = FGETSOCKOPT (S, LEVEL, OPTNAME, OPTVAL, OPTLEN)
        PRINT *, GREEN
        IF (GREEN .EQ. -1) CALL ERRORS
        END
```

2.37 GETTIMEOFDAY get the current time

Description

The **GETTIMEOFDAY** system call gets the current time.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_gettimeofday (tp, tzp);                                        ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+

+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FGETTIMEOFDAY (TP, TZP)                                          ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

tp
holds two integers:

1. the number of seconds that have elapsed since 00:00:00 January 1, 1970 GMT, plus

2. the number of microseconds that must be added to the preceding number to get the current time.

   In Pascal, **tp** is of type timeval, declared in the include file ailtypes.inc.

   In FORTRAN, **tp** is of type INTEGER TP(2).

tzp
holds two integers:

1. the time west of Greenwich in minutes.

2. the type of DST correction to apply.

   In Pascal, **tzp** is of type timezone, declared in the include file ailtypes.inc.

   In FORTRAN, **tzp** is of type INTEGER TZP(2).

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **GETTIMEOFDAY** system routine, which in these examples returns Greenwich time and the current time zone in the variables **tp** and **tzp** respectively.

Pascal

```
procedure gettimeofday1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  tp : timeval;
  tzp : timezone;

%include /usr/include/aildefs.inc

begin
  green := p_gettimeofday (tp, tzp);
  writeln ('Gettimeofday returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE GETTIMEOFDAY1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER, FGETTIMEOFDAY, TP(2), TZP(2), GREEN
      GREEN = FGETTIMEOFDAY (TP, TZP)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.38 GETUID, GETEUID, GETGID, GETEGID get a user or group identifier

Description
The four **GET** system calls described in this section return the real or effective ID of a user or group.

GETUID returns the ID of the real user of the calling process.

GETEUID returns the effective user ID of the calling process.

GETGID returns the real group ID of the calling process.

GETEGID returns the effective group ID of the calling process.

Syntax

Pascal

```
p_getuid;

p_geteuid;

p_getgid;

p_getegid;
```

FORTRAN

```
FGETUID ()

FGETEUID ()

FGETGID ()

FGETEGID ()
```

Parameters
These system calls have no parameters.

Return Values
The return value of each of the four calls is a particular ID (see description above).

In Pascal, the return value is of type ushrt

In FORTRAN, the return value is of type INTEGER*2

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **GETGID** system routine, which returns the real group ID of the calling process in the variable "blue".

Pascal

```
procedure getgid1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue : ushrt;

%include /usr/include/aildefs.inc

begin
  blue := p_getgid;
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE GETGID1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER*2 FGETGID, BLUE
      BLUE = FGETGID ()
      PRINT *, BLUE
      END
```

Notes

Because Pascal and FORTRAN lack the facilities for handling unsigned 4-byte integers, the programmer must convert parameter values that fall in the range

2 147 483 648 through 4 294 067 295

To use a parameter value in this range, subtract 4 294 067 296 from the parameter value before issuing the call (the result will always be negative).

2.39 GETXVERS get the UNIX version string

Description
The **GETXVERS** system call returns the UNIX version string.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_getxvers (xvers, length);                                      ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+

+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FGETXVERS (XVERS, LENGTH)                                        ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

xvers
is a pointer to the version string.

In Pascal, **xvers** is of type st80.

In FORTRAN, **xvers** is of type CHARACTER*80.

length
is the length of the version string.

In Pascal, **length** is of type integer.

In FORTRAN, **length** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **GETXVERS** system routine. After completion of the call, string "s" contains the UNIX version string.

Pascal

```
  procedure getxvers1;

  const
    %include /usr/include/ailpconsts.inc
  type
    %include /usr/include/ailtypes.inc
  var
```

```
    green: integer:
    s : st80;

  %include /usr/include/aildefs.inc

  begin
    green := p_getxvers (s, 10);
    writeln (s);
  end;
```

FORTRAN

```
      SUBROUTINE GETXVERS1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER, FGETXVERS, GREEN
      CHARACTER*80 S
      GREEN = FGETXVERS (S, 10)
      PRINT *, S
      END
```

2.40 IOCTL control the input and output of a device

Description

The **IOCTL** system call performs a variety of functions on both block- and character-special files (devices). (For information about available devices see *AIX Operating System Commands Reference* and *AIX Operating System Technical Reference*.)

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_ioctl (fildes, requst, arg);                                   ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FIOCTL (FILDES, REQUST, ARG)                                     ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

fildes
is the file descriptor of an opened device.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

requst
is either of two operations to be performed on the device specified by **fildes**. Both are defined in the Pascal and FORTRAN constants include files. They are as follows:

IOCTYP returns the device type associated with **fildes**. The device types are defined in the constant include files.

IOCINF stores device information specified by **fildes** in the buffer specified by **arg**.

In Pascal, **requst** is of type integer.

In FORTRAN, **requst** is of type INTEGER.

arg
is a data structure used to pass and receive values from the **IOCTL** routine.

In Pascal, **arg** is of type devptr.

Note: The Pascal type-definition file /usr/include/ailtypes.inc may have to be edited, and the data structure pointed to by devptr changed, to make that structure acceptable to the device specified in the call.

In FORTRAN, **arg** is a variable or array of type INTEGER.

Note: In FORTRAN, **arg** must be defined in the program to make it acceptable to the device specified in the call. This variable must be an array large enough to contain the structure returned by **IOCTL**. If the array is not large enough, the **IOCTL** will destroy the stackframe and cause a memory fault.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **IOCTL** system routine, which returns information about device /dev/lp in the Pascal record "green" or FORTRAN array "GREEN".

Pascal

```
procedure ioctl1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red, yellow : integer;
  green : devptr;

%include /usr/include/aildefs.inc

begin
  new (green);
  red := p_open ('/dev/lp', RDWR, 0);
  yellow := IOCINF;
  blue := p_ioctl (red, yellow, green);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE IOCTL1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FIOCTL, FOPEN, BLUE, GREEN(61), RED, YELL0W
      RED = FOPEN ('/dev/lp ', RDWR, 0)
      YELLOW = IOCINF
      BLUE = FIOCTL (RED, YELLOW, GREEN)
      PRINT *, BLUE
      END
```

2.41 KILL, KILLPG send a signal to a process or a process group

Description
The **KILL** system call sends a specified signal to a specified process. The **KILLPG** system call sends a specified signal to a specified process group.

The process receiving the signal is usually terminated as a result (see **SIGNAL** on page 2.89).

Note: Only the super-user may issue either call if the sending and receiving processes or groups have different effective user IDs.

Syntax

Pascal

```
p_kill (pid, sig);

p_killpg (pgrp, sig);
```

FORTRAN

```
FKILL (PID, SIG)

FKILLPG (PGRP, SIG)
```

Parameters

pid
is the ID of the process to which a signal is to be sent.

In Pascal, **pid** is of type integer.

In FORTRAN, **pid** is of type INTEGER.

pgrp
is the ID of the process group to which a signal is to be sent.

In Pascal, **pgrp** is of type integer.

In FORTRAN, **pgrp** is of type INTEGER.

sig
is the signal to be sent to the specified process. A process or process group may send signals to itself.

In Pascal, **sig** is of type integer.

In FORTRAN, **sig** is of type INTEGER.

Return Values
The value 0 is returned if the specified process is terminated; The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **KILL** system routine, which in these examples verifies the existence of the "special" root process, with process ID = 0.

Pascal

```
procedure kill1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue : integer;

%include /usr/include/aildefs.inc

begin
  blue := p_kill (0, 0);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE KILL1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FKILL, BLUE
      BLUE = FKILL (0, 0)
      PRINT *, BLUE
      END
```

2.42 LINK link to a file

Description
The **LINK** system call creates a link to an existing file.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_link (path1, path2);                                            ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FLINK (PATH1, PATH2)                                              ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

path1
is the name of the file to which a link is created.

In Pascal, **path1** is a string variable or constant of type st80.

In FORTRAN, **path1** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

path2
is the name of the new directory entry (link) to be created.

In Pascal, **path2** is a string variable or constant of type st80.

In FORTRAN, **path2** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **LINK** system routine, which in these examples creates a second link (/tmp/new) for the file /tmp/xxx. This will *not* be a copy of the file /tmp/xxx but an additional link to the existing file.

Pascal

```
  procedure link1;
```

```
const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  yellow : integer;
  blue, red : st80;

%include /usr/include/aildefs.inc

begin
  red := '/tmp/xxx';
  blue := '/tmp/new';
  yellow := p_link (red, blue);
  writeln (yellow);
end;
```

FORTRAN

```
      SUBROUTINE LINK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FLINK, YELLOW
      CHARACTER*80 BLUE, RED
      RED = '/tmp/xxx '
      BLUE = '/tmp/new '
      YELLOW = FLINK (RED, BLUE)
      PRINT *, YELLOW
      END
```

2.43 LISTEN "listen" for a connection to a socket

Description

The **LISTEN** system call specifies a maximum queue length for the number of pending connections to a specified socket. The call applies only to sockets of type SOCK_STREAM.

Syntax

```
+--- Pascal -----------------------------------------------------+
¦                                                                ¦
¦   p_listen (s, backlog);                                       ¦
¦                                                                ¦
¦                                                                ¦
+----------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------+
¦                                                                ¦
¦   FLISTEN (S, BACKLOG)                                         ¦
¦                                                                ¦
¦                                                                ¦
+----------------------------------------------------------------+
```

Parameters

s
is the descriptor of the socket that was created by a **SOCKET** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

backlog
specifies the maximum length of the queue of pending connections.

In Pascal, **backlog** is of type integer.

In FORTRAN, **backlog** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **LISTEN** system routine after the **backlog** parameter has been set to 1.

Pascal

```
  procedure listen1;

  const
    %include /usr/include/ailpconsts.inc
  type
```

```
  %include /usr/include/ailtypes.inc
var
  backlog, namelen, s : integer;
  addr : sockaddrptr;

%include /usr/include/aildefs.inc

begin
  s := p_socket (PF_UNIX, SOCK_DGRAM, 0);
  if (s = -1) then showerror;
  new (addr);
  addr^.sa_data := 'socket';
  addr^.sa_family := PF_UNIX;
  green := p_unlink ('socket');
  green := p_bind (s, addr, 16);
  backlog := 1;
  green := p_listen (s, backlog);
  if (green <> -1) then
    writeln ('Listen returned : OK')
  else
    writeln ('Listen returned : ERROR');
  if (green = -1) then showerror;
  green := p_shutdown (s, 2);
end;
```

FORTRAN

```
      SUBROUTINE LISTEN1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FLISTEN, FBIND, FSHUTDOWN, FSOCKET, FUNLINK
      INTEGER BACKLOG, S, GREEN
      CHARACTER*80 NAME
      S = FSOCKET (PFUNIX, SKSTRM, 0)
      NAME = 'BNAME '
      GREEN = FUNLINK (NAME)
      GREEN = FBIND (S, SKSTRM, NAME, 16)
      BACKLOG = 1
      GREEN = FLISTEN (S, BACKLOG)
      IF (GREEN .EQ. -1) THEN
         PRINT *, 'LISTEN : ERROR'
         CALL ERRORS
      ELSE
         PRINT *, 'LISTEN : OK'
      ENDIF
      GREEN = FUNLINK ('BNAME ')
      GREEN = FSHUTDOWN (S, 2)
      END
```

2.44 LOCKF lock or unlock a region of a file

Description

The **LOCKF** system call locks and unlocks regions of an open file. It is used to synchronize simultaneous access to a specified open file by multiple processes. Only one process at a time can maintain a "lock" on a region of a file. The **LOCKF** system call can invoke either of two kinds of lock: (1) enforced or (2) advisory.

1. When a process holds an *enforced* lock on a region of a file:

   a. no other process can access that region with read or write system calls; and

   b. **CREAT** and **OPEN** are prevented from truncating the file.

2. When a process holds an *advisory* lock on a region of a file:

   a. no other process can lock that region or an overlapping region with the **LOCKF** call; and

   b. the **CREAT**, **OPEN**, **READ**, and **WRITE** call are not affected, which means that a process itself must issue a **LOCKF** call in order to make advisory locks effective.

Note: To select enforced locking, the ENFMT access mode of the specified file must be set. Otherwise, locking is optional. Thus a given file can have enforced locks or advisory locks but not both.

Warning: Buffered I/O does not work properly with file locking.

Syntax

Pascal

```
p_lockf (fildes, request, size);
```

FORTRAN

```
FLOCKF (FILDES, REQUEST, SIZE)
```

Parameters

fildes
is the descriptor of an open file that has been returned by a **CREAT**, **DUP**, **DUP2**, **OPEN**, or **PIPE** system call.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

requst
can be a constant or a variable. The options are defined as constants in the Pascal and FORTRAN constants include files.

F_ULOCK unlocks a previously locked region in the file.

F_LOCK locks the region for exclusive use.

F_TLOCK determines whether another process has locked the specified region and, if not, locks the region.

F_TEST determines whether another process has already locked a region.

Note: In FORTRAN, the underscore is omitted (for example, "FULOCK").

In Pascal, **requst** is of type integer.

In FORTRAN, **requst** is of type INTEGER.

size
can be a constant or a variable and it defines the number of bytes being locked or unlocked. Unallocated "holes" in the file can also be locked (see **FCLEAR** on page 2.22).

In Pascal **size** is of type integer.

In FORTRAN, **size** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **LOCKF** system routine, which locks an open file "forward" 1000 bytes.

Pascal

```
procedure lockf1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;

%include /usr/include/aildefs.inc

begin
  red := p_open ('/usr/include/ailtypes.inc', RDONLY, 0);
  blue := p_lockf (red, F_LOCK, 1000);
  writeln (blue);
end;
```

FORTRAN

```
SUBROUTINE LOCKF1
INCLUDE (/usr/include/ailfconsts.inc)
INTEGER FLOCKF, FOPEN, BLUE, RED
RED = FOPEN ('/usr/include/ailtypes.inc ', RDONLY, 0)
BLUE = FLOCKF (RED, FLOCK, 1000)
PRINT *, BLUE
END
```

2.45 LSEEK set a read or write pointer

Description

The **LSEEK** system call sets a read or write pointer in a specified file that has been opened for reading or writing.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_lseek (fildes, offset, whence);                                ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FLSEEK (FILDES, OFFSET, WHENCE)                                  ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

fildes
is the descriptor of the file to be read from or written to; it is returned by a **CREAT**, **DUP**, **DUP2**, **FCNTL**, or **OPEN** system call.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

offset
is a value (number of bytes) used in combination with the **whence** parameter to position the pointer in the file.

In Pascal, **offset** is of type integer.

In FORTRAN, **offset** is of type INTEGER.

whence
specifies how the **offset** value will be used to position the file pointer of **fildes**.

SEEK_SET the pointer will be set to the value of **offset**.

SEEK_CUR the pointer will be set to the value of the current location plus the **offset** value.

SEEK_END the pointer will be set to the value of the **offset** number of bytes plus the size of the file.

Note: In FORTRAN, the underscore is omitted (for example, "SEEKSET").

In Pascal, **whence** is of type integer.

In FORTRAN, **whence** is of type INTEGER.

Return Values

The return value is the new location of the file pointer as measured in bytes from the beginning of the file. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **LSEEK** system routine, which moves the file pointer to the 200-byte mark of the open file specified in the call. The return value in "yellow" should in this case equal the offset of 200.

Pascal

```
procedure lseek1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red, yellow : integer;

%include /usr/include/aildefs.inc

begin
  red := p_open ('/usr/include/ailtypes.inc', RDONLY, 0);
  blue := SEEK_SET;
  green := 200;
  yellow := p_lseek (red, green, blue);
  writeln (yellow);
end;
```

FORTRAN

```
      SUBROUTINE LSEEK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FLSEEK, FOPEN, BLUE, GREEN, RED, YELLOW
      RED = FOPEN ('/usr/include/ailtypes.inc ', RDONLY, 0)
      BLUE = SEEKSET
      GREEN = 200
      YELLOW = FLSEEK (RED, GREEN, BLUE)
      PRINT *, YELLOW
      END
```

2.46 MKDIR create a directory

Description
The **MKDIR** system call creates a new directory.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_mkdir (path, mode);                                            ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FMKDIR (PATH, MODE)                                              ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

path
is the name of the new directory.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

mode
is the mask for the read, write, and execute (rwx) flags for owner, group, and others. The low-order 9 bits in **mode** are modified by the file-mode-creation mask of the process. All bits set in the creation mask are cleared. For more information, see page 2.105)

In Pascal, **mode** is of type integer.

In FORTRAN, **mode** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **MKDIR** system routine. The directory specified in the call is /usr/games, which becomes the new directory. The return value of the call is in the variable "folio".

Pascal

```
  procedure mkdir1;
```

```
const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  folio : integer;
  red : st80;

%include /usr/include/aildefs.inc

begin
  red := '/usr/games';
  folio := p_mkdir (red, 128);
  writeln (folio);
end;
```

FORTRAN

```
      SUBROUTINE MKDIR1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER  FMKDIR, FOLIO
      CHARACTER*80 RED
      RED = '/usr/games '
      FOLIO = FMKDIR (RED, 128)
      PRINT *, FOLIO
      END
```

2.47 MKNOD create a directory or special file

Description

The **MKNOD** system call creates a new regular file, special file, or directory; specifies an access mode that includes directory special-file bits; and initializes the first pointer of the i-node.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_mknod (path, mode, dev);                                         ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FMKNOD (PATH, MODE, DEV)                                           ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

path
is the name of the new file or directory.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

mode
is the access mode of the new file and includes special bits and directory bits. It is constructed by logically ORing the values of the access-attribute bits of **CHMOD** (see page 2.10.) with one of the following values, which define the file type:

S_IFDIR directory

S_IFCHR character special file

S_IFMPX multiplexed character special file the value of the low-order bit of **arg**.

S_IFBLK block special file

S_IFREG regular data file

S_IFIFO FIFO special file

The protection bits of the mode are modified by the process mode mask (see **UMASK** on page 2.105)

In Pascal, **mode** is of type integer.

In FORTRAN, **mode** is of type INTEGER.

dev

initializes the first block pointer of the i-node. For ordinary files and directories, **dev** is usually zero. In the case of a special file, **dev** specifies the file to be created. (For information on special-file bits, see *AIX Technical Reference*.)

In Pascal, **dev** is of type integer.

In FORTRAN, **dev** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGE

Examples

The Pascal procedure and FORTRAN subroutine that follow call **MKNOD** system routine, which in these examples creates a file (/tmp/junk). The value of **mode** ("blue") specifies a text file with read and write privileges for the owner of the file.

Pascal

```
procedure mknod1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;
  yellow : st80;

%include /usr/include/aildefs.inc

begin
  yellow := '/tmp/junk';
  blue := 33152;
  red := p_mknod (yellow, blue, 0);
  writeln (red);
end;
```

FORTRAN

```
      SUBROUTINE MKNOD1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FMKNOD, BLUE, RED
      CHARACTER*80 YELLOW
      YELLOW = '/tmp/junk '
      BLUE = 33152
      RED = FMKNOD (YELLOW, BLUE, 0)
```

```
PRINT *, RED
END
```

2.48 MOUNT, UMOUNT mount or unmount a file system

Description

The **MOUNT** system call mounts a removable file system on a block-structured special file, names a new root file for that file system, and specifies whether the system is write enabled or write protected.

The **UMOUNT** system call unmounts a removable file system: the associated root file is replaced by the default version, any pending I/O for the unmounted system is completed, and the system itself is marked clean.

Note: Only users with an effective user ID of super-user may issue thesecall.

Syntax

Pascal

```
p_mount (dev, dir, mflag);

p_umount (dev, flag);
```

FORTRAN

```
FMOUNT (DEV, DIR, MFLAG)

FUMOUNT (DEV, FLAG)
```

Parameters

dev

specifies the device on which the file system is to be mounted or from which it is to be unmounted.

In Pascal, **dev** is a string variable or constant of type st80.

In FORTRAN, **dev** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

dir

is used only with **MOUNT**. It is the name of the directory of the file system that is to be mounted. The file specified by **dir** must exist and it must be a directory unless the root file of the mounted file system is not a directory.

In Pascal, **dir** is a string variable or constant of type st80.

In FORTRAN, **dir** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

mflag

is used only with **MOUNT**. The least significant bit specifies whether the file system is write enabled or not.

Note: For possible values of this parameter, see Appendix B.

In Pascal, **mflag** is of type integer.

In FORTRAN, **mflag** is of type INTEGER.

flag
if set to a non-zero value, forces the unmounting of the file system even if it contains open files.

In Pascal, **flag** is of type integer.

In FORTRAN, **flag** is of type INTEGER.

Return Values
MOUNT returns the value 0 upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

UMOUNT returns the value 0 upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **UMOUNT** system routine. In these examples, the call instructs the routine to unmount a device.

Pascal

```
procedure umount1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  yellow : integer;
  blue : st80;

%include /usr/include/aildefs.inc

begin
  blue := '/dev/hd9';
  yellow := p_umount (blue, 0);
  writeln (yellow);
end;
```

FORTRAN

```
      SUBROUTINE UMOUNT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FUMOUNT, YELLOW
      CHARACTER*80 BLUE
```

```
BLUE = '/dev/hd9 '
YELLOW = FUMOUNT (BLUE, 0)
PRINT *, YELLOW
END
```

2.49 MSGCTL invoke message-control operations

Description

The **MSGCTL** system call invokes any of three message-control operations, including the storing and setting of the values in a specified message queue.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_msgctl (msqid, cmd, buf);                                        ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FMSGCTL (MSQID, CMD, BUF)                                          ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

msqid
is the identifier of a message queue created by a previous **MSGGET** call (see page 2.50). The value of **msqid** is returned by **MSGGET**.

In Pascal, **msqid** is of type integer.

In FORTRAN, **msqid** is of type INTEGER.

cmd
specifies the operation to be performed, which can be any of the options in the following list.

Note: Each option number corresponds to a mnemonic (shown in parentheses) defined in the Pascal and FORTRAN constants include files.

IPCRMD removes the message-queue identifier and its associated data structure from the operating system and destroys the associated message.

IPCSET sets the value of the following fields and the data structure associated with **msqid** to the corresponding value found in the data structure pointed to by **buf**.

In Pascal these fields are:

msg_perm.uid
msg_perm.gid
msg_perm.mode
msg_qbytes

In FORTRAN the corresponding fields are:

MSQID(1)
MSQID(2)
MSQID(5)
MSQID(12)

Note: Only a process whose effective user ID is super-user can raise the value of msg_qbytes.

IPCSTT takes the current value of each field of the data structure associated with **msqid** and stores it in the structure pointed to by the **buf** parameter (see below).

Note: The first two options can be used only when the effective user ID is equal to the super-user ID or to the value of msqid_ds@.msg_perm.uid in Pascal or MSQID(1) in FORTRAN.

In Pascal, **cmd** is of type integer.

In FORTRAN, **cmd** is of type INTEGER.

buf

points to a record of type msqid_ds. The values stored or set in this record are the current values of the data structure associated with **msqid**.

In Pascal, **buf** is of type mdsptr.

In FORTRAN, **buf** is an array(17) of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **MSGCTL** system routine. The value of the first parameter of this call is the return value of **MSGGET**. (The value of the first parameter of **MSGGET** is the return value of the **ftok** system subroutine; see **Notes** at the end of this section.) The variable "pink" specifies the option that stores the values associated with the **msqid** parameter "green" in the data structure pointed to by "yellow".

Pascal

```
procedure msgctl1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, pink, red : integer;
  orange : st80;
  brown : char;
```

```
    yellow : mdsptr;

  %include /usr/include/aildefs.inc

  begin
    new (yellow);
    brown := 'm';
    orange := '/usr/include/ailtypes.inc';
    blue := IPCCRT + IRUSR;
    red := p_ftok (orange, brown);
    green := p_msgget (red, blue);
    pink := IPCSTT;
    red := p_msgctl (green, pink, yellow);
    writeln (red);
  end;
```

FORTRAN

```
        SUBROUTINE MSGCTL1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FFTOK, FMSGGET, FMSGCTL, BLUE, GREEN, PINK, RED, YELLOW(17)
        CHARACTER*80 ORANGE
        CHARACTER BROWN
        BROWN = 'm'
        ORANGE = '/usr/include/ailtypes.inc '
        BLUE = IPCCRT + IRUSR
        RED = FFTOK (ORANGE, BROWN)
        GREEN = FMSGGET (RED, BLUE)
        PINK = IPCSTT
        RED = FMSGCTL (GREEN, PINK, YELLOW)
        PRINT *, RED
        END
```

2.50 MSGGET get or create a message queue

Description

The **MSGGET** system call gets a specified message queue identifier associated with the specified key parameter. **MSGGET** can also create the identifier and message queue if they do not already exist.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_msgget (key, msgflg);                                            ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+

+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FMSGGET (KEY, MSGFLG)                                              ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

key
determines which identifier and associated data structure to use. The **key** parameter may be equal to **0** (IPCPVT); or **key** can be an IPC key constructed by a call to the **ftok** system subroutine.

In Pascal, **key** is of type integer.

In FORTRAN, **key** is of type INTEGER.

msgflg
specifies a set of conditions (options) governing the creation of the message-queue data structure and the accessibility of the message queue. The parameter value is that of one of the following options or is constructed from two or more of those options by logical ORing. The options are defined as constants in the Pascal and FORTRAN constants include files.

IPCCRT creates the message-queue data structure when it does not exist.

IPCEXL causes **MSGGET** to fail when IPCCRT is set and the message-queue data structure exists.

IRUSR permits the process that owns the message-queue data structure to read it.

IWUSR permits the process that owns the message-queue data structure to modify it.

IRGRP permits the group associated with the message-queue data structure to read it.

IWGRP permits the group associated with the message-queue data structure to modify it.

IROTH permits others to read the message-queue data structure.

IWOTH permits others to modify the message-queue data structure.

In Pascal, **msgflg** is of type integer.

In FORTRAN, **msgflg** is of type INTEGER.

Return Values

A message-queue identifier is returned upon successful completion of the call, and the data structure (msqid_ds; see Appendix C) associated with the new identifier is initialized. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **MSGGET** system routine. (The value of the first parameter of the call is the return value of the **ftok** system subroutine. The value assigned to the parameter "blue" specifies the creation of a message queue for the process (if one does not already exist) and gives the user read access to it.

Pascal

```
procedure msgget1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red : integer;
  orange : st80;
  brown : char;

%include /usr/include/aildefs.inc

begin
  brown := 'm';
  orange := '/usr/include/ailtypes.inc';
  blue := IPCCRT + IRUSR;
  red := p_ftok (orange, brown);
  green := p_msgget (red, blue);
  writeln (green);
end;
```

FORTRAN

```
      SUBROUTINE MSGGET1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FMSGGET, FFTOK, BLUE, GREEN, RED
      CHARACTER*80 ORANGE
      CHARACTER BROWN
      BROWN = 'm'
      ORANGE = '/usr/include/ailtypes.inc '
```

```
BLUE = IPCCRT + IRUSR
RED = FFTOK (ORANGE, BROWN)
GREEN = FMSGGET (RED, BLUE)
PRINT *, GREEN
END
```

2.51 MSGRCV, MSGXRCV read and store a message

Description

Both of the **MSG** system calls read a message from a specified queue and place it in a structure specified in the call.

In addition, **MSGXRCV** will return the following items of information:

the time the message was sent

the sender's effective user ID

the sender's effective group ID

the sender's node ID

the sender's process ID

Syntax

Pascal

```
p_msgrcv (msqid, msgp, msgsz, msgtyp, msgflg);

p_msgxrcv (msqid, msgpt, msgsz, msgtyp, msgflg);
```

FORTRAN

```
FMSGRCV (MSQID, MSGP1, MSGP2, MSGSZ, MSGTYP, MSGFLG)

FMSGXRCV (MSQID, MSGPT1, MSGPT2, MSGSZ, MSGTYP, MSGFLG)
```

Parameters

msqid
is a message-queue identifier containing the message to be read.

In Pascal, **msqid** is of type integer.

In FORTRAN, **msqid** is of type INTEGER.

msgp
points to the record msgbuf, in which a type identifier and the message will be stored. This message is read from the queue specified by **msqid**. The **msgp** parameter is used only in the **MSGRCV** call.

In Pascal, **msgp** is of type mbufptr.

In FORTRAN, **msgp** is sent as two parameters:

- **msgp1** is of type INTEGER.
- **msgp2** is of type CHARACTER*80.

msgpt
points to the extended message receive buffer (msgxbuf), in which the message time, sender information, type identifier, and message will be stored. This message is read from the queue specified by **msqid**. The **msgpt** parameter is used only in the **MSGXRCV** call.

In Pascal, **msgpt** is of type msgxptr.

In FORTRAN, **msgpt** is sent as two parameters:

- **msgpt1** is an array(6) of type INTEGER.

- **msgpt2** is of type CHARACTER*80.

msgsz
is a constant or variable that specifies the length of the message in bytes. The maximum size of **msgsz** is 80 characters.

In Pascal, **msgsz** is of type integer.

In FORTRAN, **msgsz** is of type INTEGER.

msgtyp
is a constant or variable that specifies the type of the message to be read.

In Pascal, **msgtyp** is of type integer.

In FORTRAN, **msgtyp** is of type INTEGER.

msgflg
specifies the operation to be performed when the desired message is in the queue and when it is not. The value assigned to **msgflg** is that of one or more of the following:

IPCNER
truncates the message when it is longer than the number of bytes specified by **msgsz**.

IPCNWT
specifies the operation to be performed when the desired message is not in the queue.

In Pascal, **msgflg** is of type integer.

In FORTRAN, **msgflg** is of type INTEGER.

Return Values
A value equal to the number of bytes stored in mtext (of msgbuf or msgxbuf) is returned upon successful completion of a call, and the data structure associated with the message-queue identifier is modified as follows:

msg_qnum is decremented by 1

msg_lpid is set equal to the process ID of the calling process

msg_rtime is set equal to the current time

The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **MSGRCV** system routine. The value of the first parameter of this call is the return value of **MSGGET**. (The value of the first parameter of **MSGGET** is the return value of the **ftok** system subroutine; see **Notes** at the end of this section.) The variable "orange" specifies the maximum length of the message. The value printed out is the number of bytes received from a message.

Pascal

```
procedure msgrcv1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, grey, orange, pink, purple, red : integer;
  white : st80;
  brown : char;
  yellow : mbufptr;

%include /usr/include/aildefs.inc

begin
  new (yellow);
  brown := 'w';
  white := '/usr/include/ailtypes.inc';
  blue := 0;
  green := IPCNER;
  orange := 50;
  pink := IPCCRT + IRUSR;
  purple := p_ftok (white, brown);
  red := p_msgget (purple, pink);
  grey := p_msgrcv (red, yellow, orange, blue, green);
  writeln (grey);
end;
```

FORTRAN

```
      SUBROUTINE MSGRCV1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FMSGRCV, FMSGGET, FFTOK, BLUE, GREEN, GREY, ORANGE
      INTEGER PINK, PURPLE, RED, VIOLET
      CHARACTER*80 WHITE, YELLOW
      CHARACTER BROWN
      BROWN = 'w'
      WHITE = '/usr/include/ailtypes.inc '
      BLUE = 0
      GREEN = IPCNER
```

```
ORANGE = 50
PINK = IPCCRT + IRUSR
PURPLE = FFTOK (WHITE, BROWN)
RED = FMSGGET (PURPLE, PINK)
GREY = FMSGRCV (RED, VIOLET, YELLOW, ORANGE, BLUE, GREEN)
PRINT *, GREY
END
```

2.52 MSGSND send a message to a queue

Description

The **MSGSND** system call sends a message to a specified queue.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_msgsnd (msqid, msgp, msgsz, msgflg);                            ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FMSGSND (MSQID, MSGP1, MSGP2, MSGSZ, MSGFLG)                      ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

msqid
is a message-queue identifier to which a message is to be sent.

In Pascal, **msqid** is of type integer.

In FORTRAN, **msqid** is of type INTEGER.

msgp
is the pointer to the record msgbuf, which contains the message to be sent.

In Pascal, **msgp** is of type mbufptr.

In FORTRAN, **msgp** is sent as two parameters:

- **msgp1** is of type INTEGER.

- **msgp2** is of type CHARACTER*80.

msgsz
is a constant or variable that specifies the length of the message in bytes. The maximum value of **msgsz** is 80.

In Pascal, **msgsz** is of type integer.

In FORTRAN, **msgsz** is of type INTEGER.

msgflg
specifies the action taken when either of the following conditions prevents the message from being sent:

the number of bytes already in the queue is equal to the number specified by msg_qbytes.

the total number of messages in all queues in the system is equal to the system-imposed limit.

In Pascal, **msgflg** is of type integer.

In FORTRAN, **msgflg** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call, and the data structure associated with the message-queue identifier is modified as follows:

msg_qnum is incremented by 1

msg_lspid is set equal to the process ID of the calling process

msg_stime is set equal to the current time

The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **MSGSND** system routine. The value of the first parameter of this call is the return value of **MSGGET**. (The value of the first parameter of **MSGGET** is the return value of the **ftok** system subroutine; see Appendix E.) The variable "orange" specifies the length of the message.

Pascal

```
procedure msgsnd1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc

var
  blue, grey, orange, pink, purple, red : integer;
  white : st80;
  brown : char;
  yellow : mbufptr;

%include /usr/include/aildefs.inc

begin
  new (yellow);
  brown := 'w';
  white := '/usr/include/ailtypes.inc';
  blue := IPCNWT;
  orange := 27;
  pink := IPCCRT + IRUSR + IWUSR;
  yellow@.mtype := 1;
  yellow@.mtext := 'This is 1 test for messages';
  purple := p_ftok (white, brown);
  red := p_msgget (purple, pink);
  grey := p_msgsnd (red, yellow, orange, blue);
  writeln (grey);
end;
```

FORTRAN

```
SUBROUTINE MSGSND1
INCLUDE (/usr/include/ailfconsts.inc)
INTEGER FMSGSND, FMSGGET, FFTOK, BLUE, GREY, ORANGE, PINK
INTEGER PURPLE, RED, YELLOW
CHARACTER*80 WHITE, VIOLET
CHARACTER BROWN
BROWN = 'w'
WHITE = '/usr/include/ailtypes.inc '
BLUE = IPCNWT
ORANGE = 27
PINK = IPCCRT + IRUSR + IWUSR
YELLOW = 1
VIOLET = 'This is 1 test for messages'
PURPLE = FFTOK (WHITE, BROWN)
RED = FMSGGET (PURPLE, PINK)
GREY = MSGSND (RED, YELLOW, VIOLET, ORANGE, BLUE)
PRINT *, GREY
END
```

2.53 NICE set a process priority

Description

The **NICE** system call assigns a new CPU priority to a process by adding a specified value to its current **NICE** value.

If this value results in a priority number outside the valid range, the **NICE** routine will reset the priority to the nearest limit.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_nice (incr);                                                   ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FNICE (INCR)                                                     ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

incr
is a value that--when added to the priority number of the current process--determines the new priority number of the current process.

In Pascal, **incr** is of type integer.

In FORTRAN, **incr** is of type INTEGER.

Return Values

The new **NICE** value minus 20 is the value returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **NICE** system routine. The priority number of the current process is increased by 5, thereby lowering the priority. The **incr** parameter is specified with the variable "red". The return value printed is the new priority value minus 20.

Pascal

```
procedure nice1;

const
   %include /usr/include/ailpconsts.inc
type
```

```
     %include /usr/include/ailtypes.inc
  var
     blue, red : integer;

  %include /usr/include/aildefs.inc

  begin
     red := 5;
     blue := p_nice (red);
     writeln (blue);
  end;
```

FORTRAN

```
        SUBROUTINE NICE1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FNICE, BLUE, RED
        RED = 5
        BLUE = FNICE (RED)
        PRINT *, BLUE
        END
```

2.54 OPEN open a file for reading or writing

Description

The **OPEN** system call opens a specified file for reading or writing or both, depending on the access mode specified in the call.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_open (path, oflag, mode);                                         ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FOPEN (PATH, OFLAG, MODE)                                           ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

path
is the name of the file to be opened.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

oflag
specifies one or a combination of the options listed below. The parameter value is that of one of the following options or is constructed from two or more of those options by logical ORing. The options are defined as constants in the Pascal and FORTRAN constants include files (see Appendixes).

Note: The RDONLY, WRONLY, and RDWR values *cannot* be logically ORed together.

RDONLY opens the file for reading.

WRONLY opens the file for writing.

RDWR opens the file for both reading and writing.

NDELAY open without delay. This flag may affect subsequent reads and writes.

APPEND sets the file pointer to the end of the file prior to each write.

CREATE has no effect if the file specified by **path** exists. However, if the specified file does not exist, the file owner's ID and the files's group ID are set to the effective user ID of the process; and the access mode is set to **mode**.

TRUNC truncates the file length to zero.

EXCL when this option and **CREATE** are set, **OPEN** will fail if the file exists.

In Pascal, **oflag** is of type integer.

In FORTRAN, **oflag** is of type INTEGER.

mode
is used with the **CREATE** value of **oflag**.

Note: For more information on the **mode** parameter, see **CHMOD** on page 2.10 and **STATX** on page 2.98.

Return Values
The return value is the file descriptor of the opened file. This file descriptor will be needed for subsequent input-output operations. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **OPEN** system routine, which returns a file descriptor in the variable "red". If the call is successful, the number printed out is a valid file descriptor.

Pascal

```
procedure open1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  red : integer;

%include /usr/include/aildefs.inc

begin
  red := p_open ('/usr/include/ailtypes.inc', RDONLY, 0);
  writeln(red);
end;
```

FORTRAN

```
      SUBROUTINE OPEN1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FOPEN, RED
      CHARACTER*80 BLUE
      BLUE = '/usr/include/ailtypes.inc '
      RED = FOPEN (BLUE, RDONLY, 0)
      PRINT *, RED
      END
```

2.55 PAUSE wait for a signal

Description
The **PAUSE** system call suspends the execution of a process until it receives a signal.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_pause;                                                           ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FPAUSE ()                                                          ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters
This system call has no parameters.

Return Values
There is no return value from a successful completion of **PAUSE**. The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **PAUSE** system routine, which suspends the calling process until the signal from the **ALARM** call is received.

Pascal

```
procedure pause1;

const
  %include /usr/include/ailfconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue : integer;
  green, red : usign;

%include /usr/include/aildefs.inc

begin
  red := 20;
  green := p_alarm (red);
  writeln (green);
  blue := p_pause
end;
```

FORTRAN

```
      SUBROUTINE PAUSE1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FPAUSE, FALARM, BLUE, GREEN, RED
      RED = 20
      GREEN = FALARM (RED)
      PRINT *, GREEN
      BLUE = FPAUSE ()
      END
```

2.56 PIPE create an interprocess channel

Description

The **PIPE** system call creates an interprocess communication mechanism--called a "pipe" or "channel"--that allows the passing of data between processes. After a pipe has been set up, two or more cooperating processes (created by subsequent **FORK** routines) can pass data to one another with **READ** and **WRITE** calls.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_pipe (fildes);                                                 ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FPIPE (FILDES)                                                   ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

fildes
is an array of two file descriptors, both of which are returned by a **PIPE** call. The first element of the array holds the file descriptor for the read end of the pipe; the second element holds the file descriptor for the write end of the pipe.

In Pascal, **fildes** is a variable of type piparray.

In FORTRAN, **fildes** is an array(2) of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails (for example, if too many files are open).

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **PIPE** system routine. A pipe is created between two files whose descriptors are returned: the read end of the pipe is returned in the first element of the array "red" and the write end is returned in the second.

Pascal

```
  procedure pipe1;

  const
    %include /usr/include/ailpconsts.inc
```

```
type
  %include /usr/include/ailtypes.inc
var
  blue : integer;
  red : piparray;

%include /usr/include/aildefs.inc

begin
  blue := p_pipe (red);
  writeln (blue);
  writeln (red[1]);
  writeln (red[2]);
end;
```

FORTRAN

```
      SUBROUTINE PIPE1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FPIPE, BLUE, RED(2)
      BLUE = FPIPE (RED)
      PRINT *, BLUE
      PRINT *, RED(1)
      PRINT *, RED(2)
      END
```

2.57 PLOCK lock or unlock a process, text, or data

Description
The **PLOCK** system call allows the calling process to lock or unlock its text segment (text lock), its data segment (data lock), or both (process lock) into memory. Locked segments are "pinned" in memory and are unaffected by paging.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_plock (op);                                                       ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FPLOCK (OP)                                                         ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

op
is a constant or variable that specifies one of four options:

UNLOCK remove the locks.

PROCLOCK lock text and data segments into memory.

TXTLOCK lock text segment into memory.

DATLOCK lock data segment into memory.

In Pascal, **op** is of type integer.

In FORTRAN, **op** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **PLOCK** system routine. The value of the **op** parameter ("red") specifies that the routine lock the current text segment into memory.
Pascal

```
  procedure plock1;
```

```
const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;

%include /usr/include/aildefs.inc

begin
  red := TXTLOCK;
  blue := p_plock (red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE PLOCK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FPLOCK, BLUE, RED
      RED = TXTLOCK
      BLUE = FPLOCK (RED)
      PRINT *, BLUE
      END
```

2.58 PROFIL generate an execution-time profile

Description

The **PROFIL** system call generates a histogram of periodically sampled values of the program counter of the calling process.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_profil (buf, bufsiz, offset, scale);                            ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+

+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FPROFIL (BUF, BUFSIZ, OFFSET, SCALE)                              ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

buf
for any value of **bufsiz** except -1, points to an area of memory, and its length in bytes is given by **bufsiz**. If the value of **bufsiz** is -1, then the parameters **offset** and **scale** are ignored and **buf** points to an array of "prof" structures (declared in ailtypes.inc and available only in Pascal).

In Pascal, **buf** is of type intptr.

In FORTRAN, **buf** is of type INTEGER*2.

bufsiz
specifies the size of the buffer in bytes. A value of 0 (zero) renders the routine ineffective. (See **Notes**.)

In Pascal, **bufsiz** is of type usign.

In FORTRAN, **bufsiz** is of type INTEGER.

offset
specifies the value to be subtracted from the program counter. (See **Notes**.)

In Pascal, **offset** is of type usign.

In FORTRAN, **offset** is of type INTEGER.

scale
specifies the value by which the quantity (program count - **offset**) is multiplied before the value in **buf** is incremented. (See **Notes**.)

In Pascal, **scale** is of type usign.

In FORTRAN, **scale** is of type INTEGER.

Return Values

There is no return value from a successful **PROFIL** call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **PROFIL** system routine. With the values assigned in the example, all instructions will be mapped to the area in memory pointed to by the variable "yellow".

Pascal

```
procedure profil1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
  yellow : shrtptr;
  blue, indigo, violet : usign;

%include /usr/include/aildefs.inc

begin
   new (yellow);
  blue := 2;
  indigo := 0;
  violet := 1;
  green := p_profil (yellow, blue, indigo, violet)
end;
```

FORTRAN

```
      SUBROUTINE PROFIL1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FPROFIL, BLUE, GREEN, INDIGO, VIOLET, YELLOW*2
      BLUE = 2
      INDIGO = 0
      VIOLET = 1
      GREEN = FPROFIL (YELLOW, BLUE, INDIGO, VIOLET)
      END
```

Notes
Because Pascal and FORTRAN lack the facilities for handling unsigned 4-byte integers, the programmer must convert parameter values that fall in the range

2 147 483 648 through 4 294 067 295

To use a parameter value in this range, subtract 4 294 067 296 from the parameter value before issuing the call (the result will always be negative).

2.59 PTRACE trace the execution of a child process

Description

The **PTRACE** routine enables a parent process to control the execution of a child process and to examine and change its memory image. The routine is used primarily for breakpoint debugging.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                     ¦
¦   p_ptrace (requst, pid, addr, data, buff);                         ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                     ¦
¦   FUNCTION FPTRACE (REQUST, PID, ADDR, DATA, BUFF)                  ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

requst
is a variable that specifies a trace operation (see *AIX Technical Reference*).

In Pascal, **requst** is of type integer.

In FORTRAN, **requst** is of type INTEGER.

pid
is a variable that contains the process ID of the traced process. This process must be an immediate child of the tracing process.

In Pascal, **pid** is of type integer.

In FORTRAN, **pid** is of type INTEGER.

addr
Depending on the value of **requst**, this parameter

- points to an area where data is returned; or
- indicates a register whose value is to be modified or returned; or
- points to a block of data (in the child process) to be read from or written to.

In Pascal, **addr** is of type intptr.

In FORTRAN, **addr** is of type INTEGER.

data
when it is not ignored, usually holds data for requests that write to the memory image of the traced process.

In Pascal, **data** is of type integer.

In FORTRAN, **data** is of type INTEGER.

buff
is a pointer to a block of data (for any **requst** that requires a buffer).

In Pascal, **buff** is of type intptr.

In FORTRAN, **buff** is of type INTEGER.

Return Values
For the values that are returned by **PTRACE**, see the descriptions of the arguments to the **requst** parameter. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **PTRACE** system routine. A child process is created by a **FORK** system call. The child process then calls **PTRACE**, requesting that it be traced by the parent (**requst** = 0). The parent process waits for a signal from the child and then calls **PTRACE**, which returns the value of register 2 used by the child process (**requst** = 11).

Pascal

```
procedure ptrace1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, orange, red, yellow : integer;

%include /usr/include/aildefs.inc

begin
  green := p_fork;
  if green = 0 then
     begin
        orange := p_alarm (1);
        orange := p_ptrace (0, 0, nil, 0, nil);
        for blue := 1 to 10 do
           for red := 1 to 100 do
              write ('z');
        writeln;
     end

  else
     begin
        orange := p_wait (yellow);
        writeln ('return from wait   ', orange);
        orange := p_ptrace (11, green, nil, 2, nil);
        writeln ('register two contains   ', orange);
```

```
      end
   end;
```

FORTRAN

```
      SUBROUTINE PTRACE1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FPTRACE, FALARM, FFORK, FWAIT
      INTEGER BLUE, GREEN, ORANGE, RED, YELLOW
      GREEN = FFORK ()

      IF (GREEN .EQ. 0) THEN
         ORANGE = FALARM (1)
         ORANGE = FPTRACE (0, 0, 0, 0, 0)
         DO 10 BLUE = 1, 10
            DO 20 RED = 1, 100
               PRINT *, 'z'
      ELSE
         ORANGE = FWAIT (YELLOW)
         PRINT *, 'RETURN FROM WAIT   ', ORANGE
         ORANGE = FPTRACE (11, GREEN, 0, 2, 0)
         PRINT *, 'REGISTER TWO CONTAINS  ', ORANGE
      ENDIF
      END
```

2.60 READ, READX read from a file

Description

The **READ** system call reads a specified number of bytes from a file into a buffer.

The **READX** system call invokes the same function as **READ**, but it provides the alternative of communication with character device drivers that require more information or return more status information than **READ** can handle.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_read (fildes, buf, nbytes);                                    ¦
¦                                                                    ¦
¦   p_readx (fildes, buf, nbytes, ext);                              ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- Pascal external function definitions ---------------------------+
¦                                                                    ¦
¦   function p_read (fildes : integer; buf : readptr;                ¦
¦                    nbytes : integer) : integer; external;          ¦
¦                                                                    ¦
¦   function p_readx (fildes : integer; buf : readptr;               ¦
¦                   nbytes, ext : integer) : integer; external;      ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FREAD (FILDES, BUF, NBYTES)                                      ¦
¦                                                                    ¦
¦   FREADX (FILDES, BUF, NBYTES, EXT)                                ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

fildes
    is the descriptor returned by a successful **CREAT**, **DUP**, **DUP2**, **FCNTL**, **OPEN**, **PIPE**, **SOCKET**, or **SOCKETPAIR** system call.

        In Pascal, **fildes** is of type integer.

        In FORTRAN, **fildes** is of type INTEGER.

buf
    is a pointer to a buffer. The bytes read from the file specified by **fildes** are placed in this buffer when a **READ** or **READX** system call is executed.

        In Pascal, **buf** is of type readptr. (Readptr is a user-defined pointer to a packed array of type character.)

In FORTRAN, **buf** is a user-defined array of type CHARACTER.

nbytes
is the number of bytes to be read from the file specified by **fildes**.

In Pascal, **nbytes** is of type integer.

In FORTRAN, **nbytes** is of type integer.

ext
is a parameter of the **READX** call only. It provides a value or a pointer to a communication area for specific devices.

In Pascal, **ext** is of type integer.

In FORTRAN, **ext** is of type INTEGER.

In Pascal and FORTRAN, **ext** is device-dependent (see *AIX Technical Reference*).

Return Values

The return value is the actual number of bytes read from the file. If the return value is 0 (zero), the end file has been reached. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **READ** system routine, which reads a specified number of bytes from a file that has been opened for reading. In these examples, 100 bytes are read from the file /usr/include/ailtypes.inc into the buffer pointed to by the Pascal variable "yellow" and by the FORTRAN string "YELLOW".

Pascal

```
procedure read1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
  readary = packed array[1..100] of char;
  readptr = @readary;
var
  blue, orange, red : integer;
  yellow : readptr;

function p_read (fildes : integer; buf : readptr;
                nbytes : integer) : integer; external;

begin
  new (yellow);
  blue := p_open ('/usr/include/ailtypes.inc', RDONLY, 0);
  red := 100;
  orange := p_read (blue, yellow, red);
```

```
  writeln (orange);
end;
```

FORTRAN

```
      SUBROUTINE READ1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FREAD, FOPEN, BLUE, ORANGE, RED
      CHARACTER*100 YELLOW
      BLUE = FOPEN ('/usr/include/ailtypes.inc ', RDONLY, 0)
      RED = 100
      ORANGE = FREAD (BLUE, YELLOW, RED)
      PRINT *, ORANGE
      END
```

2.61 READLINK read the value of a symbolic link

Description

The **READLINK** system call places a specified number of characters from the symbolic-link path in a specified buffer.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_readlink (path, buf, bufsize);                                   ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FREADLINK (PATH, BUF, BUFSIZE)                                     ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

path
points to the path name of an existing file.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

buf
is the user's buffer to be filled with read data.

In Pascal, **buf** is a string variable or constant of type st80.

In FORTRAN, **buf** is a string variable or constant of type CHARACTER*80.

bufsize
is the size of **buf**

In Pascal, **bufsize** is of type integer.

In FORTRAN, **bufsize** is of type INTEGER.

Return Values

The count of characters read into the buffer is returned to the calling process. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **READLINK** system routine, after first creating a symbolic link between

/bushel/light/hide and /usr/include/aildefs.inc. The system call places the name of the symbolic link in the parameter **buf**. After successful completion of the call, the link is removed by **UNLINK**.

Pascal

```
procedure readlink1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
  path, buf : st80;

%include /usr/include/aildefs.inc

begin
  path := '/bushel/light/hide';
  green := p_symlink ('/usr/include/aildefs.inc', path);
  if (green = -1) then showerror;
  green := p_readlink (path, buf, 50);
  writeln ('Readlink returned: ', green : 2);
  if (green = -1) then showerror;
  writeln ('buf = ', buf);
  green := p_unlink ('/bushel/light/hide');
end;
```

FORTRAN

```
      SUBROUTINE READLINK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FREADLINK, FSYMLINK, FUNLINK, GREEN
      CHARACTER*80 BUF, P1, P2
      P1 = '/usr/include/aildefs.inc '
      P2 = '/bushel/light/hide '
      GREEN = FSYMLINK (P1, P2)
      IF (GREEN .EQ. -1) CALL ERRORS
      GREEN = FREADLINK (P2, BUF, 50)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      PRINT *, BUF
      GREEN = FUNLINK (P2)
      END
```

2.62 READV read input into multiple buffers

Description

The **READV** system call obtains data from a specified source and reads that data into a specified set of buffers.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_readv (d, iov, iovcnt);                                         ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+

+--- Pascal external function definition ------------------------------+
¦                                                                     ¦
¦   function p_readv (d : integer; var iov : iovarr;                  ¦
¦                     iovcnt : integer) : integer; external;          ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+

+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   This system call is not available in FORTRAN.                     ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

d
is a file descriptor or a socket descriptor.

In Pascal, **d** is of type integer.

iov
is an array of buffers.

In Pascal, **iov** is an array of records of type iovrec (user-defined).

iovcnt
is the number of buffers of the type specified by **iov**

In Pascal, **iovcnt** is of type integer.

Return Values

The number of bytes read and placed in a buffer is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

Examples

In the Pascal procedure that follows, five iovec records are initialized with base addresses and a buffer length of 10. Socket descriptor **s** is created by a **SOCKET** system call, and **READV** is called to read information from the socket into the five buffers pointed to by **iov**.

Pascal

```
procedure readv1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
  buf = packed array[1..10] of char;
  bufptr = ^buf;
  iovrec = record
     iov_len : integer;
     iov_base : bufptr;
   end;
  iovarr = array[1..5] of iovrec;
var
  i, s, green : integer;
  arr : st5;
  iov : iovarr;

%include /usr/include/aildefs.inc

function p_readv (d : integer; var iov : iovarr;
                                iovcnt : integer) : integer; external;

begin
  for i := 1 to 5 do
  begin
    iov[i].iov_len := 10;
    new(iov[i].iov_base);
  end;
  s := p_open ('/usr/include/aildefs.inc', RDONLY, 0);
  green := p_readv (s, iov, 5);
  if (green <> -1) then
    writeln ('Readv returned: OK')
  else
    writeln ('Readv returned: ERROR')
  if (green = -1) then showerror;
end;
```

2.63 REBOOT reinitialize or halt system operation

Description

The **REBOOT** system call makes a "request" that the operating system be reinitialized ("rebooted") or that it be stopped ("halted"). If the call fails, it returns a value; otherwise, it does not.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_reboot (howto);                                                ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FREBOOT (HOWTO)                                                  ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

howto
specifies one of the following flags:

RBNOSYNC prevents the normals **WRITE** of buffered data to file systems.

RBHALT stops system operation.

In Pascal, **howto** is of type integer.

In FORTRAN, **howto** is of type INTEGER.

Return Values

There is no return value from a successful **REBOOT** call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **REBOOT** system routine, with the **howto** flag set to RBHALT. The AIX subsystem is terminated and not restarted.

Pascal

```
procedure reboot1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  howto : integer;
```

```
%include /usr/include/aildefs.inc

begin
  howto := RBHALT;
  p_reboot (howto);
end;
```

FORTRAN

```
      SUBROUTINE REBOOT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FREBOOT, HOWTO
      HOWTO = RBHALT
      CALL FREBOOT (HOWTO)
      END
```

2.64 RECV, RECVMSG, RECVFROM receive a message from a socket

Description

The **RECV**, **RECVMSG**, and **RECVFROM** system calls receive a message from a specified socket.

Note: The **RECV** system call can be used only when the specified socket is in a connected state. The **RECVMSG** and **RECVFROM** calls can be used at any time.

Syntax

Pascal

```
p_recv (s, buf, len, flags);

p_recvmsg (s, msg, flags);

p_recvfrom (s, buf, len, flags, from, fromlen);
```

Pascal external function definitions

```
function p_recv (s : integer; var msg : msgarr; len : integer;
                              flags : integer) : integer; external;

function p_recvfrom (s : integer; var msg : msgarr; var len : integer
                     flags : integer; from : sockaddrptr;
                     fromlen : intptr) : integer; external;
```

FORTRAN

```
FRECV (S, BUF, LEN, FLAGS)

FRECVMSG (S, MSG, FLAGS)

FRECVFROM (S, BUF, LEN, FLAGS, FROM, FROMLEN)
```

Parameters

s
is the descriptor of a socket created by a **SOCKET** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

buf
is the structure in which the message is to be received.

Note: The **buf** parameter is used only in the **RECV** and **RECVFROM** system calls.

In Pascal, **buf** is an array of type msgarr (a user-defined array of type character).

In FORTRAN, **buf** is a user-defined array of type CHARACTER.

len
is the length of the message received. The **len** parameter is used only in the **RECV** and **RECVFROM** system calls.

In Pascal, **len** is of type integer.

In FORTRAN, **len** is of type INTEGER.

flags
is an argument whose value is specified by logically OR-ing one or both of the values shown here:

MSG_OOB processes the out-of-band data on sockets that support it.

MSG_PEEK peeks at the incoming message.

Note: In FORTRAN, the underscore is omitted (for example, "MSGOOB").

The **flags** parameter is used only in the **RECV** and **RECVFROM** system calls.

In Pascal, **flags** is of type integer.

In FORTRAN, **flags** is of type INTEGER.

msg
is a message header to be received.

In Pascal, **msg** is of type msghdrptr, declared in the include file ailtypes.inc.

In FORTRAN, **msg** is of type CHARACTER*80.

from
receives the source address of the message if the argument is a nonzero value.

In Pascal, **from** is of type sockaddrptr, declared in the include file ailtypes.inc.

In FORTRAN, **from** is of type CHARACTER*14 and corresponds to sockaddr.sa_data in Pascal.

fromlen
is initialized to the size of the **from** parameter. On return, this value is changed to the actual size of the address stored there.

In Pascal, **fromlen** is of type intptr.

In FORTRAN, **fromlen** is of type INTEGER.

Return Values

The length of the message, in bytes, is returned upon successful completion of the call. A value of -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **RECV** system routine. Because **RECV** receives a message only when a socket is in a connected state, sockets **s** and "s1" are created, after which "s1" is bound to the name "socket" and connected to socket **s**. Finally, a message is received from "s1".

Pascal

```
procedure recv1;

const
  %include /usr/include/ailpconsts.inc
type
  msgarr = packed array[1..50] of character;

  %include /usr/include/ailtypes.inc

var
  flags, len, namelen, s, s1, green : integer;
  buf : msgarr;
  name : sockaddrptr;

%include /usr/include/aildefs.inc

function p_recv (s : integer; var buf : msgarr; var len : integer;
                              flags : integer) : integer; external;


begin
  s := p_socket (PF_UNIX, SOCK_STREAM, 0);
  flags := MSG_DONTROUTE + MSG_OOB;
  len := 50;
  green := p_recv (s, buf, len, flags);
  writeln ('Recv returned: ', green : 2);
  if (green = -1) then showerror;
  end;
```

FORTRAN

```
      SUBROUTINE RECV1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FRECV, FSOCKET, FLAGS, GREEN
      CHARACTER*50 BUF
      FLAGS = MSGDONTROUTE +MSGOOB
      S = FSOCKET (PFUNIX, SKSTRM, 0)
      IF (S .EQ. -1) CALL ERRORS
      GREEN = FRECV (S, BUF, 50, FLAGS)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.65 RENAME rename a directory

Description
The **RENAME** system call renames a directory or file in a file system.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_rename (frompath, topath);                                        ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FRENAME (FROMPATH, TOPATH)                                          ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

frompath
is the name of the directory or file to be renamed.

In Pascal, **frompath** is a string variable or constant of type st80.

In FORTRAN, **frompath** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

topath
is the new name of the directory or file.

In Pascal, **topath** is a string variable or constant of type st80.

In FORTRAN, **topath** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **RENAME** system routine. The directory to be renamed by the call is /usr/games, which becomes /usr/work. The return value of the call is in the variable "folio".
Pascal

```
procedure rename1;

const
```

```
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  folio : integer;
  blue, red, : st80;

%include /usr/include/aildefs.inc

begin
  red := '/usr/games';
  blue := '/usr/work';
  folio := p_rename (red, blue);
  writeln (folio);
end;
```

FORTRAN

```
      SUBROUTINE RENAME1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER  FRENAME, FOLIO
      CHARACTER*80 BLUE, RED
      RED = '/usr/games '
      BLUE = '/usr/work '
      FOLIO = FRENAME (RED, BLUE)
      PRINT *, FOLIO
      END
```

2.66 RMDIR remove a directory

Description
The **RMDIR** system call removes a directory specified in the call.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_rmdir (path);                                                     ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FRMDIR (PATH)                                                       ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

path
is the name of the directory to be removed.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **RMDIR** system routine. The directory specified in the call is /usr/games, which is removed. The return value of the call is in the variable "folio".

Pascal

```
procedure rmdir1;

consts
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  folio : integer;
  red : st80;

%include /usr/include/aildefs.inc
```

```
begin
  red := '/usr/games';
  folio := p_rmdir (red);
  writeln (folio);
end;
```

FORTRAN

```
      SUBROUTINE RMDIR1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER  FRMDIR, FOLIO
      CHARACTER*80 RED
      RED = '/usr/games '
      FOLIO = FRMDIR (RED)
      PRINT *, FOLIO
      END
```

2.67 SELECT check the status of file descriptors and message queues

Description

The **SELECT** system call checks specified file descriptors and message queues for readiness to read or write or for any exceptional condition that may be pending.

Note: For more information about the **SELECT** system routine, and particularly about message queues, see the corresponding description in *AIX Operating System Technical Reference*.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_select (nfds, readfds, writefds, exceptfds, timeout);           ¦
¦                                                                     ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FSELECT (NFDS, READFDS, WRITEFDS, EXCEPTFDS, TIMEOUT)             ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

nfds
specifies the number of file descriptors being selected.

In Pascal, **nfds** is of type integer.

In FORTRAN, **nfds** is of type INTEGER.

readfds
points to a mask specifying a set of file descriptors or message queues to be checked for readiness to read (receive). Those that are ready are said to meet the selection criteria.

In Pascal, **readfds** is of type integer.

In FORTRAN, **readfds** is of type INTEGER.

writefds
points to a mask specifying a set of file descriptors or message queues to be checked for readiness to write (send). Those that are ready are said to meet the selection criteria.

In Pascal, **writefds** is of type integer.

In FORTRAN, **writefds** is of type INTEGER.

exceptfds
points to a mask specifying a set of file descriptors or message queues to be checked for exceptions. Those that have exceptions pending are said to meet the selection criteria.

In Pascal, **exceptfds** is of type integer.

In FORTRAN, **exceptfds** is of type INTEGER.

timeout
specifies the maximum length of time that the calling process will wait for at least one of the files or message queues specified in the masks to "test positive" for readiness or for a pending exception.

In Pascal, **timeout** is of type timevalptr.

In FORTRAN, **exceptfds** is an array of type INT with two elements. This array corresponds to the Pascal data structure--defined in the constants include file (Appendix B)--as follows:

TIMEOUT(1) = timeout.tv_sec

TIMEOUT(2) = timeout.tv_usec

Return Values

The value representing the total number of file descriptors and message queues that meet the selection criteria is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SELECT** system subroutine, which in these examples checks file descriptors 0, 1, and 2 for readiness to read (**rfds** points to bit mask 7). Upon return, the bit mask is overwritten with one showing which file descriptors have data ready.

Pascal

```
procedure select1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green, efds, rfds, wfds : integer;
  timeout : timevalptr;

begin
  new (timeout);
  timeout^.tv_sec := 5;
  rfds^ := 7;
  wfds^ := 0;
  efds^ := 0;
  green := p_select (3, rfds, wfds, efds, timeout)
  writeln (green);
end;
```

FORTRAN

```
SUBROUTINE SELECT1
INCLUDE (/usr/include/ailfconsts.inc)
INTEGER FSELECT, GREEN, TOUT(2)
TOUT(1) = 5
GREEN = FSELECT (3, 7, 0, 0, TOUT)
PRINT *, GREEN
END
```

2.68 SEMCTL invoke semaphore-control operations

Description

The **SEMCTL** system call invokes a variety of semaphore-control operations, most of which involve getting and setting the values of a data structure containing information about a set of semaphores.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

Pascal

```
p_semctl (semid, semnum, cmd, arg);
```

FORTRAN

```
FSEMCTL (SEMID, SEMNUM, CMD, ARG)
```

Parameters

semid
is the identifier of a semaphore set created by a previous **SEMGET** call (see page 2.69). The value of **semid** is returned by the **SEMGET** call.

In Pascal, **semid** is of type integer.

In FORTRAN, **semid** is of type INTEGER.

semnum
specifies the particular semaphore that will be affected by the control operation invoked by the call.

In Pascal, **semnum** is of type integer.

In FORTRAN, **semnum** is of type INTEGER.

cmd
specifies the control operation to be performed, which can be any of the options in the following list. These options are executed with respect to the semaphores specified by **semid** and **semnum**.

Note: Each constant is defined in the Pascal and FORTRAN constants include file (see Appendix B).

The fields referred to in the option descriptions below belong to the sem record (see Appendix C).

GETVAL returns the value of the semval field of the semaphore specified by **semid** and **semnum**.

SETVAL sets the value of the semval field of the semaphore set according to the array pointed to by the field arg.val.

GETPID returns the value of the sempid field of the semaphore specified by **semid** and **semnum**.

GTNCNT returns the value of the semncnt field of the semaphore specified by **semid** and **semnum**.

GTZCNT returns the value of the semzcnt field of the semaphore specified by **semid** and **semnum**.

The following **cmd** options return and set every semval field in the set of semaphores.

GETALL takes the values of the semval field of the semaphore specified by **semid** and **semnum** and stores them in the array pointed to by the field arg.arry.

SETALL sets semvals according to the array pointed to by arg.arry.

IPCSTT takes the current value of each field of the data structure associated with **semid** and stores it in the structure pointed to by the field arg.buf. In FORTRAN, information is stored in the first 14 elements of the field arg.arry (for further information see Table A on page 2.68).

IPCSET sets the value of the following fields of the data structure associated with **semid** to the corresponding values found in the structure pointed to by arg.buf.

sem_perm.uid
sem_perm.gid
sem_perm.mode (low-order nine bits only)

In FORTRAN these fields are set according to elements 1, 2, and 5 of the field arg.arry.

Note: This option can be used only when the effective user ID is equal to the super-user ID or to the user ID.

IPCRMD removes the semaphore identifier and its associated data structure from the operating system.

Note: This option can be used only when the effective user ID is equal to the super-user ID or to the user ID.

In Pascal, **cmd** is of type integer.

In FORTRAN, **cmd** is of type INTEGER.

arg

is a data structure determined by the **cmd** parameter. The values returned to the Pascal record and the FORTRAN array are listed on the next page.
For **cmd** options GETVAL, SETVAL, GETPID, GTNCNT, and GTZCNT:

Pascal	**FORTRAN**	**Description**
arg.val	ARG(1)	The values of the sem record are set and returned here.

For **cmd** options GETALL and SETALL:

Pascal	**FORTRAN**	**Description**
arg.arry@[1] . . . arg.arry@[1000]	ARG(1) . . . ARG(1000)	The values of the semary record are set and returned here.

For **cmd** options IPCSTT and IPCSET:

Pascal	**FORTRAN**	**Description**
arg.buf@sem_perm.uid	ARG(1)	owner's user ID
arg.buf@sem_perm.gid	ARG(2)	owner's group ID
arg.buf@sem_perm.cuid	ARG(3)	creator's user ID
arg.buf@sem_perm.cgid	ARG(4)	creator's group ID
arg.buf@sem_perm.mode	ARG(5)	access mode
arg.buf@sem_perm.seq	ARG(6)	lot-usage sequence number
arg.buf@sem_perm.key	ARG(7)	key value
arg.buf@sem_base@semval	ARG(8)	operation permission structure
arg.buf@sem_base@sempid	ARG(9)	ID of last process that issued **SEMOP**
arg.buf@sem_base@semncnt	ARG(10)	number of processes awaiting semval > cval
arg.buf@sem_base@semzcnt	ARG(11)	number of processes awaiting semval = 0
arg.buf@sem_nsems	ARG(12)	number of semaphores in a set
arg.buf@semlcnt	ARG(13)	processes waiting on

		locked semaphore
arg.buf@sem_otime	ARG(14)	time of last **SEMOP** call
arg.buf@sem_ctime	ARG(15)	last time this structure was changed by a **SEMCTL** call

In Pascal, **arg** is of type semrec.

In FORTRAN, **arg** is a 1000-element array of type INTEGER.

Return Values
The value returned from a successful call varies with the **cmd** option specified.

GTNCNT semncnt

GETPID sempid

GETVAL semval

GTZCNT semzcnt

All Others 0

The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SEMCTL** system routine. In these examples, a semaphore identifier is retrieved by a call to **SEMGET** from the associated key parameter ("red") returned by a call to the **ftok** system subroutine. The call to **SEMCTL** stores the current value of each member of the data structure associated with the **semid** parameter ("green") in the structure yellow.buf (in Pascal) or YELLOW(1)..YELLOW(15) in FORTRAN.

Pascal

```
procedure semctl1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, pink, purple, red : integer;
  orange : st80;
  brown : char;
  yellow : semrec;

%include /usr/include/aildefs.inc
```

```
begin
   new (yellow.buf);
  brown := 'm';
  orange := '/tmp/junk';
  blue := IPCCRT + IRUSR;
  red := p_ftok (orange, brown);
  green := p_semget (red, 20, blue);
  pink := 20;
  purple := p_semctl (green, pink, 2, yellow);
  writeln (purple);
end;
```

FORTRAN

```
      SUBROUTINE SEMCTL1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSEMCTL, FFTOK, FSEMGET, BLUE, GREEN, PINK
      INTEGER PURPLE, RED, YELLOW(1000)
      CHARACTER BROWN, ORANGE(80)
      BROWN = 'm'
      ORANGE = '/tmp/junk '
      BLUE = IPCCRT + IRUSR
      RED = FFTOK (ORANGE, BROWN)
      GREEN = FSEMGET (RED, 20, BLUE)
      PINK = 20
      PURPLE = FSEMCTL (GREEN, PINK, 2, YELLOW)
      PRINT *, PURPLE
      END
```

2.69 SEMGET get or create a semaphore-set ID

Description

The **SEMGET** system call returns a semaphore-set ID associated with the specified **key** parameter.

Syntax

Pascal

```
p_semget (key, nsems, semflg);
```

FORTRAN

```
FSEMGET (KEY, NSEMS, SEMFLG)
```

Parameters

key
is a semaphore-set ID that has been assigned directly by the programmer or has been returned by the **ftok** system subroutine or similar algorithm.

In Pascal, **key** is of type integer.

In FORTRAN, **key** is of type INTEGER.

nsems
specifies the number of semaphores in a set.

In Pascal, **nsems** is of type integer.

In FORTRAN, **nsems** is of type INTEGER.

semflg
specifies one or more conditions (options) governing the creation of a semaphore-set data structure and the accessibility of the semaphore set. The parameter value is that of one of the following options or is constructed from two or more of those options by logical ORing. The options are defined as constants in the Pascal and FORTRAN constants include files.

IPCCRT creates a data structure if one does not exist.

IPCEXL causes **SEMGET** to fail if IPCCRT is also set and the data structure already exists.

IRUSR permits the process that owns the data structure to read it.

IWUSR permits the process that owns the data structure to modify it.

IRGRP permits the group associated with the data structure to read it.

IWGRP permits the group associated with the data structure to modify it.

IROTH permits others to read the data structure.

IWOTH permits others to modify the data structure.

In Pascal, **semflg** is of type integer.

In FORTRAN, **semflg** is of type INTEGER.

Return Values

A semaphore-set ID is returned upon successful completion of the call. The value -1 is returned and error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **SEMGET** system routine, which in these examples returns a semaphore identifier associated with the key parameter ("red") returned by a call to the **ftok** system subroutine. This identifier is the value printed out.

Pascal

```
procedure semget1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red : integer;
  orange : st80;
  brown : char;

%include /usr/include/aildefs.inc

begin
  brown := 'm';
  orange := '/tmp/junk';
  blue := IPCCRT + IRUSR;
  red := p_ftok (orange, brown);
  green := p_semget (red, 20, blue);
  writeln (green);
end;
```

FORTRAN

```
      SUBROUTINE SEMGET1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSEMGET, FFTOK, BLUE, GREEN, RED
      CHARACTER BROWN, ORANGE(80)
      BROWN = 'm'
```

```
ORANGE = '/tmp/junk '
BLUE = IPCCRT + IRUSR
RED = FFTOK (ORANGE, BROWN)
GREEN = FSEMGET (RED, 20, BLUE)
PRINT *, GREEN
END
```

2.70 SEMOP perform semaphore operations

Description

The **SEMOP** system call invokes a group of semaphore operations that are performed on a specified semaphore set.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_semop (semid, sops, nsops);                                    ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FSEMOP (SEMID, SOPS, NSOPS)                                      ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

semid
is the ID of the semaphore set that is to be operated on.

In Pascal, **semid** is of type integer.

In FORTRAN, **semid** is of type INTEGER.

sops
is a pointer to an array of semaphore operation data structures. The breakdown of this parameter for each of the **n** semaphores is as follows:

Pascal	**FORTRAN**	**Description**
nsops[n].sem_num	NSOPS(n,1)	Semaphore number
nsops[n].sem_op	NSOPS(n,2)	Semaphore operation
nsops[n].sem_flg	NSOPS(n,3)	Operation flags

Each semaphore operation specified by sem_op (FORTRAN, NSOPS(n,2)) is performed on the corresponding semaphore specified by sem_num (FORTRAN, NSOPS(n,1)). The sem_flg (FORTRAN, NSOPS(n,3)) value can be 0, one of the following constants, or the value obtained from logically ORing (adding) the following constants defined in the Pascal and FORTRAN constants include files.

SEMNDO (SEM_UNDO)
SEMODR (SEM_ORDER)
IPCNWT (IPC_NOWAIT)

Note: For further information about these constants and the semaphore operations, see *AIX Operating System Technical Reference*.

In Pascal, **sops** is of type semopary.

In FORTRAN, **sops** is an array(1000,3) of type INTEGER.

nsops

specifies the number of semaphore operations to be performed. A semaphore set is limited to 1000 semaphores.

In Pascal, **nsops** is of type integer.

In FORTRAN, **nsops** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. In addition, each value of **sempid** for each semaphore in the array pointed to by **sops** is set to the process ID of the calling process. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **SEMOP** system routine. In these examples, a semaphore identifier is retrieved by a call to **SEMGET** from the associated **key** parameter ("red") returned by a call to the **ftok** system subroutine. The call to **SEMGET** would typically be part of a program used between two processes using semaphores to buffer information. The call to **SEMOP** is used by the sending process to perform two semaphore operations. The first operation decrements a counter of empty buffer available upon sending information. The second operation increments a second counter of data packages that can be received by a second process.

Pascal

```
procedure semop1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, grey, pink, red : integer;
  orange : st80;
  brown : char;
  yellow : semopary;

%include /usr/include/aildefs.inc

begin
  brown := 'z';
  orange := '/tmp/junk';
  grey := IPCCRT + IRUSR + IWUSR;
  red := p_ftok (orange, brown);
  pink := p_semget (red, 2, grey);
  yellow[1].sem_num := 1;
  yellow[2].sem_num := 2;
  yellow[1].sem_op := 1;
  yellow[2].sem_op := -1;
  yellow[1].sem_flg := 0;
  yellow[2].sem_flg := 0;
  blue := p_semop (pink, yellow, 2);
  writeln (blue);
```

```
end;
```

FORTRAN

```
      SUBROUTINE SEMOP1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSEMOP, FSEMGET, FFTOK, BLUE, GREY
      INTEGER PINK, RED, YELLOW(1000,3)
      CHARACTER BROWN, ORANGE(80)
      BROWN = 'z'
      ORANGE = '/tmp/junk '
      GREY = IPCCRT + IRUSR + IWUSR
      RED = FFTOK (ORANGE, BROWN)
      PINK = FSEMGET (RED, 2, GREY)
      YELLOW(1,1) = 1
      YELLOW(2,1) = 2
      YELLOW(1,2) = 1
      YELLOW(2,2) = -1
      YELLOW(1,3) = 0
      YELLOW(2,3) = 0
      BLUE = SEMOP (PINK, YELLOW, 2)
      PRINT *, BLUE
      END
```

2.71 SEND, SENDMSG, SENDTO send a message from a socket

Description

The **SEND**, **SENDMSG**, and **SENDTO** system calls send a message from a specified socket.

Note: The **SEND** system call can be used only when the specified socket is in a connected state. The **SENDMSG** and **SENDTO** calls can be used at any time.

Syntax

Pascal

```
p_send (s, msg, len, flags);

p_sendmsg (s, msg, flags);

p_sendto (s, msg, len, flags, to, tolen);
```

Pascal external function declarations

```
function p_send (s : integer; msg : msgarr; len : integer;
                              flags : integer) : integer; external;

function p_sendto (s : integer; msg : msgarr; len : integer;
                     flags : integer; to : sockaddrptr;
                     tolen : integer) : integer; external;
```

FORTRAN

```
FSEND (S, MSG, LEN, FLAGS)

FSENDMSG (S, MSG, FLAGS)

FSENDTO (S, MSG, LEN, FLAGS, TO, TOLEN)
```

Parameters

s
is the descriptor of a socket created by a **SOCKET** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

len
is the length of the message to be sent. The **len** parameter is used only in the **SEND** and **SENDTO** system calls.

In Pascal, **len** is of type integer.

In FORTRAN, **len** is of type INTEGER.

flags
is an argument whose value is specified by logically OR-ing one or both of the values shown here:

MSG_OOB processes the out-of-band data on sockets that support this notion.

MSG_DONTROUTE sends the message without reference to routing tables.

The **flags** parameter is used only in the **SEND** and **SENDTO** system calls.

In Pascal, **flags** is of type integer.

In FORTRAN, **flags** is of type INTEGER.

msg
is a message header to be received.

When the **SENDMSG** system call is used:

In Pascal, **msg** is of type msghdrptr, declared in the include file ailtypes.inc.

In FORTRAN, **msg** is of type CHARACTER*80.

When the **SEND** and **SENDTO** system calls are used:

In Pascal, **msg** is an array of type msgarr (a user-defined packed array of type character).

In FORTRAN, **msg** is a user-defined array of type CHARACTER.

to
is the address of the target.

In Pascal, **to** is of type sockaddrptr, declared in the include file ailtypes.inc.

In FORTRAN, **to** is of type CHARACTER*14. The final character of the string must be a blank space.

tolen
is the size of the data in the **to** parameter.

In Pascal, **tolen** is of type integer.

In FORTRAN, **tolen** is of type INTEGER.

Return Values
The length of the message, in bytes, is returned upon successful completion of the call. A value of -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **SEND** system routine. Because **SEND** sends a message only when a socket is in a connected state, sockets "s" and "s1" are created, after which "s1" is bound to the name "socket" and connected to socket "s". Finally, a message is sent from "s" to the connected socket (in this case, "s1").

Pascal

```
procedure send1;

const
  %include /usr/include/ailpconsts.inc
type
  msgarr = packed array[1..50] of char;

  %include /usr/include/ailtypes.inc

var
  flags, len, s : integer;
  msg : msgarr;

function p_send (s : integer; msg : msgarr;
                              len, flags : integer) : integer; external;

%include /usr/include/aildefs.inc

begin
  s := p_socket (PF_UNIX, SOCK_STREAM, 0);
  msg := 'This is a short message';
  len := 23;
  flags := MSG_DONTROUTE + MSG_OOB;
  green := p_send (s, msg, len, flags);
  writeln ('Send returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE SEND1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSEND, FSOCKET, FLAGS, LEN, S
      CHARACTER*50 MSG
      FLAGS = MSGDONTROUTE + MSGOOB
      S = FSOCKET (PFUNIX, SKSTRM, 0)
      IF (S .EQ. -1) CALL ERRORS
      MSG = 'This is a short message '
      LEN = 23
      GREEN = FSEND (S, MSG, LEN, FLAGS)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.72 SETGROUPS set a group access list

Description

The **SETGROUPS** system call sets, or creates, the group access list of the current user process according to the values set in an array specified in the call.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_setgroups (ngrps, gidset);                                       ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+

+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FSETGROUPS (NGRPS, GIDSET)                                         ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

ngrps
is the number of entries in the array pointed to by **gidset**. This number may not exceed the constant NGROUP defined in the Pascal and FORTRAN constants include files.

In Pascal, **ngrps** is of type integer.

In FORTRAN, **ngrps** is of type INTEGER.

gidset
is an array containing the values to be placed in the group access list. The maximum number of elements the array may hold is equal to the constant NGROUP defined in the Pascal and FORTRAN constants include files.

In Pascal, **gidset** is an array of type intngroup. (Setptr is a pointer to a user-defined integer array.)

In FORTRAN, **gidset** is a user-defined array, of type INTEGER, containing up to NGROUP elements.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **SETGROUPS** system routine, which in these examples sets the group access list of the current user process to that of the three named elements of the array pointed to (Pascal) or specified (FORTRAN) by the variable "red".

Pascal

```
procedure setgroups1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green : integer;
  red : intngroup;

begin
  red[1] := 1;
  red[2] := 2;
  red[3] := 3;
  green := 3;
  blue := p_setgroups (green, red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE SETGROUPS1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSETGROUPS, BLUE, GREEN, RED(3)
      RED(1) = 1
      RED(2) = 2
      RED(3) = 3
      GREEN = 3
      BLUE = FSETGROUPS (GREEN, RED)
      PRINT *, BLUE
      END
```

2.73 SETHOSTID set an identifier for the host machine

Description

The **SETHOSTID** system call sets a unique identifier for the current host.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal -------------------------------------------------------------+
¦                                                                         ¦
¦   p_sethostid (hostid);                                                 ¦
¦                                                                         ¦
¦                                                                         ¦
+-------------------------------------------------------------------------+
```

```
+--- FORTRAN ------------------------------------------------------------+
¦                                                                         ¦
¦   FSETHOSTID (HOSTID)                                                   ¦
¦                                                                         ¦
¦                                                                         ¦
+-------------------------------------------------------------------------+
```

Parameters

hostid
is the unique identifier assigned to the current host.

In Pascal, **hostid** is of type integer.

In FORTRAN, **hostid** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page set the host ID to 25.

Pascal

```
procedure sethostid1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;

%include /usr/include/aildefs.inc

begin
```

```
   green := p_sethostid(25);
    writeln ('Sethostid returned: ', green : 2);
   if (green = -1) then showerror;
  end;
```

FORTRAN

```
        SUBROUTINE SETHOSTID1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FSETHOSTID, GREEN
        GREEN = FSETHOSTID(25)
        PRINT *, GREEN
        IF (GREEN .EQ. -1) CALL ERRORS
        END
```

2.74 SETHOSTNAME set the name of the current host

Description

The **SETHOSTNAME** system call sets the name of the current host machine.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ---------------------------------------------------------+
|                                                                    |
|   p_sethostname (name, namelen);                                   |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
|                                                                    |
|   FSETHOSTNAME (NAME, NAMELEN)                                     |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
```

Parameters

name
is the name of the host machine.

In Pascal, **name** is of type st80.

In FORTRAN, **name** is of type CHARACTER*80. The terminating character of the string must be a blank space.

namelen
is the length of the name parameter.

In Pascal, **namelen** is of type integer.

In FORTRAN, **namelen** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page set the name of the current host to "HNAME".

Pascal

```
procedure sethostname1;

const
  %include /usr/include/ailpconsts.inc
```

```
type
  %include /usr/include/ailtypes.inc
var
  green, namelen : integer;
  name : st80;

%include /usr/include/aildefs.inc

begin
  namelen := 5;
  name := 'HNAME ';
  green := p_sethostname (name, namelen);
  writeln ('Sethostname returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE SETHOSTNAME1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSETHOSTNAME, GREEN
      GREEN = FSETHOSTNAME ('HNAME ', 5)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.75 SETITIMER set the value of an internal timer

Description

The **SETITIMER** system call sets the value of internal timer specified in the call.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ---------------------------------------------------------+
|                                                                    |
|   p_setitimer (which, value, ovalue);                              |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
|                                                                    |
|   FSETITIMER (WHICH, VALUE, OVALUE)                                |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
```

Parameters

which
specifies one of the following internal timers:

ITIMER_REAL the timer decrements in real time.

ITIMER_VIRTUAL the timer decrements in process virtual time (it runs only when the process is executing).

ITIMER_PROF the timer decrements both in process virtual time and when the operating system is executing on behalf of the process.

Note: In FORTRAN, the underscore is omitted (for example, "ITIMERREAL").

In Pascal, **which** is of type integer.

In FORTRAN, **which** is of type INTEGER.

value
is a variable in which the time is returned when the call is executed.

In Pascal, **value** is of type itimerval, declared in the include file ailtypes.inc.

In FORTRAN, **value** is an array of four integers, or INTEGER VALUE(4).

ovalue
is a variable in which the previous timer value is returned when the call is executed.

In Pascal, **ovalue** is of type itimerval, declared in the include file ailtypes.inc.

In FORTRAN, **ovalue** is an array of four integers, or INTEGER VALUE(4).

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code is set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SETITIMER** system routine, which in these examples set the value of the ITIMER_REAL timer to "5" and returns the previous value in the variable "ovalue".

Pascal

```
procedure setitimer1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  which : integer;
  ovalue, vvalue : itimerval;

%include /usr/include/aildefs.inc

begin
  with vvalue do
    begin
      it_interval.tv_sec := 5;
      it_interval.tv_usec := 4;
      it_value.tv_sec := 3;
      it_value.tv_usec := 2;
    end;
  which := ITIMER_REAL;
  green := p_setitimer (which, vvalue, ovalue);
  writeln ('Setitimer returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE SETITIMER1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSETITIMER, VAL(4), OVAL(4), GREEN
      VAL(1) = 5
      VAL(2) = 4
      VAL(3) = 3
      VAL(4) = 2
      GREEN = FSETITIMER (ITIMERREAL, VAL, OVAL)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
```

```
END
```

2.76 SETLOCAL set the alias for <LOCAL>

Description
The **SETLOCAL** system call sets the calling process' alias for <LOCAL>.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                        ¦
¦   p_setlocal (localname)                                               ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                        ¦
¦   FSETLOCAL (LOCALNAME)                                                ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

Parameters

localname
is the pathname for <LOCAL>.

In Pascal, **localname** is of type st80.

In FORTRAN, **localname** is of type CHARACTER*80. The terminating character must be a blank space.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **SETLOCAL** system routine, which in these examples sets the value of the current <LOCAL> to "NEW_AIX" (Pascal) or "NEWAIX" (FORTRAN).

Pascal

```
procedure setlocal1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
  buf : st80;

  %include /usr/include/aildefs.inc

begin
```

```
    buf := 'new_aix';
    green := p_setlocal (buf);
    writeln (buf);
    writeln ('Setlocal returned: ', green : 2);
    if (green = -1) then showerror;
  end;
```

FORTRAN

```
      SUBROUTINE SETLOCAL1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSETLOCAL, GREEN
      CHARACTER BUF(80)
      BUF = 'NEWAIX '
      GREEN = FSETLOCAL (BUF)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.77 SETPGRP, SETPGID set a process group ID

Description

The **SETPGRP** and **SETPGID** system calls set a process group ID.

The **SETPGRP** system call sets the group ID of the calling process to its process ID.

The **SETPGID** system call is used either to join a calling process to a process group or to create a new process group.

Syntax

Pascal

```
p_setpgrp;

p_setpgid (pid, pgid)
```

FORTRAN

```
FSETPGRP ()

FSETPGID (PID, PGID)
```

Parameters

The **SETPGRP** system call has no parameters.

pid
is the process group ID to be set.

In Pascal, **pid** is of type integer.

In FORTRAN, **pid** is of type integer.

pgid
specifies the value to which the **pid** is to be set.

In Pascal, **pgid** is of type integer.

In FORTRAN, **pgid** is of type integer.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **SETPGRP** system routine, which returns a new process group ID in the variable "blue".

Pascal

```
procedure setpgrp1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue : integer;

%include /usr/include/aildefs.inc

begin
  blue := p_setpgrp;
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE SETPGRP1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSETPGRP, BLUE
      BLUE = FSETPGRP ()
      PRINT *, BLUE
      END
```

2.78 SETSOCKOPT set options on sockets

Description

The **SETSOCKOPT** system sets the options for a specified socket. These options may exist at multiple protocol levels, and are always present at the uppermost socket level.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

Pascal

```
p_setsockopt (s, level, optname, optval, optlen)
```

FORTRAN

```
FSETSOCKOPT (S, LEVEL, OPTNAME, OPTVAL, OPTLEN)
```

Parameters

s
is the descriptor of a socket that was created with a **SOCKET** system call.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

level
is level at which the desired option resides. To manipulate options at the socket level, specify the level as SOL_SOCKET.

In Pascal, **level** is of type integer.

In FORTRAN, **level** is of type INTEGER;

optname
is the option name, passed uninterpreted to the appropriate protocol module for interpretation. The socket-level options are:

SO_DEBUG turns on recording of debugging information.

SO_REUSEADDR allows local address reuse.

SO_KEEPALIVE keeps connections alive.

SO_DONTROUTE does not apply routing on outgoing messages.

SO_LINGER lingers on a **CLOSE** system call if data is present.

SO_OOBINLINE leaves received out-of-band data in line.

SO_SNDBUF sends buffer size.

SO_RCVBUF receives buffer size.

SO_ERROR gets error status.

SO_TYPE gets socket type.

SO_BROADCAST requests permission to transmit broadcast messages.

Note: In FORTRAN, the underscore is omitted (for example, "SODEBUG").

In Pascal, **optname** is of type integer.

In FORTRAN, **optname** is of type INTEGER.

optval
points to a buffer, in which the option values are returned by the system call.

In Pascal, **optval** is of type st80.

In FORTRAN, **optval** is of type CHARACTER*80. The terminating character must be a blank space.

optlen.
specifies the length of the buffer pointed to by **optval**.

In Pascal, **optlen** is of type integer.

In FORTRAN, **optlen** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SETSOCKOPT** system routine, which in these examples sets the options for socket **s** The level has been set to SOL_SOCKET and the option name to SO_DEBUG.

Pascal

```
procedure setsockopt1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  level, optlen, optname, s, green : integer;
  optval : st80;

%include /usr/include/aildefs.inc
```

```
begin
  s := p_socket (PF_UNIX, SOCK_STREAM, 0);
  level := SOL_SOCKET;
  optname := SO_DEBUG;
  optval := '';
  optlen := 0;
  green := p_setsockopt (s, level, optname, otpval, optlen);
  writeln ('Setsockopt returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE SETSOCKOPT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSETSOCKOPT, FSOCKET, LEVEL, OPTLEN, OPTNAME, S, GREEN
      CHAR*80 OPTVAL
      S = FSOCKET (PFUNIX, SKSTRM, 0)
      OPTNAME = SODEBUG
      IF (S .EQ. -1) CALL ERRORS
      LEVEL = SOLSOCKET
      GREEN = FSETSOCKOPT (S, LEVEL, OPTNAME, 0, 0)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.79 SETTIMEOFDAY set the current time

Description

The **SETTIMEOFDAY** system call sets the current time.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                        ¦
¦   p_settimeofday (tp, tzp);                                            ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                        ¦
¦   FSETTIMEOFDAY (TP, TZP)                                              ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

Parameters

tp
holds two integers:

1. the number of seconds that have elapsed since 00:00:00 January 1, 1970 GMT, plus

2. the number of microseconds that must be added to the preceding number to get the current time.

   In Pascal, **tp** is of type timeval, declared in the include file ailtypes.inc.

   In FORTRAN, **tp** is of type INTEGER TP(2).

tzp
holds two integers:

1. the time west of Greenwich in minutes.

2. the type of DST correction to apply.

   In Pascal, **tzp** is of type timezone, declared in the include file ailtypes.inc.

   In FORTRAN, **tzp** is of type INTEGER TZP(2).

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **SETTIMEOFDAY** system routine, which in these examples sets the current Greenwich time and the current time to the values that **tp** and **tzp** are given when they are initialized.

Pascal

```
procedure settimeofday1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  tp : timeval;
  tzp : timezone;

%include /usr/include/aildefs.inc

begin
  tp.tv_sec := 34567;
  tp.tv_usec := 12345;
  tzp.tz_minuteswest := 93845;
  green := p_settimeofday (tp, tzp);
  writeln ('Settimeofday returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE SETTIMEOFDAY1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER, FSETTIMEOFDAY, TP(2), TZP(2), GREEN
      TP(1) = 123445
      TP(2) = 567889
      TZP(1) = 48604
      GREEN = FSETTIMEOFDAY (TP, TZP)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.80 SETUID, SETGID set user or group identifiers

Description
The **SET** system calls described in this section set the user or group IDs to values specified in the call. Both the effective and the real IDs are set.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_setuid (uid);                                                     ¦
¦                                                                       ¦
¦   p_setgid (gid);                                                     ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FSETUID (UID)                                                       ¦
¦                                                                       ¦
¦   FSETGID (GID)                                                       ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

uid
is used with **SETUID**. It is the new value of the user ID to be set.

In Pascal, **uid** is of type integer.

In FORTRAN, **uid** is of type INTEGER.

gid
is used with **SETGID**. It is the new value of the new group ID to be set.

In Pascal, **gid** is of type integer.

In FORTRAN, **gid** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of a call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SETGID** system routine, which sets the real and effective group IDs. In these examples a value is obtained through a call to **GETGID** and then sent to **SETGID**.

Pascal

```
procedure setgid1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue : integer;
  red : ushrt;

%include /usr/include/aildefs.inc

begin
  red := p_getgid;
  blue := p_setgid (red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE SETGID1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER*2 FGETGID, FSETGID, BLUE, RED
      RED = FGETGID ()
      BLUE = FSETGID (RED)
      PRINT *, BLUE
      END
```

2.81 SETXVERS set the UNIX version string

Description
The **SETXVERS** system call sets the value of the UNIX version string.

Syntax

```
+--- Pascal -------------------------------------------------------+
¦                                                                  ¦
¦   p_setxvers (xvers);                                            ¦
¦                                                                  ¦
¦                                                                  ¦
+------------------------------------------------------------------+
```

```
+--- FORTRAN ------------------------------------------------------+
¦                                                                  ¦
¦   FSETXVERS (XVERS)                                              ¦
¦                                                                  ¦
¦                                                                  ¦
+------------------------------------------------------------------+
```

Parameters

xvers
is a pointer to the version string.

In Pascal, **xvers** is of type st80.

In FORTRAN, **xvers** is of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SETXVERS** system routine, which in these examples sets the value of the version string to "NEW_VERSION".

Pascal

```
procedure setxvers1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green: integer:
  s, s1 : st80;

%include /usr/include/aildefs.inc

begin
```

```
    s := 'NEW_VERSION';
    green := p_setxvers (s);
    green := p_getxvers (s1, 10);
    writeln (s);
  end;
```

FORTRAN

```
        SUBROUTINE SETXVERS1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER, FSETXVERS, FGETXVERS, GREEN
        CHARACTER*80 S, S1
        S = 'NEW_VERSION '
        GREEN = FSETXVERS (S)
        GREEN = FGETXVERS (S1, 10)
        PRINT *, S1
        END
```

2.82 SHMAT attach a shared-memory segment or mapped file

Description

The **SHMAT** system call attaches one of the following to the address space of the calling process:

a shared memory segment, o
a mapped file associated with a shared-memory identifier (returned b **SHMGET**), or
a file descriptor (returned by **OPEN**).

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_shmat (shmid, shmadr, shmflg);                                    ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FSHMAT (SHMID, SHMADR, SHMFLG)                                      ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

shmid
is either a shared-memory identifier returned by **SHMGET** or a file descriptor returned by **OPEN**.

In Pascal, **shmid** is of type integer.

In FORTRAN, **shmid** is of type INTEGER.

shmadr
determines the address to which the shared-memory segment is attached.

In Pascal, **shmadr** is of type integer.

In FORTRAN, **shmadr** is of type INTEGER.

shmflg
specifies a set of conditions governing the attachment of a shared-memory segment or a mapped file to an address space. The value assigned to **shmflg** is that of one or more of the options in the following list. These are defined in the Pascal and FORTRAN constants include files.

SHMRND rounds the address given by the **shmadr** parameter to the next lower segment boundary if necessary.

SHMRDO specifies read-only mode (the default is read-write mode).

In Pascal, **shmflag** is of type integer.

In FORTRAN, **shmflg** is of type INTEGER.

Return Values

The start address of the attached shared-memory segment or mapped file is returned on successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **SHMAT** system routine. In these examples, the shared-memory identifier returned by a **SHMGET** call is used to specify the shared-memory segment that **SHMAT** attaches to the address of the calling process.

Pascal

```
procedure shmat1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red : integer;
  orange : st80;
  brown : char;

%include /usr/include/aildefs.inc

begin
  brown := 'm';
  orange := '/tmp/junk';
  blue := IPCCRT + IRUSR;
  red := p_ftok (orange, brown);
  green := p_shmget (red, 512, blue);
  blue := p_shmat (green, 0, 0);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE SHMAT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSHMAT, FSHMGET, FFTOK, BLUE, GREEN, RED
      CHARACTER BROWN, ORANGE(80)
      BROWN = 'm'
      ORANGE = '/tmp/junk '
      BLUE = IPCCRT + IRUSR
      RED = FFTOK (ORANGE, BROWN)
      GREEN = FSHMGET (RED, 512, BLUE)
      BLUE = FSHMAT (GREEN, 0, 0)
      PRINT *, BLUE
      END
```

2.83 SHMCTL invoke shared-memory-control operations

Description

The **SHMCTL** system call invokes three shared-memory-control operations.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_shmctl (shmid, cmd, buf);                                       ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FSHMCTL (SHMID, CMD, BUF)                                         ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

shmid
is a shared-memory-segment identifier returned by the **SHMGET** call.

In Pascal, **shmid** is of type integer.

In FORTRAN, **shmid** is of type INTEGER

cmd
specifies the control operation to be performed. These operations are defined in the Pascal and FORTRAN constants include files.

IPCRMD removes the shared-memory identifier specified by **shmid** from the system and erases the shared-memory segment and associated data structure.

Note: This option can be executed only by a process that has an effective user ID equal to that of the super-user or to the value of shm.perm.uid in the data structure.

IPCSET sets the value of the following members of the data structure associated with **shmid** to the corresponding value found in the structure pointed to by the **buf** parameter:

shperm.uid
shper.gid
shperm.mode (low-order nine bits only)

Note: This **cmd** option can be executed only by a process that has an effective user ID equal to that of super-user or to the value of shm.perm.uid in the data structure associated with the **shmid** parameter.

IPCSTT places the current value of each member of the data structure associated with **shmid** in the structure pointed to by the **buf** parameter. The current process must have read permissions on this shared-memory segment or mapped file.

In Pascal, **cmd** is of type integer.

In FORTRAN, **cmd** is of type INTEGER.

buf
is a pointer to the data structure to be modified.

In Pascal, **buf** is of type smds.

In FORTRAN, **buf** is an array(12) of type INTEGER.

This array corresponds to the Pascal data structure--defined in the aildefs.inc file (Appendix C)--as follows:

BUF(1) = shperm.uid, shperm.gid (2 bytes each)

BUF(2) = shperm.cuid, shperm.cgid (2 bytes each)

BUF(3) = shperm.mode, shperm.seg (2 bytes each)

BUF(4) = shperm.key

BUF(5) = shperm.shseqsz

BUF(6) = shperm.spare0

BUF(7) = shperm.shlpid

BUF(8) = shcpid

BUF(9) = shnattach, shcnattach (2 bytes each)

BUF(10) = shatime

BUF(11) = shdtime

BUF(12) = shctime

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **SHMCTL** system routine. In these examples, the **cmd** parameter ("pink") specifies an option that will place information about a shared-memory segment (identified by the **shmid** parameter, or "green") in the data structure pointed to by the **buf** parameter. In Pascal this structure is the record pointed to by the variable "yellow". In FORTRAN, "YELLOW" is an array. The value printed is the process user ID.

Pascal

```
procedure shmctl1;

const
  %include/usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, pink, red : integer;
  orange : st80;
  brown : char;
  yellow : smds;

%include /usr/include/aildefs.inc

begin
  brown := 'm';
  orange := '/tmp/junk';
  blue := IPCCRT + IRUSR;
  red := p_ftok (orange, brown);
  green := p_shmget (red, 512, blue);
  pink := IPCSTT;
  red := p_shmctl (green, pink, yellow);
  writeln (red);
end;
```

FORTRAN

```
      SUBROUTINE SHMCTL1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSHMCTL, FSHMGET, FFTOK, BLUE, GREEN, PINK, RED, YELLOW(12)
      CHARACTER BROWN, ORANGE(80)
      BROWN = 'm'
      ORANGE = '/tmp/junk '
      BLUE = IPCCRT + IRUSR
      RED = FFTOK (ORANGE, BROWN)
      GREEN = FSHMGET (RED, 512, blue)
      PINK = IPCSTT
      RED = FSHMCTL (GREEN, PINK, YELLOW)
      PRINT *, RED
      END
```

2.84 SHMDT detach a shared-memory or mapped file segment

Description

The **SHMDT** system call detaches a shared-memory segment from the data segment of the calling process. Shared memory segments must be explicitly detached using **SHMDT**.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_shmdt (shmadr);                                                 ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FSHMDT (SHMADR)                                                   ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

shmadr
is the address at which the memory segment is detached from the address space of the calling process. It is the same address as that at which the segment was originally attached (see **SHMAT** on page 2.82)

In Pascal, **shmadr** is of type integer.

In FORTRAN, **shmadr** is of type INTEGER.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **SHMDT** system routine, which in these examples detaches the shared-memory segment identified by the address returned by a call to **SHMAT**.

Pascal

```
procedure shmdt1;

const
  %include/usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, grey, pink, red : integer;
  orange : st80;
  brown : char;
```

```
%include /usr/include/aildefs.inc

begin
  brown := 'm';
  orange := '/tmp/junk';
  grey := IPCCRT + IRUSR;
  red := p_ftok (orange, brown);
  pink := p_shmget (red, 512, grey);
  blue := p_shmat (pink, 0, 0);
  green := p_shmdt (blue);
  writeln (green);
end;
```

FORTRAN

```
      SUBROUTINE SHMDT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSHMDT, FSHMAT, FSHMGET, FFTOK, BLUE, GREEN
      INTEGER GREY, PINK, RED
      CHARACTER BROWN, ORANGE(80)
      BROWN = 'm'
      ORANGE = '/tmp/junk '
      GREY = IPCCRT + IRUSR
      RED = FFTOK (ORANGE, BROWN)
      PINK = FSHMGET (RED, 512, GREY)
      BLUE = FSHMAT (PINK, 0, 0)
      GREEN = FSHMDT (BLUE)
      PRINT *, GREEN
      END
```

2.85 SHMGET get a shared-memory-segment identifier

Description

The **SHMGET** system call returns a shared-memory-segment ID associated with the specified **key** value.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_shmget (key, size, shmflg);                                       ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FSHMGET (KEY, SIZE, SHMFLG)                                         ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

key
is either the value 0 (IPCPVT) or an IPC key returned by the **ftok** system subroutine. A shared-memory ID, its associated data structure, and shared-memory segment, equal in bytes to the value of **size** is created if:

key is set equal to **0** (IPCPVT).

or

key does not already have a shared-memory ID associated with it *and* the **shmflg** parameter is set equal to the constant IPCCRT.

The initial values of the data structure associated with a newly created shared-memory ID are listed later in this section under **Return Values**

In Pascal, **key** is of type integer.

In FORTRAN, **key** is of type INTEGER.

size
is the number of bytes in the shared-memory segment.

In Pascal, **size** is of type integer.

In FORTRAN **size** is of type INTEGER.

shmflg
specifies a set of conditions (options) governing the creation of a shared-memory data structure and the accessibility of the segment. The parameter value is that of one of the following options or is constructed from two or more of those options by logical ORing. The options are defined as constants in the Pascal and FORTRAN constants include files.

IPCCRT creates a data structure if one does not exist.

IPCEXL causes **SHMGET** to fail if IPCCRT is also set and the data structure already exists.

IRUSR permits the process that owns the data structure to read it.

IWUSR permits the process that owns the data structure to modify it.

IRGRP permits the group associated with the data structure to read it.

IWGRP permits the group associated with the data structure to modify it.

IROTH permits others to read the data structure.

IWOTH permits others to modify the data structure.

In Pascal, **shmflg** is of type integer.

In FORTRAN, **shmflg** is of type INTEGER.

Return Values

A shared-memory ID is returned upon successful completion of the call. The data structure associated with a newly created ID (smds; see Appendix C) is initialized. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **SHMGET** system routine, which in these examples returns a shared-memory identifier associated with the value of **key** ("red") returned by the call to the **ftok** system subroutine.

Pascal

```
procedure shmget1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red : integer;
  orange : st80;
  brown : char;

%include /usr/include/aildefs.inc

begin
  brown := 'm';
  orange := '/tmp/junk';
```

```
    blue := IPCCRT + IRUSR;
    red := p_ftok (orange, brown);
    green := p_shmget (red, 512, blue);
    writeln (green);
  end;
```

FORTRAN

```
      SUBROUTINE SHMGET1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSHMGET, FFTOK, BLUE, GREEN, RED
      CHARACTER BROWN, ORANGE(80)
      BROWN = 'm'
      ORANGE = '/tmp/junk '
      BLUE = IPCCRT + IRUSR
      RED = FFTOK (ORANGE, BROWN)
      GREEN = FSHMGET (RED, 512, BLUE)
      PRINT *, GREEN
      END
```

2.86 SHUTDOWN shut down part or all of a full-duplex connection

Description
The **SHUTDOWN** system call disables a specified connected socket from sending or receiving or both.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_shutdown (s, how);                                              ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FSHUTDOWN (S, HOW)                                                ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

s
is the descriptor of the socket that is to be shut down.

In Pascal, **s** is of type integer.

In FORTRAN, **s** is of type INTEGER.

how
specifies one of three options:

0 prevents further receives.

1 prevents further sends.

2 prevents further receives and sends.

In Pascal, **how** is of type integer.

In FORTRAN, **how** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SHUTDOWN** system routine, which in these examples, with **how** set to 0 (zero), disables the specified socket from receiving.

Pascal

```
procedure shutdown1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  s, green : integer;
  sv : intz;

%include /usr/include/aildefs.inc

begin
  s := p_socketpair (PF_UNIX, SOCK_DGRAM, 0, sv);
  if (s = -1) then showerror;
  green := p_shutdown (sv&lrbk., 0);
  writeln ('Shutdown returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE SHUTDOWN1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSHUTDOWN, FSOCKETPAIR, S, GREEN, SV(2)
      S = FSOCKETPAIR (PFUNIX, SKDGRAM,, 0, SV)
      IF (S .EQ. -1) CALL ERRORS
      GREEN = FSHUTDOWN (SV(1), 0)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.87 SIGACTION specify the action to be taken upon receipt of a signal

Description

The **SIGACTION** system call enables the calling process to examine or change the action to be taken when it receives a specified signal.

The signals that can be specified in a **SIGACTION** call are listed in the descriptions of the **sig** parameter.

Syntax

Pascal

```
p_sigaction (sig, act, oact);
```

FORTRAN

This system call is not available in FORTRAN.

Parameters

sig

is a number that specifies a particular signal. The signals that can be specified in a **SIGACTION** call are listed here and are defined in the Pascal constants include file. For more information about these signals, refer to Volume 1 of the *AIX Operating System Technical Reference*.

Signal	Signal Number	Event
SIGHUP	1	Hangup
SIGINT	2	Interrupt
SIGQT	3*	Quit
SIGILL	4*	Illegal instruction (not reset when caught)
SIGTRAP	5	Trace trap (not reset when caught)
SIGIOT	6	Abort process (see **FABORT** on page 2.21)
SIGEMT	7	EMT instruction
SIGFPE	8	Arithmetic exception, floating-point exception, or integer divide by zero.
SIGKIL	9	Kill (cannot be caught or ignored)
SIGBUS	10	Specification exception

SIGSEGV	11	Segmentation violation
SIGSYS	12	Bad parameter to system call
SIGPIPE	13	Write on pipe when there is no process to read it
SIGALRM	14	Alarm clock
SIGTERM	15	Software termination signal
SIGURG	16	Urgent condition on I/O channel
SIGSTOP	17	Stop (cannot be caught or ignored)
SIGSTP	18	Interactive stop
SIGCONT	19	Continue if stopped (cannot be caught or ignored)
SIGCHLD	20	To parent on child stop or exit
SIGPTTIN	21	Background read attempted from control terminal
SIGPTTOU	22	Background write attempted to control terminal
SIGIO	23	Input/output possible or completed
SIGXCPU	24	CPU time limit exceeded (see **setrlimit** in *AIX Operating System Technical Reference*)
SIGXFSZ	25	File size limit exceeded (see **setrlimit** in *AIX Operating System Technical Reference*)
reserved	26	
SIGMSG	27	Input data has been stored in the HFT monitor mode ring buffer
SIGWINCH	28	Window-size change
SIGPWR	29	Power-failure restart
SIGUSR1	30	User-defined signal 1
SIGUSR2	31	User-defined signal 2
SIGPROF	32	Profiling time alarm (see **GETITIME** on page 2.31)
SIGDANGER	33	System crash is imminent
SIGPROF	34	Virtual time alarm (see **SETITIME** on page 2.75)

reserved	35-58	
SIGGRANT	60	Grant HFT monitor access
SIGRETRACT	61	Relinquish HFT monitor access
SIGSOUND	62	An HFT sound control has completed execution
reserved	63	

Note: For more information about these signals, see *AIX Operating System Technical Reference*.

In Pascal, **sig** is of type integer.

act
if not nil, points to a structure that describes the action to be taken on receipt of the **sig** signal.

In Pascal, **act** is of type sigactptr.

oact
if not nil, points to a structure in which the signal action data in effect at the time of the **SIGACTION** system call is returned.

In Pascal, **oact** is of type sigactptr.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

Examples

The Pascal procedure that follows calls the **SIGACTION** system routine, which in this example returns data that was in effect at the time the interrupt signal (SIGINT) was issued. The data is returned in the parameter **oact**.

Pascal

```
procedure sigaction1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  rc : integer;
  oact : sigactptr;

%include /usr/include/aildefs.inc

begin
  rc := p_sigaction (2, nil, oact);
```

```
  writeln (rc);
end;
```

2.88 SIGBLOCK block one or more signals

Description

The **SIGBLOCK** system call blocks one or more specified signals until a subsequent **SIGSETMASK** "unblocks" them (see page 2.89 for a complete list of signals).

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_sigblock (mask);                                                  ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FSIGBLOCK (MASK)                                                    ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

mask
specifies the signal(s) to be blocked by logically ORing the parameter value with the previous signal mask of the calling process.

Note: To set the mask value, use a number equal to 2 (two) raised to the (signal-number - 1) power. For example, the mask value that will block **SIGNAL** 31 is 2¦0 (see page 2.92).

In Pascal, **mask** is of type integer.

In FORTRAN, **mask** is of type INTEGER.

Return Values

The value that the signal mask had prior to the **SIGBLOCK** call is returned upon successful completion of the call.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **SIGBLOCK** system routine, which in these examples blocks interrupt signals and illegal instruction signals that may be sent to the calling process. The return value printed out is equal to 2 (the previous masked blocked signal value) after the second call.

Pascal

```
 procedure sigblock1;

 const
   %include /usr/include/ailpconsts.inc
```

```
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;

%include /usr/include/aildefs.inc

begin
  red := p_sigblock (2);
  blue := p_sigblock (4);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE SIGBLOCK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSIGBLOCK, BLUE, RED
      RED = FSIGBLOCK (2)
      BLUE = FSIGBLOCK (4)
      PRINT *, BLUE
      END
```

2.89 SIGNAL specify the process response to a signal

Description

The **SIGNAL** system call sets the calling process to respond in one of three ways to the receipt of a signal:

"catch" the signal
ignore the signal; o
terminate its own execution (EXIT). Termination is the default event

The signals that can be specified in a **SIGNAL** call are listed in the descriptions of the **sig** and **action** parameters.

Syntax

Pascal

```
p_signal (sig, action, func);
```

FORTRAN

```
FSIGNAL (SIG, ACTION, FUNC)
```

Parameters

sig
is a number that specifies a particular signal. If a repeated signal arrives before the last one can be reset, it will not be caught (see **Notes**, item 2).

The signals that can be specified in a **SIGNAL** call are listed on the next two pages and are defined in the Pascal and FORTRAN constants include files.

act
if not nil, points to a structure that describes the action to be taken on receipt of the signal specified by the **sig** parameter. The signals that can be specified in a **SIGNAL** call are listed on the next two pages and are defined in the Pascal and FORTRAN constants include files. For more information about these signals, refer to Volume 1 of the *AIX Operating System Technical Reference*.

Signal	**Signal Number**	**Event**
SIGHUP	1	Hangup
SIGINT	2	Interrupt
SIGQT	3*	Quit
SIGILL	4*	Illegal instruction (not reset when caught)

SIGTRAP	5	Trace trap (not reset when caught)
SIGIOT	6	Abort process (see **FABORT** on page 2.21)
SIGEMT	7	EMT instruction
SIGFPE	8	Arithmetic exception, floating-point exception, or integer divide by zero.
SIGKIL	9	Kill (cannot be caught or ignored)
SIGBUS	10	Specification exception
SIGSEGV	11	Segmentation violation
SIGSYS	12	Bad parameter to system call
SIGPIPE	13	Write on pipe when there is no process to read it
SIGALRM	14	Alarm clock
SIGTERM	15	Software termination signal
SIGURG	16	Urgent condition on I/O channel
SIGSTOP	17	Stop (cannot be caught or ignored)
SIGSTP	18	Interactive stop
SIGCONT	19	Continue if stopped (cannot be caught or ignored)
SIGCHLD	20	To parent on child stop or exit
SIGPTTIN	21	Background read attempted from control terminal
SIGPTTOU	22	Background write attempted to control terminal
SIGIO	23	Input/output possible or completed
SIGXCPU	24	CPU time limit exceeded (see **setrlimit** in *AIX Operating System Technical Reference*)
SIGXFSZ	25	File size limit exceeded (see **setrlimit** in *AIX Operating System Technical Reference*)
reserved	26	
SIGMSG	27	Input data has been stored in the HFT-monitor-mode ring buffer
SIGWINCH	28	Window-size change

SIGPWR	29	Power-failure restart
SIGUSR1	30	User-defined signal 1
SIGUSR2	31	User-defined signal 2
reserved	35-38	

In Pascal, **action** is of type integer.

In FORTRAN, **action** is of type INTEGER.

func
is used when a signal is to be caught and **action** is set equal to SIGFNC. This parameter directs the receiving process of the signal to execute the function specified. The **func** parameter is given the value nil in Pascal and 0 (zero) in FORTRAN if the value of **action** is SIGDFL or SIGIGN.

When calling a function from Pascal or FORTRAN, the function name should be the parameter.

In Pascal, **func** is a function name.

In FORTRAN, **func** is a function name.

Return Values

The previous value of **action** is returned for the specified **sig** upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutines on the next page call the **SIGNAL** system routine. In this example, **sig** is assigned a value of 2 (SIGINT, interrupt signal). The **action** parameter is given the prescribed action SIGIGN, which causes the process to ignore the interrupt signal (that is, it does not terminate). The **func** parameter is sent as nil since no function address is needed in this instance.

Pascal

```
procedure signal1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red, yellow : integer;

%include /usr/include/aildefs.inc
```

```
  begin
    blue := 1;
    red := 2;
    yellow := p_signal (red, blue, nil);
    writeln (yellow)
  end;
```

FORTRAN

```
        SUBROUTINE SIGNAL1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FSIGNAL, BLUE, RED, YELLOW
        BLUE = 1
        RED = 2
        YELLOW = FSIGNAL (RED, BLUE, 0)
        PRINT *, YELLOW
        END
```

Notes

1. The SIGKIL signal cannot be caught and it cannot be ignored.

2. The **SIGVEC** system call provides an enhanced signal-handling capacity that avoids this difficulty (see page 2.95).

2.90 SIGPAUSE release a blocked signal and wait for an interrupt

Description
The **SIGPAUSE** system call resets the signal mask of the calling process and causes the calling process to wait for a signal to arrive. The arrival of the signal terminates the call and restores the signal mask to its previous value.

Note: This system call allows the masking of signals 1-31.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_sigpause (sigmask);                                             ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FSIGPAUSE (SIGMASK)                                               ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

sigmask
is the value to which the signal mask of the calling process is set when the call is issued.

In Pascal, **sigmask** is of type integer.

In FORTRAN, **sigmask** is of type INTEGER.

Return Values
If the signal is caught by the calling process and control is returned from the signal handler, the calling process resumes execution after the **SIGPAUSE** system call, which always returns the value -1 and sets an error code in **errno**.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **SIGPAUSE** system routine. In these examples, the first call is to the **ALARM** system routine, which sends a signal to the calling process after 10 seconds. The call to **SIGPAUSE** sets the signal mask to the value of the **sigsetmask** parameter ("blue") to block interrupts.

Pascal

```
  procedure sigpause1;

  const
    %include /usr/include/ailpconsts.inc
```

```
type
  %include /usr/include/ailtypes.inc
var
  blue, green, orange, red : integer;

%include /usr/include/aildefs.inc

begin
  orange := 10;
  green := p_alarm (orange);
  blue := 2;
  red := p_sigpause (blue)
end;
```

FORTRAN

```
      SUBROUTINE SIGPAUSE1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSIGPAUSE, FALARM, BLUE, GREEN, ORANGE, RED
      ORANGE = 10
      GREEN = FALARM (ORANGE)
      BLUE = 2
      RED = FSIGPAUSE (BLUE)
      END
```

2.91 SIGPROCMASK set the current signal mask

Description

The **SIGPROCMASK** system call changes the signal mask of the calling process.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_sigprocmask (how, set, oset);                                    ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   F_SIGPROCMASK (HOW, SET, OSET)                                     ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

how
specifies the manner in which the signal mask (the set of signals to be blocked) is defined. It may have one of three values:

SIG_BLOCK the resulting set is a union of the current set of signals and the signal set pointed to by the **set** parameter.
SIG_UNBLOCK the resulting set is the intersection of the current set of signals and the complement of the signal set pointed to by the **set** parameter.
SIGSETMASK the resulting set is the set of signals pointed to by the **set** parameter.

In Pascal, **how** is of type integer.

In FORTRAN, **how** is of type INTEGER.

set
points to a set of signals to be used to change the currently blocked set.

In Pascal, **set** is of type sigset_t.

In FORTRAN, **set** is an array of 4 elements of type INTEGER. This array corresponds to a Pascal data struncture defined in the ailtypes.inc file (see Appendix C) as follows:

SET(1) = set.setsize

SET(2) = set.sigs[1]

SET(3) = set.sigs[2]

SET(4) = sigmsk.sigs[3]

oset
is not nil, points to the space in which the call stores the signal

mask in effect at that time.

In Pascal, **oset** is of type sigset_t.

In FORTRAN, **oset** is an array of 4 elements of type INTEGER. This array corresponds to a Pascal data struncture defined in the ailtypes.inc file (see Appendix C) as follows:

OSET(1) = set.setsize

OSET(2) = set.sigs[1]

OSET(3) = set.sigs[2]

OSET(4) = sigmsk.sigs[3]

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SIGPROCMASK** system routine, which in these examples blocks signal 14 (alarm clock). The call to **SIGBLOCK** returns the previous mask value, which should be what it has just been set to (8192). This mask value is also printed out.

Pascal

```
procedure sigprocmask1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  orange, pink, red : integer;
  blue : sigset_t;

%include /usr/include/aildefs.inc

begin
  blue.setsize := 1;  := 8192;
  blue.sigs[1] := 8192;
  red := p_sigprocmask (SIG_SETMASK, blue, pink);
  writeln (red);
  orange := p_sigblock (0);
  writeln (orange);
end;
```

FORTRAN

```
      SUBROUTINE SIGPROCMASK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSIGPROCMASK, FSIGBLOCK, BLUE(4), ORANGE, PINK, RED
      BLUE(1) = 1
      BLUE(2) = 8192
      RED = FSIGPROCMASK (SIG_SETMASK, BLUE, PINK)
```

```
PRINT *, RED
ORANGE = FSIGBLOCK (0)
PRINT *, ORANGE
END
```

2.92 SIGSETMASK set the signal mask of the current process

Description
The **SIGSETMASK** system call sets the signal mask of the current process to a particular value, thereby specifying which signal will be blocked from receiving (that is, which signal the calling process will block).

Syntax

```
+--- Pascal -----------------------------------------------------+
¦                                                                ¦
¦   p_sigsetmask (mask);                                         ¦
¦                                                                ¦
¦                                                                ¦
+----------------------------------------------------------------+

+--- FORTRAN ----------------------------------------------------+
¦                                                                ¦
¦   FSIGSETMASK (MASK)                                           ¦
¦                                                                ¦
¦                                                                ¦
+----------------------------------------------------------------+
```

Parameters

mask
specifies the signal(s) to be blocked.

Note: To set the mask, use a number equal to 2 (two) raised to the (signal-number - 1) power. For example, the mask value that will block **SIGNAL** 31 is 2¦0.

In Pascal, **mask** is of type integer.

In FORTRAN, **mask** is of type INTEGER.

Return Values
The value that the signal mask had before **SIGBLOCK** was called is returned on successful completion of the **SIGSETMASK** call.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **SIGSETMASK** system routine, which in these examples blocks signal 14 (alarm clock). The call to **SIGBLOCK** returns the previous mask value, which should be what it has just been set to (8192). This mask value is also printed out ("orange").

Pascal

```
procedure sigsetmask1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
```

```
var
  blue, orange, red : integer;

%include /usr/include/aildefs.inc

begin
  blue := 8192;
  red := p_sigsetmask (blue);
  writeln (red);
  orange := p_sigblock (0);
  writeln (orange);
end;
```

FORTRAN

```
      SUBROUTINE SIGSETMASK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSIGSETMASK, FSIGBLOCK, BLUE, ORANGE, RED
      BLUE = 8192
      RED = FSIGSETMASK (BLUE)
      PRINT *, RED
      ORANGE = FSIGBLOCK (0)
      PRINT *, ORANGE
      END
```

2.93 SIGSTACK set and get a signal-stack context

Description

The **SIGSTACK** system call defines an alternate stack on which signals are to be processed.

Warning: A signal stack does not automatically increase in size as a normal stack does. If the stack overflows, unpredictable results may occur.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_sigstack (instack, outstack);                                    ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   This system call is not available in FORTRAN.                      ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

instack
points to a signal-stack data structure if the parameter value is **not** nil. If the parameter value **is** nil, then the signal-stack state is not set.

instack is of type stackptr.

outstack
points to a signal-stack data structure if the parameter value is **not** nil. If the parameter value **is** nil, the previous signal-stack state is not reported.

outstack is of type stackptr.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

The return value is of type integer

Examples

The Pascal procedure on the next page calls the **SIGSTACK** system routine. In this example the values being passed to **SIGSTACK** are the **instack** ("yellow") and **outstack** ("green") parameters. The example merely shows the proper call: it neither sets a new stack nor stores the old (both parameters are set to nil).

Pascal

```
  procedure sigstack1;
```

```
const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;
  green, yellow : stackptr;

%include /usr/include/aildefs.inc

begin
  new (yellow);
  new (green);
  new (yellow@.ss_sp);
  new (green@.ss_sp);
  yellow@.ss_sp := nil;
  green@.ss_sp := nil;
  red := p_sigstack (green, yellow);
  writeln (red);
end;
```

2.94 SIGSUSPEND reset the signal mask and wait for an interrupt

Description

The **SIGSUSPEND** system call resets the signal mask of the calling process and causes the calling process to wait for a signal to arrive. The arrival of the signal terminates the call and restores the signal mask to its previous value.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_sigsuspend (sigmsk);                                            ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FSIGSUSPEND (SIGMSK)                                              ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

sigmsk
is the value to which the signals mask of the calling process is set when the call is issued.

In Pascal, **sigmsk** is of type sigset_t.

In FORTRAN, **sigmsk** is an array of 4 elements of type INTEGER. This array corresponds to a Pascal data struncture defined in the ailtypes.inc file (see Appendix C) as follows:

SIGMSK(1) = sigmsk.setsize

SIGMSK(2) = sigmsk.sigs[1]

SIGMSK(3) = sigmsk.sigs[2]

SIGMSK(4) = sigmsk.sigs[3]

Return Values

If the signal is caught by the calling process and control is returned from the signal handler, the calling process resumes execution after the **SIGSUSPEND** system call, which always returns the value -1 and sets an error code in **errno**.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **SIGSUSPEND** system routine. In these examples, the first call is to the **ALARM** system routine, which sends a signal to the calling process after 10 seconds. The call to **SIGSUSPEND** sets the signal mask to the value of the **sigmsk** parameter ("blue") to block interrupts.

Pascal

```
procedure sigsuspend1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green, orange, red : integer;
  blue : sigset_t;

%include /usr/include/aildefs.inc

begin
  orange := 10;
  green := p_alarm (orange);
  blue := setsize := 1;
  blue := sig[1] := 3;
  red := p_sigsuspend (blue)
end;
```

FORTRAN

```
      SUBROUTINE SIGSUSPEND1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSIGSUSPEND, FALARM, BLUE(4), GREEN, ORANGE, RED
      ORANGE = 10
      GREEN = FALARM (ORANGE)
      BLUE(1) = 1
      BLUE(2) = 3
      RED = FSIGSUSPEND (BLUE)
      END
```

2.95 SIGVEC select signal-handling facilities

Description

The **SIGVEC** system call allows the user to select standard or enhanced signal-handling facilities. Like the **SIGNAL** call, it specifies the action to be taken on receipt of a given signal.

Warning: The **SIGVEC** call does not check the validity of the sv_handler pointer. If this pointer is pointing outside the address space of the process, a memory-fault message is returned to the process when the system attempts to use the signal handler.

Syntax

Pascal

```
p_sigvec (sig, invec, outvec);
```

FORTRAN

This system call is not available in FORTRAN.

Parameters

sig
: is the identifying number of a signal (See page 2.89 for a complete list of signals.).

  sig is of type integer.

invec
: specifies a handler routine and mask for use in delivering a signal when the parameter value is **not** nil. When the parameter value **is** nil, the signal-handler information is not set. The value of the sv_onstak field of the **invec** record specifies one of three options:

  0 the enhanced signal and the process signal on the process stack are used.

  1 the enhanced signal and the process signal on a separate stack are used.

  2 standard signal processing is used.

  invec is of type sigvecptr.

outvec
: points to a record where the previous handling information for the signal in the structure is stored, when it is **not** nil. Information for the signal is stored in the **SIGVEC** data structure pointed to by **outvec**. If the value of the **outvec** parameter is nil, the previous signal-handler information is not reported.

  outvec is of type sigvecptr.

Return Values

There is no return value from a successful **SIGVEC** call.

Examples

The Pascal procedure that follows calls the **SIGVEC** system routine. In these examples, the value passed to **SIGVEC** by the parameter **sig** ("yellow"), specifies signal (2) and the **invec** and **outvec** parameters ("blue" and "red", respectively). The default action is specified by the variable "orange"; the **invec** and **outvec** parameters are set equal to 'nil' because they are not necessary for this action.

Pascal

```
procedure sigvec1;

const
  %include usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : sigvecptr;
  green, orange, yellow : integer;

%include /usr/include/aildefs.inc

begin
  yellow := 2;
  orange := SIGDFL;
  new (blue);
  new (red);
  red := nil;
  blue@.sv_handler := nil;
  blue@.sv_mask := 0;
  blue@.sv_onstack := 0;
  green := p_sigvec (yellow, orange, blue, red)
end;
```

2.96 SOCKET create an endpoint for communication

Description

The **SOCKET** system call creates an endpoint for communication and returns a descriptor.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_socket (domain, ttype, protocol)                                 ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FSOCKET (DOMAIN, TTYPE, PROTOCOL)                                  ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

domain
specifies one of two "domains" of communication:

PF_UNIX AIX path names

PF_INET ARPA internet addresses.

Note: In FORTRAN, the underscore is omitted (for example, "PFUNIX").

In Pascal, **domain** is of type integer.

In FORTRAN, **domain** is of type INTEGER.

ttype
specifies one of two types of communication semantics:

SOCK_STREAM sequenced streams with a transmission mechanism for out-of-band data.

Note: In FORTRAN, use SKSTRM.

SOCK_DGRAM datagrams, or connectionless messages, of a fixed maximum length (usually small).

Note: In FORTRAN, use SKDGRAM.

In Pascal, **ttype** is of type integer.

In FORTRAN, **ttype** is of type INTEGER.

protocol
specifies a particular protocol to be used with the socket.

In Pascal, **protocol** is of type integer.

In FORTRAN, **protocol** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SOCKET** system routine, which in these examples is issued with domain set to "PF_UNIX", type to "SOCK_STREAM", and protocol to 0. A socket descriptor is returned in "green".

Pascal

```
procedure socket1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;

%include /usr/include/aildefs.inc

begin
  green := p_socket (PF_UNIX, SOCK_STREAM, 0);
  writeln ('Socket returned: ', green : 2);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE SOCKET1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSOCKET, GREEN
      GREEN = FSOCKET (PFUNIX, SKSTRM, 0)
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.97 SOCKETPAIR create a pair of connected sockets

Description

The **SOCKETPAIR** system call creates an unnamed pair of connected sockets.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_socketpair (domain, type, protocol, sv)                        ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FSOCKETPAIR (DOMAIN, TYPE, PROTOCOL, SV)                         ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

domain
specifies one of two "domains" of communication:

PF_UNIX AIX path names

PF_INET ARPA internet addresses.

Note: In FORTRAN, the underscore is omitted (for example, "PFUNIX").

In Pascal, **domain** is of type integer.

In FORTRAN, **domain** is of type INTEGER.

type
specifies one of two types of communication semantics:

SOCK_STREAM sequenced streams with a transmission mechanism for out-of-band data.

Note: In FORTRAN, use SKSTRM.

SOCK_DGRAM datagrams, or connectionless messages of a fixed maximum length (usually small).

Note: In FORTRAN, use SKDGRAM.

In Pascal, **type** is of type integer.

In FORTRAN, **type** is of type INTEGER.

protocol
specifies a particular protocol to be used with the socketpair.

In Pascal, **protocol** is of type integer.

In FORTRAN, **protocol** is of type INTEGER.

sv

is an array in which two descriptors are returned upon completion of the call.

In Pascal, **sv** is of type int2 (defined as an array[1..2] of integer in the ailtypes.inc file; see Appendix C).

In FORTRAN, **sv** is an array of type integer with two elements.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **SOCKETPAIR** system routine, which in these examples is issued with domain set to "PF_UNIX" and type to "SOCK_STREAM". The **protocol** parameter is optional. The socketpair descriptors are returned in sv[1] and sv[2].

Pascal

```
procedure socketpair1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
  sv : int2;

%include /usr/include/aildefs.inc

begin
  green := p_socketpair (PF_UNIX, SOCK_STREAM, 0, sv);
  if (green = -1) then showerror;
end;
```

FORTRAN

```
      SUBROUTINE SOCKETPAIR1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSOCKETPAIR, SV(2), GREEN
      GREEN = FSOCKETPAIR (PFUNIX, SKSTRM, 0, SV)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      END
```

2.98 STATX, FSTATX, STAT, FSTAT, LSTAT, FULLSTAT, FFULLSTAT return the status

Description

These calls obtain status information about files, including hidden directories and symbolic links.

STATX and **FSTATX** obtain status information about a specified file, hidden directory, or symbolic link.

STAT and **FSTAT** obtain status information about a specified file.

LSTAT obtains status information about a specified symbolic link.

FULLSTAT and **FFULLSTAT** obtain status information about a specified file.

Note: **STATX** and **FSTATX** replace five system calls: **STAT**, **FSTAT**, **LSTAT**, **FULLSTAT**, and **FFULLSTAT**. All five calls have been included in this manual for reasons of compatibility (see **Notes** at the end of this section).

Note: Only the file owner and the super-user may issue these calls.

Syntax

Pascal

```
p_statx (path, buf, len, cmd);

p_fstatx (fildes, buf, len, cmd);

p_stat (path, buf);

p_fstat (fildes, buf);

p_lstat (path, buf);

p_fullstat (path, cmd, buf);

p_ffullstat (fildes, cmd, buf);
```

FORTRAN

```
FFSTATX (PATH, BUF, LEN, CMD)

FFFSTATX (FILDES, BUF, LEN, CMD)

FFSTAT (PATH, BUF)

FFFSTAT (FILDES, BUF)

FLSTAT (PATH, BUF)

FFFULLSTAT (PATH, CMD, BUF)
```

```
¦   FFFFULLSTAT (FILDES, CMD, BUF)                                      ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

path
is used only in the **STATX**, **STAT** and **LSTAT** system calls. It specifies the file whose status is to be checked.

In Pascal, **path** is a string variable or constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

fildes
is used only in the **FSTATX**, **FSTAT**, **FULLSTAT**, and **FFULLSTAT** system calls. It is a descriptor obtained from a successful **FCNTL**, **OPEN**, **PIPE**, **SOCKET**, or **SOCKETPAIR** system call.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

buf
is required for all five system calls. It points to a buffer where status information about the specified file is returned.

In Pascal, **buf** is of type statrec.

In FORTRAN, **buf** is the name of an array of 30 elements of type INTEGER. This array corresponds to the Pascal data structure--defined in the ailtypes.inc file (Appendix C)--as follows:

BUF(1) = buf.st_dev

BUF(2) = buf.st_ino

BUF(3) = buf.st_mode

BUF(4) = buf.st_nlink

BUF(5) = buf.st_uid

BUF(6) = buf.st_gid

BUF(7) = buf.st_rdev

BUF(8) = buf.st_size

BUF(9) = buf.st_atime

BUF(10) = buf.st_mtime

BUF(11) = buf.st_ctime

BUF(12) = buf.fst.uid_raw

BUF(13) = buf.fst.gid_raw

BUF(14) = buf.fst_type

BUF(15) = buf.uid_rev_tag

BUF(16) = buf.gid_rev_tag

BUF(17) = buf.fst_other_gid_list

BUF(18) = buf.fst_other_gid_count

BUF(19) = buf.fst_vfs

BUF(20) = buf.fst_nid

BUF(21) = buf.fst_flag

BUF(20) = buf.fst_i_gen

BUF(23...BUF(30) = buf.fst_reserved[1]...buf.fst_reserved[8]

len
specifies the amount of information to be returned.

In Pascal, **len** is of type integer.

In FORTRAN, **len** is of type INTEGER.

cmd
determines the interpretation of **path**:

STX_LINK specifies that **path** identifies a symbolic link.

STX_HIDDEN specifies that **path** identifies a hidden directory.

STX_MOUNT specifies that **path** identifies a mounted-over file or directory.

Note: In FORTRAN, the underscore is omitted (for example, "STXLINK").

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **STATX** system routine. In these examples, information about the file specified by the **path** parameter ("blue") is returned in the **buf** parameter

("yellow"). The value of the file mode for file /usr/include/aildefs.inc is the value printed out.

Pascal

```
procedure statx1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  red : integer;
  blue : st80;
  yellow : statrec;

%include /usr/include/aildefs.inc

begin
  blue := '/usr/include/aildefs.inc';
  red := p_statx (blue, yellow, STATSIZE, 0);
  writeln (yellow@.st_mode);
end;
```

FORTRAN

```
      SUBROUTINE STATX1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FFSTATX, RED, YELLOW(18)
      CHARACTER*80 BLUE
      BLUE = '/usr/include/aildef.inc '
      RED = FFSTATX (BLUE, YELLOW, STATSIZE, 0)
      PRINT *, YELLOW(3)
      END
```

Notes

The following interfaces provide compatibility with programs written for AIX/RT or other versions of the UNIX operating system.

stat (path, stbuf) is equivalent to

statx (path, buf, STATSIZE, O)

lstat (path, buf)

is equivalent to

statx (path, buf, STATSIZE, STX_LINK)

fstat (fildes, buf)

is equivalent to

fstatx (fildes, buf, STATSIZE, O)

fullstat (path, cmd, buf)

is equivalent to

```
statx (path, buf, FULLSTATSIZE, cmd)
```

```
ffullstat (fildes, cmd, buf)
```

is equivalent to

```
statx (fildes, buf, FULLSTATSIZE, cmd)
```

2.99 STIME set the system clock

Description
The **STIME** system call sets the system's internal clock to a time and date that are calculated from a value specified in the call.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                        ¦
¦   p_stime (tp);                                                        ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                        ¦
¦   FSTIME (TP)                                                          ¦
¦                                                                        ¦
¦                                                                        ¦
+------------------------------------------------------------------------+
```

Parameters

tp
is the number of seconds that have elapsed since 00:00:00 January 1, 1970 GMT. Given this number, the routine calculates the time and date and resets the system's internal clock accordingly.

In Pascal, **tp** is of type integer.

In FORTRAN, **tp** is of type INTEGER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call is issued by anyone other than the super-user or if it fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine on the next page call the **STIME** system routine. The value of "cronos" is the interval (in seconds) between 00:00:00 January 1, 1970 GMT and the time to which the system clock is to be set. The return value of the call is in the variable "titan".

Pascal

```
  procedure stime1;

  const
    %include /usr/include/ailpconsts.inc
  type
```

```
    %include /usr/include/ailtypes.inc
  var
    cronos, titan : integer;

  %include /usr/include/aildefs.inc

  begin
    cronos := 31536000;
    titan  := p_stime (cronos);
    writeln (titan);
  end;
```

FORTRAN

```
        SUBROUTINE STIME1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FSTIME, CRONOS, TITAN
        CRONOS = 31536000
        TITAN  = FSTIME (CRONOS)
        PRINT *, TITAN
        END
```

2.100 SYMLINK create a symbolic link to a file

Description

The **SYMLINK** system call creates a symbolic link to a file.

Syntax

```
+--- Pascal -----------------------------------------------------------+
|                                                                      |
|   p_symlink (path1, path2);                                          |
|                                                                      |
|                                                                      |
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
|                                                                      |
|   FSYMLINK (PATH1, PATH2);                                           |
|                                                                      |
|                                                                      |
+----------------------------------------------------------------------+
```

Parameters

path1
is the name of the existing file to which a link is created. If **path1** is not a full pathname (that is, does not begin with "/"), it is evaluated in the context of path2, not the current working directory.

In Pascal, **path1** is a string variable or constant of type st80.

In FORTRAN, **path1** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

path2
is the name of the file created.

In Pascal, **path2** is a string variable or constant of type st80.

In FORTRAN, **path2** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine on the next page call the **SYMLINK** system routine, which in these examples creates a symbolic link to a physical file (/usr/include/aildefs.inc) by creating /bushel/light/hide. After the successful completion of the call, the two files are unlinked by a call to **UNLINK**.

Pascal

```
procedure symlink1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
  path1, path2 : st80;

%include /usr/include/aildefs.inc

begin
  path1 := '/usr/include/aildefs.inc';
  path2 := '/bushel/light/hide';
  green := p_symlink (path1, path2);
  writeln ('Symlink returned: ', green : 2);
  if (green = -1) then showerror;
  green := p_unlink (path2);
end;
```

FORTRAN

```
      SUBROUTINE SYMLINK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FSYMLINK, FUNLINK, GREEN
      CHARACTER*80 P1, P2
      P1 = '/usr/include/aildefs.inc '
      P2 = '/bushel/light/hide '
      GREEN = FSYMLINK (P1, P2)
      PRINT *, GREEN
      IF (GREEN .EQ. -1) CALL ERRORS
      GREEN = FUNLINK (P2)
      END
```

2.101 SYNC update a file system

Description
The **SYNC** system call writes modified information in core memory to disk, including modified super-blocks, i-nodes, and delayed block I/O.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_sync;                                                            ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FFSYNC ()                                                          ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

This system call has no parameters.

Return Values
The write operation may be scheduled but is not necessarily complete upon return from the **SYNC** call, and no value is returned.

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **SYNC** system routine. In these examples, all information in memory that should be on disk is written to disk.

Pascal

```
  procedure sync1;

  const
    %include /usr/include/ailpconsts.inc
  var
    blue : integer;

  %include /usr/include/aildefs.inc

  begin
    blue := p_sync
  end;
```

FORTRAN

```
        SUBROUTINE SYNC1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FFSYNC, BLUE
        BLUE = FFSYNC ()
        END
```

2.102 TIME get the system time

Description

The **TIME** system call returns the length of the interval (in seconds) from 00:00:00 Jan. 1, 1970 GMT to the current (system) time.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_time (tloc);                                                     ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FTIME (TLOC)                                                       ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

tloc
is a variable that receives the length of the interval (in seconds from 00:00:00 Jan. 1, 1970 GMT to the current time) upon return from the call.

In Pascal, **tloc** is of type integer.

In FORTRAN, **tloc** is of type INTEGER.

Return Values

The current time is returned upon successful completion of the call. When the value returned is other than 0 (zero), it is also stored in the location to which **tloc** points.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **TIME** system routine. The length of the interval, expressed in seconds, is returned in the variable "perdu". The return value of the call is in the variable "temps".

Pascal

```
  procedure time1;

  const
    %include /usr/include/ailpconsts.inc
  type
    %include /usr/include/ailtypes.inc
  var
    temps, perdu : integer;
```

```
%include /usr/include/aildefs.inc

begin
  temps := p_time (perdu);
  writeln (perdu);
end;
```

FORTRAN

```
      SUBROUTINE TIME1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FTIME, TEMPS, PERDU
      TEMPS = FTIME (PERDU)
      PRINT *, PERDU
      END
```

2.103 TIMES get the process times

Description
The **TIMES** system call returns time-accounting information about the current process and about the terminated child processes of the current process.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_times (buf);                                                     ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+

+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FTIMES (BUF)                                                       ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

buf
is a pointer to a data structure in which information about the current process times is placed.

In Pascal, **buf** is of type tms.

In FORTRAN, **buf** is an array(4) of type INTEGER. This array corresponds to the Pascal data structure--defined in in the ailtypes.inc file (Appendix C)--as follows:

BUF(1) = buf.tms_utime

BUF(2) = buf.tms_stime

BUF(3) = buf.tms_cutime

BUF(4) = buf.tms_cstime

Return Values
The elapsed time from a system-defined reference date to the current process time is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow issue a **TIMES** system call. A child process is created by a call to **FORK**. The return value is in the variable "green". The call to **TIMES** stores information in the buffer "colors". Both examples print the value in the tms_stime field, which is the CPU time used by the system on behalf of the calling process.

Pascal

```
procedure times1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, red : integer;
  colors : tms;

%include /usr/include/aildefs.inc

begin
  green := p_fork;
  if green = 0 then
    red := p_execl ('/bin/sh', 'sh', '-c' , 'date', '')
  else
    begin
      blue := 0 ;
      green := p_wait (blue);
      red := p_times (colors);
      writeln ('stime     ', colors.tms_stime);
    end
end;
```

FORTRAN

```
      SUBROUTINE TIMES1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FTIMES, FEXECL, FFORK, FWAIT
      INTEGER BLUE, COLORS(4), GREEN, RED
      GREEN = FFORK ()
      IF (GREEN .EQ. 0) THEN
         RED = FEXECL ('/bin/sh ', 'sh ', '-c ', 'date ', ' ')
      ELSE
         BLUE = 0
         GREEN = FWAIT (BLUE)
         RED = FTIMES (COLORS)
         PRINT *, 'stime     ', COLORS(2)
      ENDIF
      END
```

2.104 ULIMIT get and set process limits

Description
The **ULIMIT** system call controls the limits of a process file.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

Pascal

```
p_ulimit (cmd, newlim);
```

FORTRAN

```
FULIMIT (CMD, NEWLIM)
```

Parameters

cmd
is a constant or a variable that can have one of the following values:

1 gets the process file-size limit.

2 sets the limit of the file size of the process to the value of **newlim** (see next parameter).

Note: Any process may decrease the limit, but only a process with an effective user ID of super-user may increase the limit.

3 retrieves the maximum possible break value (see **BRK** on page 2.7).

In Pascal, **cmd** is of type integer.

In FORTRAN, **cmd** is of type INTEGER.

newlim
is used only with **cmd** option 2 to increment the limit.

In Pascal, **newlim** is of type integer.

In FORTRAN, **newlim** is of type INTEGER.

Return Values
A nonnegative value is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **ULIMIT** system routine, which in these examples returns the maximum possible break value (specified by the **cmd** parameter value of 3) in the variable "blue".

Pascal

```
procedure ulimit1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue : integer;

%include /usr/include/aildefs.inc

begin
  blue := p_ulimit (3, 0);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE ULIMIT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FULIMIT, BLUE
      BLUE = FULIMIT (3, 0)
      PRINT *, BLUE
      END
```

2.105 UMASK get and set a file-creation-mode mask

Description

The **UMASK** system call sets a mask that is used whenever a file is created by a **CREAT** or **MKNOD** call. The access mode of the newly created file (see **CHMOD** on page 2.10) is set to the value of **cmask**. Only the low-order nine bits of the mask (the protection bits) participate.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_umask (cmask);                                                    ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+

+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FUMASK (CMASK)                                                      ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

cmask
is the boolean complement of the new file's access mode.

In Pascal, **cmask** is of type integer.

In FORTRAN, **cmask** is of type INTEGER.

Return Values

The previous value of the mask is returned upon successful completion of the call. The initial value of the mask is 0 (zero), specifying"no restrictions."

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **UMASK** system routine with the value of the **cmask** parameter ("red") equal to 0 (zero). This value specifies the elimination of all restrictions on the file-creation mode. The value printed out is the previous value of the mask.

Pascal

```
procedure umask1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;
```

```
%include /usr/include/aildefs.inc

begin
  red := 0;
  blue := p_umask (red);
  writeln (blue);
end;
```

FORTRAN

```
      SUBROUTINE UMASK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FUMASK, BLUE, RED
      RED = 0
      BLUE = FUMASK (RED)
      PRINT *, BLUE
      END
```

2.106 UNAME, UNAMEX get the name of the current operating system

Description

The **UNAME** and **UNAMEX** system calls retrieve and store information that identifies the current operating system. They store this information in a data structure specified in the call.

The **UNAMEX** call is used in local area networks where a binary node is appropriate.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                     ¦
¦   p_uname (name);                                                   ¦
¦                                                                     ¦
¦   p_unamex (xname);                                                 ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                     ¦
¦   FUNCTION FUNAME (NAME)                                            ¦
¦                                                                     ¦
¦   FUNCTION FUNAMEX (XNAME)                                          ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

name
is used only with the **UNAME** call. It points to the appropriate data structure (unam).

In Pascal, **name** is of type unam.

In FORTRAN, **name** is an array(5) of type CHARACTER*32.

xname
is used only with the **UNAMEX** call. It points to the appropriate data structure (xunam).

In Pascal, **xname** is of type xunam.

In FORTRAN, **xname** is an array(4) of type INTEGER.

Return Values

A nonnegative number is returned upon successful completion of the call (see **Notes**). The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow print the name of the current operating system. The return value for **UNAME** is in the variable "nemo". Other information returned concerning the current operating system is located in the four remaining fields of the record "verne".

The **UNAMEX** call, which is used in a local-area-network environment, returns the binary node number in a variable parameter of type xunam.

Pascal

```
  procedure uname1;

  const
    %include /usr/include/ailpconsts.inc
  type
    %include /usr/include/ailtypes.inc
  var
    nemo : integer;
    verne : unam;

  %include /usr/include/aildefs.inc

  begin
    nemo := p_uname (verne);
    writeln (verne.sysname);
  end;
```

FORTRAN

```
        SUBROUTINE UNAME1
        INCLUDE (/usr/include/ailfconsts.inc)
        INTEGER FUNAME, NEMO
        CHARACTER*32 VERNE(5)
        NEMO = FUNAME (VERNE)
        PRINT *, VERNE(1)
        END
```

Notes
If the unamx.nid field of the parameter's return value is a negative number, add 4 294 967 296 to that number to obtain the correct value.

2.107 UNLINK delete a directory entry

Description
The **UNLINK** system call deletes the directory entry of a specified file.

Syntax

```
+--- Pascal ------------------------------------------------------------+
¦                                                                       ¦
¦   p_unlink (path);                                                    ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

```
+--- FORTRAN -----------------------------------------------------------+
¦                                                                       ¦
¦   FUNLINK (PATH)                                                      ¦
¦                                                                       ¦
¦                                                                       ¦
+-----------------------------------------------------------------------+
```

Parameters

path
is the name of the file to be deleted.

In Pascal, **path** is a string variable or constant of type st80,

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type INTEGER

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **UNLINK** system routine, which in these examples removes the directory entry specified in the **path** parameter ("blue"), assuming that file /tmp/xxx exists.

Pascal

```
procedure unlink1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  yellow : integer;
  blue : st80;

%include /usr/include/aildefs.inc
```

```
begin
  blue := '/tmp/xxx';
  yellow := p_unlink (blue);
  writeln (yellow);
end;
```

FORTRAN

```
      SUBROUTINE UNLINK1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FUNLINK, YELLOW
      CHARACTER*80 BLUE
      BLUE = '/tmp/xxx '
      YELLOW = FUNLINK (BLUE)
      PRINT *, YELLOW
      END
```

2.108 USRINFO get and set user information

Description

The **USRINFO** system call gets and sets information about the owner of the calling process.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_usrinfo (cmd, buf, count);                                     ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FUSRINFO (CMD, BUF, COUNT)                                       ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

cmd
is a constant or a variable with two possible arguments (SETUINF or GETUINF) as defined in the Pascal and FORTRAN constants include files.

In Pascal, **cmd** is of type integer.

In FORTRAN, **cmd** is of type INTEGER.

buf
is a pointer to a user buffer. The length of this buffer, in bytes, is usually equal to the constant INFSIZ(64).

In Pascal, **buf** is of type charinfsiz.

In FORTRAN, **buf** is a user-defined array of type CHARACTER.

count
is the number of bytes to be copied from or to the user buffer.

In Pascal, **count** is of type integer.

In FORTRAN, **count** is of type INTEGER.

Return Values

A nonnegative number indicating the number of bytes read is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **USRINFO** system routine, which gets information about the owner of the current process. In these examples, the information is written to the array pointed to (Pascal) or specified by (FORTRAN) the variable "yellow". The number of bytes written to the array is returned in the variable "blue". Note that, in Pascal, "yellow" is the user-defined array (of type usrary) pointed to by usrptr.

Pascal

```
procedure usrinfo1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;
  yellow : charinfsiz;

begin
  blue := p_usrinfo (GETINF, yellow, INFSIZ);
  writeln (blue);
  for red := 1 to blue do
    write (yellow[red]);
    writeln;
end;
```

FORTRAN

```
      SUBROUTINE USRINFO1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FUSRINFO, BLUE
      CHARACTER*64 YELLOW
      BLUE = FUSRINFO (GETINF, YELLOW, INFSIZ)
      PRINT *, BLUE
      PRINT *, YELLOW (1 : BLUE)
      END
```

2.109 USTAT get file-system information

Description
The **USTAT** system call retrieves and stores information about a mounted file system.

Syntax

```
+--- Pascal ----------------------------------------------------------+
¦                                                                     ¦
¦   p_ustat (dev, buf);                                               ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

```
+--- FORTRAN ---------------------------------------------------------+
¦                                                                     ¦
¦   FUSTAT (DEV, ABUF, BBUF)                                          ¦
¦                                                                     ¦
¦                                                                     ¦
+---------------------------------------------------------------------+
```

Parameters

dev
is the ID of the device corresponding to the element *strdev* of the data structure returned by **STAT**.

In Pascal, **dev** is of type integer.

In FORTRAN, **dev** is of type INTEGER.

buf
is the pointer to the data structure that holds the retrieved information.

In Pascal, **buf** if of type ustatrec.

In FORTRAN, **buf** is divided into two parameters:

- **abuf** is an array(2) of type INTEGER.
- **bbuf** is an array(2,6) of type CHARACTER.

Return Values
The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **USTAT** system routine. In these examples, information about the device specified by the **dev** parameter ("blue") is returned in the **buf** parameter ("yellow"). The value assigned to **dev**(1) specifies /dev/hd1. Normally this parameter value is obtained from a field of the information returned by a **STAT** call.
Pascal

```
procedure ustat1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, red : integer;
  yellow : ustatrec;

%include /usr/include/aildefs.inc

begin
  blue := 1;
  red := p_ustat (blue, yellow);
  writeln (red);
end;
```

FORTRAN

```
      SUBROUTINE USTAT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FUSTAT, BLUE, GREEN(2), RED
      CHARACTER YELLOW(2,6)
      BLUE = 1
      RED = FUSTAT (BLUE, GREEN, YELLOW)
      PRINT *, RED
      END
```

2.110 UTIME set the file times

Description

The **UTIME** system call sets the access and modification times of a specified file. The 'i-node changed' time of the file is set to the current time.

Note: Only users with an effective user ID of super-user may issue this call.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_utime (path, times);                                             ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FUTIME (PATH, TIMES)                                               ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

path
is the name of the file whose times are to be set.

In Pascal, **path** is a string variable or a constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

times
is a pointer to a two-element array. The first element holds the new accessed time. The second element holds the new updated time.

In Pascal, **times** is of type utimptr.

In FORTRAN, **times** is the name of an array consisting of two elements of type INTEGER.

Note: If **times** is given the value nil in Pascal or -1 in FORTRAN, the access and modification **times** of the file in **path** are set equal to the current time.

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **UTIME** system routine. In these examples, the access and modification times of the file specified by the **path** parameter ("blue") are set to the current time.

Pascal

```
procedure utime1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  red : integer;
  blue : st80;
  yellow : utimptr;

%include /usr/include/aildefs.inc

begin
  blue := '/usr/include/ailtypes.inc';
  yellow := nil;
  red := p_utime (blue, yellow);
  writeln (red);
end;
```

FORTRAN

```
      SUBROUTINE UTIME1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FUTIME, RED, YELLOW(2)
      CHARACTER*80 BLUE
      BLUE = '/usr/include/ailtypes.inc '
      YELLOW(1) = -1
      RED = FUTIME (BLUE, YELLOW)
      PRINT *, RED
      END
```

2.111 UTIMES set the file times

Description

The **UTIMES** system call sets the accessed and updated times of a specified file to specified values.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_utimes (ffile, tvp);                                             ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FUTIMES (FFILE, TVP)                                               ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

ffile
is the name of the file whose times are to be set.

In Pascal, **ffile** is a string variable or a constant of type st80.

In FORTRAN, **path** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

tvp
contains the updated times.

In Pascal, **tvp** is of type timeval2 (an array of two timeval records).

In FORTRAN, **tvp** is an integer array of four elements. This array corresponds to the Pascal data structure--defined in the ailtypes.inc file (Appendix C)--as follows:

TVP(1) = tvp[1].tv_sec

TVP(2) = tvp[1].tv_usec

TVP(3) = tvp[2].tv_sec

TVP(4) = tvp[2].tv_usec

Return Values

The value 0 is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine that follow call the **UTIMES** system routine.

Pascal

```
procedure utimes1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  green : integer;
  ffile : st80;
  tvp : timeval2;

%include /usr/include/aildefs.inc

begin
  ffile := '/tmp/junk';
  green :- p_open (ffile, CREATE, 0);
  tvp[1].tv_sec := 1;
  tvp[1].tv_usec := 2;
  tvp[2].tv_sec := 3;
  tvp[2].tv_usec := 4;
  green := p_utimes (ffile, tvp);
    if green = -1 then
      writeln ('Utimes: ERROR')
    else
      writeln ('Utimes: ok ');
  green := p_unlink (ffile);
end;
```

FORTRAN

```
      SUBROUTINE UTIMES1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FUTIMES, FOPEN, TVP(4), GREEN
      CHARACTER*80 FFILE
      FFILE = '/tmp/junk '
      GREEN = FOPEN (FFILE, CREATE, 0)
      TVP(1) = 1
      TVP(2) = 2
      TVP(3) = 3
      TVP(4) = 4
      GREEN = FUTIMES (FFILE, TVP)
        IF (GREEN .EQ1. -1) THEN
          PRINT *, 'UTIMES: ERROR'
          CALL ERRORS
        ELSE
          PRINT *, 'UTIMES: OK'
        ENDIF
      END
```

2.112 WAIT, WAIT3 wait for a child process to terminate

Description

The **WAIT** and **WAIT3** system calls cause the calling process to delay until a signal is received or until one of the child processes terminates or stops in a trace mode. However, the routine does not delay the calling process if a child process that has *not* been waited for has already stopped or terminated before the call was issued.

WAIT3 returns more information than **WAIT**.

Syntax

Pascal

```
p_wait (stinfo);

p_wait3 (status, options, usage)
```

FORTRAN

```
FWAIT (STINFO)

FWAIT3 (STATUS, OPTIONS, USAGE)
```

Parameters

stinfo
is the termination status returned by one of the child processes to the parent process.

In Pascal, the termination status is of type integer.

In FORTRAN, the termination status is of type INTEGER.

status
is the termination status returned by one of the children of the calling process.

In Pascal, **status** is of type integer.

In FORTRAN, the **status** is of type INTEGER.

options
specifies either or (by logical ORing) both of two conditions of execution:

WNOHANG causes **WAIT3** not to delay if no processes are ready to report their status.

WUNTRACED causes **WAIT3** to return information when children of the calling process have stopped.

In Pascal, **options** is of type integer.

In FORTRAN, the **options** is of type INTEGER.

usage
describes the total resources used on all sites by the terminated process.

In Pascal, **usage** is of type rusageptr.

In FORTRAN, **rusage** is of type INTEGER USAGE(23).

Return Values
The process ID of a stopped or terminated child process is returned upon successful completion of the **WAIT** system call. The value 0 is returned upon successful completion of the **WAIT3** system call. The value -1 is returned and an error code set in **errno** if either call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **WAIT** system routine as well as two others that are commonly used in the context of a wait call: **FORK** and **EXECL**.

In both examples, the result is the creation of a new process that is a copy of the parent process. The **WAIT** call allows the inner loop of the child process to complete execution before the parent process proceeds further. Without the **WAIT** call, it is likely that the child process cannot complete the inner loop before the parent issues the **EXECL** call and prints the date. The **WAIT** call guarantees that the child process will complete the loop before the **EXECL** call is issued.

Pascal

```
procedure wait1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
var
  blue, green, orange, pink, purple, red, yellow : integer;

%include /usr/include/aildefs.inc

begin
  green := p_fork;
  if green = 0 then
     begin
       for orange := 1 to 40 do
         writeln ('child process');
       purple := p_exit (pink)
     end;
  blue := p_wait (red);
  writeln (blue);
  yellow := p_execl ('/bin/sh', 'sh', '-c', 'date', '')
end;
```

FORTRAN

```
      SUBROUTINE WAIT1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FWAIT, FEXECL, FEXIT, FFORK, BLUE, GREEN, ORANGE
      INTEGER PINK, PURPLE, RED, YELLOW
      GREEN = FFORK ()
      IF (GREEN .EQ. 0) THEN
         DO 10 ORANGE = 1,40
            PRINT *, 'CHILD PROCESS'

 10   CONTINUE
      PURPLE = FEXIT (PINK)
      ENDIF
      BLUE = FWAIT (RED)
      PRINT *, BLUE
      YELLOW = FEXECL ('/bin/sh ', 'sh ', '-c ', 'date ', ' ')
      END
```

2.113 WRITE, WRITEX write to a file

Description

The **WRITE** system call writes a specified number of bytes from a specified area to a specified file.

The **WRITEX** system call invokes additional communications facilities.

Syntax

Pascal

```
p_write (fildes, buffer, nbytes);

p_writex (fildes, buffer, nbytes, ext);
```

Pascal external function definitions

```
function p_write (fildes : integer; buffer : writptr;
                  nbytes : integer) : integer; external;

function p_writex (fildes : integer; buffer : writptr;
                 nbytes, ext : integer) : integer; external;
```

FORTRAN

```
FWRITE (FILDES, BUFFER, NBYTES)

FWRITEX (FILDES, BUFFER, NBYTES, EXT)
```

Parameters

fildes
is the descriptor of the file to be written to and is returned by a successful **CREAT**, **DUP**, **DUP2**, **FCNTL**, **OPEN**, **PIPE**, **SOCKET**, or **SOCKETPAIR** system call.

In Pascal, **fildes** is of type integer.

In FORTRAN, **fildes** is of type INTEGER.

buffer
is a pointer to a buffer of **nbytes** contiguous bytes that are written to the output file. The number of characters actually written is returned. It should be regarded as an error if the return value differs from the number requested.

In Pascal, **buffer** is a pointer of type writptr. (Writptr is a pointer to a user-defined packed array of type char.)

In FORTRAN, **buffer** is a user-defined array of type CHARACTER.

nbytes
is the number of bytes to be written to the specified file.

In Pascal, **nbytes** is of type integer.

In FORTRAN, **nbytes** is of type INTEGER.

ext
is a parameter of the **WRITEX** call. It provides a value or a pointer to a communications area for specific devices.

In Pascal, **ext** is of type integer.

In FORTRAN, **ext** is of type INTEGER.

In Pascal and FORTRAN, **ext** is device-dependent (see *AIX Technical Reference*).

Return Values
The return value is the number of bytes written to the specified file. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples
The Pascal procedure and FORTRAN subroutine that follow call the **WRITE** system routine, which writes a specified number of bytes to a file that has been opened for writing. In these examples, 35 bytes are written to the file /tmp/junk from the Pascal packed array "yellow" and from the FORTRAN array "YELLOW".

Pascal

```
procedure write1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
  writary = packed array[1..35] of char;
  writptr = @writary;
var
  blue, orange, red : integer;
  yellow : writptr;

function p_write (fildes : integer; buf : writptr;
                nbytes : integer) : integer; external;

begin
  new(yellow);
  yellow@ := 'test file for the WRITE system call';
  blue := p_open ('/tmp/junk', WRONLY, 0);
  red := 35;
  orange := p_write (blue, yellow, red);
  writeln (orange);
```

```
end;
```

FORTRAN

```
      SUBROUTINE WRITE1
      INCLUDE (/usr/include/ailfconsts.inc)
      INTEGER FWRITE, FOPEN, BLUE, ORANGE, RED
      CHARACTER*35 YELLOW
      BLUE = FOPEN ('/tmp/junk ', WRONLY, 0)
      YELLOW = 'test file for the WRITE system call '
      RED = 35
      ORANGE = FWRITE (BLUE, YELLOW, RED)
      PRINT *, ORANGE
      END
```

2.114 WRITEV write output from multiple buffers

Description

The **WRITEV** system call obtains data from a specified set of buffers and writes it to a specified object.

Syntax

Pascal

```
p_writev (d, iov, iovcnt);
```

Pascal external function definition

```
function p_writev (d : integer; var iov : iovarr;
                   iovcnt : integer) : integer; external;
```

FORTRAN

This system call is not available in FORTRAN.

Parameters

d
is a file descriptor or a socket descriptor.

In Pascal, **d** is of type integer.

In FORTRAN, **d** is of type INTEGER.

iov
is an array of buffers.

In Pascal, **iov** is an array of records of type iovrec (user-defined).

iovcnt
is the number of buffers of the type specified by **iov**

In Pascal, **iovcnt** is of type integer.

Return Values

The number of bytes written is returned upon successful completion of the call. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

Examples

In the Pascal procedure that follows, five iovec records are initialized with base addresses and a buffer length of 10. The buffers are filled with "123456789" strings. File descriptor "s" is created by an **OPEN** system call, and **WRITEV** is called to write information to file "s" from the five buffers pointed to by **iov**.

Pascal

```
procedure writev1;

const
  %include /usr/include/ailpconsts.inc
type
  %include /usr/include/ailtypes.inc
  buf = packed array[1..10] of char;
  bufptr = ^buf;
  iovrec = record
     iov_len : integer;
     iov_base : bufptr;
   end;
  iovarr = array[1..5] of iovrec;
var
  i, s, green : integer;
  arr : st5;
  iov : iovarr;

%include /usr/include/aildefs.inc

function p_writev (d : integer; var iov : iovarr;
                                 iovcnt : integer) : integer; external;

begin
  for i := 1 to 5 do
  begin
    iov[i].iov_len := 10;
    iov[i].iov_base^ := '123456789';
  end;
  s := p_open ('/tmp/junk', RDWR + CREATE, 0);
  green := p_writev (s, iov, 5);
  if (green <> -1) then
    writeln ('Writev returned: OK')
  else
    writeln ('Writev returned: ERROR')
  if (green = -1) then showerror;
  s := p_unlink ('/tmp/junk');
end;
```

A.0 Appendix A. Error Codes and Error Messages

This appendix describes the errors that can occur when a system call is issued. Some subroutines that invoke system calls indicate errors in a similar way.

System calls indicate the occurrence of an error by returning a special value. This value is almost always -1, but you should check the description of the particular system call to be sure. A number identifying the error condition is stored in an external variable called **errno** (see "Return Values, Error Codes, and Error Messages" in Chapter 1 for information on how to access *errno*). This variable is not cleared when a system call is successful, so its value is meaningful only after one error has occurred and before another.

The **errno.h** header file declares the **errno** variable and defines the name of each error condition.

For each error code the following list gives the code number, the symbolic name defined in the **errno.h**, header file, and the associated error message. (For additional information, see **perror** in *AIX Operating System Technical Reference*.)

```
 1  EPERM  Not the owner
 2  ENOENT  No such file or directory
 3  ESRCH  No such process
 4  EINTR  Interrupted system call
 5  EIO  I/O error
 6  ENXIO  No such device or address
 7  E2BIG  Argument list too long
 8  ENOEXEC  Exec format error
 9  EBADF  Bad file number
10  ECHILD  No child process
11  EAGAIN  No more processes
12  ENOMEM  Not enough space
13  EACCES  Permission denied
14  EFAULT  Bad address
15  ENOTBLK  Block device required
16  EBUSY  Mount device busy
17  EEXIST  File exists
18  EXDEV  Cross-device link
19  ENODEV  No such device
20  ENOTDIR  Not a directory
21  EISDIR  Is a directory
22  EINVAL  Invalid argument
23  ENFILE  File table overflow
24  EMFILE  Too many open files
25  ENOTTY  Not a typewriter
26  ETXTBSY  Text file busy
27  EFBIG  File too large
28  ENOSPC  No space left on device
29  ESPIPE  Illegal seek
30  EROFS  Read-only file system
31  EMLINK  Too many links
32  EPIPE  Broken pipe
33  EDOM  Math argument
34  ERANGE  Result too large
35  ENOMSG  No message of desired type
36  EIDRM  Identifier removed
37  ECHRNG  Channel number out of range
```

```
38  EL2NSYNC  Level 2 not synchronized
39  EL3HLT  Level 3 halted
40  EL3RST  Level 3 reset
41  ELNRNG  Link number out of range
42  EUNATCH  Protocol driver not attached
43  ENOCSI  No CSI structure available
44  EL2HLT  Level 2 halted
45  EDEADLK  Potential deadlock
```

B.0 Appendix B. Pascal Constants

The following definitions of constants are required for Pascal calling sequences.

ACCESS

```
F_OK = 0  { search for a file }
X_OK = 1  { test for execute permission }
W_OK = 2  { test for write permission }
R_OK = 4  { test for read permission }
```

CHOWNX

```
T_OWNER_AS_IS = 4
T_GROUP_AS_IS = 32
```

DISCLAIM

```
ZERO_MEM = 0
```

FCNTL

```
F_DUPFD    = 0
F_GETFD    = 1
F_SETFD    = 2
F_GETFL    = 3
F_SETFL    = 4
F_GETLK    = 5
F_SETLK    = 6
F_SETLKW   = 7
F_OPENLOCK = 8
F_GETOWN   = 9
F_SETOWN   = 10

F_RDLCK    = 1
F_WRLCK    = 2
F_UNLCK    = 3
```

FULLSTAT and **FFULLSTAT**

```
FLSTAT = 0
FLSTRV = 1
FLSTOT = 2

FS_VMP = 1
```

GETGRP

```
NGROUP = 26  { maximum number of group access entries allowed }
```

GETITIMER and **SETITIMER**

```
ITIMER_REAL    = 0
ITIMER_VIRTUAL = 1
ITIMER_PROF    = 2
```

GETSOCKOPT and **SETSOCKOPT**

```
SO_DEBUG       = 1
```

```
  SO_ACCEPTCONN  = 2
  SO_REUSEADDR   = 4
  SO_KEEPALIVE   = 8
  SO_DONTROUTE   = 16
  SO_BROADCAST   = 32
  SO_USELOOPBACK = 64
  SO_LINGER      = 128
  SO_OOBINLINE   = 256
  SOL_SOCKET     = 65530
```

IOCTL

```
  IOCTYP = 65280
  IOCINF = 65281

  { device types }

    DDLP   = 'l'  { line printer }
    DDTAPE = 'M'  { mag tape }
    DDTTY  = 'T'  { terminal }
    DDDISK = 'R'  { disk }
    DDRTC  = 'c'  { real-time (calendar) clock }
    DDPSEU = 'Z'  { pseudo-device }
    DDNET  = 'N'  { networks }
    DDEN   = 'E'  { Ethernet interface }
    DDEM78 = 'e'  { 3278/79 emulator }

  { tape-drive types }

    STREAM = 1  { streaming tape drive }
    STRSTP = 2  { start-stop tape drive }

  { flags }

    DFIXED = 01  { non-removable }
    DFRAND = 02  { random access possible }
    DFFAST = 04
```

LOCKF

```
  F_ULOCK = 0
  F_LOCK  = 1
  F_TLOCK = 2
  F_TEST  = 3
```

LSEEK

```
  SEEK_SET = 0
  SEEK_CUR = 1
  SEEK_END = 2
```

MOUNT

```
  MC_MOUNTS = 0
```

MOUNT and **UMOUNT**

```
  {flags}

    MNTRDO  = 1
```

```
    MNTRMB  = 2
    MNTDEV  = 4
    MNTREM  = 256

  { types }

    MNTAIX = 0
    MNTDS  = 1
```

MSGGET

```
  IXOTH  = 1     { other: execute, search permission }
  IWOTH  = 2     { other: write permission }
  IROTH  = 4     { other: read permission }
  IRWXO  = 7     { other: execute, read, write permission }
  IXGRP  = 8     { group: execute, search permission }
  IWGRP  = 16    { group: write permission }
  IRGRP  = 32    { group: read permission }
  IRWXG  = 56    { group: execute, read, write permission }
  IXUSR  = 64    { owner: execute, search permission }
  IWUSR  = 128   { owner: write permission }
  IRUSR  = 256   { owner: read permission }
  IRWXU  = 448   { owner: execute, read, write permission }
  IPCCRT = 512   { create entry if key doesn't exist }
  IPCEXL = 1024  { fail if key exists }
  IPCALC = 32768 { use if identifier exists }
  ENFMT  = ISGID { enables enforcement-mode record locking }
```

MSGRCV

```
  IPCNWT = 2048  { specify response to non-existent message;
                     also used in SEMOP as a sem_flg value }
  IPCNER = 4096  { truncate a message that is too long }
```

OPEN

```
  CREATE = 256  { open with file create; uses third OPEN arg }
  TTRUNC = 4096 { open with truncation }
  EXCL   = 8192 { exclusive open }
```

OPEN and **CREAT**

```
  RDONLY  = 0
  WRONLY  = 1
  RDWR    = 2
  NDELAY  = 4   { non-blocking I/O }
  APPEND  = 8   { append; writes guaranteed at the end) }
  DEFERC  = 32
```

OPEN, **CREAT**, **MKNOD**, AND **CHMOD**

```
  IEXEC  = 64     { owner: execute, search permission }
  IWRITE = 128    { owner: write permission }
  IREAD  = 256    { owner: read permission }

  ISVTX  = 512    { save text even after use }
  ISGID  = 1024   { set group id on execution }
  ISUID  = 2048   { set user id on execution }
  IFIFO  = 4096   { fifo }
  IFCHR  = 8192   { character special }
```

```
  IFDIR  = 16384  { directory }
  IFBLK  = 24576  { block special }
  IFREG  = 32768  { regular }
  IFMT   = 61440  { type of file }
  IFMPX  = IFCHR + ISVTX { multiplexed character-special file }
```

PLOCK

```
  UNLOCK   = 0
  PROCLOCK = 1
  TXTLOCK  = 2
  DATLOCK  = 4
```

REBOOT { these flags are defined in the file newconsts.inc }

```
  RBASKNAME = 1
  RBNOSYNC  = 4
  RBHALT    = 8
```

SEMCTL

```
  IPCRMD = 0
  IPCSET = 1
  IPCSTT = 2
  GTNCNT = 3
  GETPID = 4
  GETVAL = 5
  GETALL = 6
  GTZCNT = 7
  SETVAL = 8
  SETALL = 9
```

SEMOP

```
  SEMNDO = 4096
```

SENDTO, **SENDMSG**, **SENDFROM**, **RECV**, **RECVMSG**, and **RECVFROM**
{ these constants are defined in the file newconsts.inc)

```
  MSG_OOB       = 1
  MSG_PEEK      = 2
  MSG_DONTROUTE = 4
  MSG_MAXIOVLEN = 16
```

SHMAT

```
  SHMMAP = 2048
  SHMRDO = 4096
  SHMRND = 8192
  SHMCPY = 16384
  SHMLBA = 268435456
```

SIGNAL

```
  SIG_BLOCK   =  0
  SIG_UNBLOCK =  1
  SIG_SETMASK =  2

  SIGHUP     =    1     { hangup }
  SIGINT     =    2     { interrupt or rubout }
```

```
SIGQIT     =    3     { quit (ASCII FS) }
SIGILL     =    4     { illegal instruction (not reset when caught) }
SIGTRP     =    5     { trace trap, not reset when caught }
SIGIOT     =    6     { IOT instruction (abort) }

SIGEMT     =    7     { EMT instruction }
SIGFPE     =    8     { floating point exception }
SIGKIL     =    9     { kill (cannot be caught or ignored) }
SIGBUS     =   10     { bus error }
SIGSGV     =   11     { segmentation violation }
SIGSYS     =   12     { bad argument to system call }
SIGPIP     =   13     { write on a pipe with no one to read it }
SIGALM     =   14     { alarm clock }
SIGTRM     =   15     { software termination signal from kill }
SIGU1      =   16     { user-defined signal 1 }

SIGSTOP     =   17
SIGTSTP     =   18
SIGCONT     =   19
SIGCHLD     =   20
SIGTTIN     =   21
SIGTTOU     =   22
SIGIO       =   23
SIGXCPU     =   24
SIGXFSZ     =   25
SIGMSG      =   27

SIGWINCH    =   28
SIGPWR      =   29
SIGUSR1     =   30
SIGUSR2     =   31
SIGPROF     =   32
SIGDANGER   =   33
SIGVTALRM   =   34
SIGGRANT    =   60
SIGRETRACT  =   61
SIGSOUND    =   62

SIGDFL     =    0     { for signal code parameter default }
SIGIGN     =    1     { for signal code parameter ignore }
SIGADDR    =    2     { for sigvec code parameter handler address }
```

SOCKET

```
PF_UNIX = 1
PF_INET = 2

SOCK_STREAM = 1
SOCK_DGRAM  = 2
```

STATX and **FSTATX**

```
STX_LINK   = 1
STX_MOUNT  = 2
STX_HIDDEN = 4

STATSIZE = 100
```

USRINFO

```
GETINF = 1   { used as a parameter in the system call }
SETINF = 2   { used as a parameter in the system call }
INFSIZ = 64  { a constant equal to the size of the user buffer
                                              in the system call }
```

WAIT3 { these constants are defined in the file newconsts.inc }

```
WNOHANG   = 1
WUNTRACED = 2
```

C.0 Appendix C. Pascal Type Declarations

The following declarations of types are required for Pascal calling sequences.

```
cargv      = array[1..80] of cstrptr;
charinfsiz = packed array[1..64] of char;
charnine   = packed array[1..9] of char;
charptr    = @char;
char160    = packed array[1..160] of char;
char32     = packed array[1..32] of char;
char45     = packed array[1..45] of char;
cstring    = packed array[1..81] of char;
cstrptr    = @cstring;
cstr12     = packed array[1..13] of char;
intngroup  = packed array[1..26] of integer;
intptr     = @integer;
pasargv    = array[1..80] of st80;
piparray   = array[1..2] of integer;
short      = -32767..32767;
shrtptr    = @short;
st12       = string(12);
st12ptr    = @st12;
st512      = string(255);   {changed from 512 because of limit}
st512ptr   = @st512;
st80       = string(80);
st80ptr    = @st80;
ushrt      = -32767..32767;
usign      = integer;
```

FULLSTAT and **FFULLSTAT**

```
vset    = (VDIR, VCHR, VBLK, VREG, VMPC, VFIFO, VBAD, VUNDEF );
tagset  = (CALLER, OTHER, SOMONE, NOONE);
vtype   = VDIR..VUNDEF;
tagtype = CALLER..NOONE;

fullstatrec = record
              st_dev          : integer;
              st_ino          : integer;
              st_mode         : integer;
              st_nlink        : ushrt;
              spare0          : ushrt;
              st_uid          : integer;
              st_gid          : integer;
              st_rdev         : integer;
              st_size         : integer;
              st_atime        : integer;
              spare1          : integer;
              st_mtime        : integer;
              spare2          : integer;
              st_ctime        : integer;

              st_spare3       : integer;
              st_blksize      : integer;
              st_blocks       : integer;
              fst_i_gen       : integer;
              fst_vfs         : integer;
              fst_flag        : integer;
              st_cmtcnt       : integer;
```

```
              st_fstore       : integer;
              st_version      : integer;
              st_css          : short;
              st_ss           : short;
              st_rdevsite     : short;
              st_spare4       : short;
              fst_nid         : integer;
              fst_uid_raw     : usign;
              fst_gid_raw     : usign;
              fst_uid_rev_tag : usign;
              fst_gid_rev_tag : usign;
            end;
  fullstatptr = @fullstatrec;
  fullstatarr = array[1..30] of integer;
```

Message routines

```
  msgxbuf = record
              mtime : integer;
              muid  : short;
              mgid  : short;
              mnid  : integer;
              mpid  : short;
              mtype : integer;
              mtext : st80;
            end;
  msgxptr = @msgxbuf;

  msg = record
          next   : msgptr;
          mattr  : msgxbuf;
          mtxtsz : short;
          mloc   : short;
        end;
  msgptr = @msg;
  msgary = array[1..100] of msg;

  msqid_ds = record
               msg_perm   : perm;
               msg_first  : msgptr;
               msg_last   : msgptr;
               msg_cbytes : ushrt;
               msg_qnum   : ushrt;
               msg_qbytes : ushrt;
               msg_lspid  : integer;
               msg_lrpid  : integer;
               msg_stime  : integer;
               msg_rtime  : integer;
               msg_ctime  : integer;
             end;
  mdsptr = @msqid_ds;

  msgbuf = record
             mtype : integer;
             mtext : st80;
           end;
  mbufptr = @msgbuf;

  perm = record
           uid  : short;
```

```
         gid  : short;
         cuid : short;
         cgid : short;
         mode : short;
         seq  : short;
         key  : integer;
       end;
```

Semaphore routines

```
  sem = record
        semval  : short;
        sempid  : short;
        semncnt : short;
        semzcnt : short;
      end;

  semptr = @sem;

  semid_ds = record
             sem_perm  : perm;
             sem_base  : semptr;
             sem_nsems : short;
             semlcnt   : short;
             sem_otime : integer;
             sem_ctime : integer;
           end;
  semidptr = @semid_ds;

  semary    = array [1..1000] of short;
  semaryptr = @semary;
  abc = 0..2;
  semrec = record
      case abc of
           0 : (val  : integer);
           1 : (buf  : semidptr);
           2 : (arry : semaryptr);
         end;

  sembuf = record
           sem_num : short;
           sem_op  : short;
           sem_flg : short;
         end;
  semopary = array[1..1000] of sembuf;
```

Shared-memory routines

```
  smds = record
         shperm     : perm;
         shsegsz    : integer;
         shlpid     : integer;
         shcpid     : integer;
         shnattach  : short;
         shcnattach : short;
         shatime    : integer;
         shdtime    : integer;
         shctime    : integer;
         spare0     : integer;
```

```
     end;
  smdsptr = @smds;
```

Signal routines

```
  signalstack = record
                 ss_sp : cstrptr;
                 ss_onstack : integer;
               end;
  stackptr = @signalstack;

  signalvec = record
               sv_handler : intptr;
               sv_mask    : integer;
               sv_onstack : integer;
             end;
  sigvecptr = @signalvec;
```

New signal calls

```
  nsigtype = array[1..3] of usign;

  sigset_t = record
              setsize : integer;
              sigs    : nsigtype;
            end;
  sigset_tptr = @sigset_t;

  sigact = record
            sa_mask    : sigset_t;
            sa_flags   : integer;
            sa_handler : integer;
          end;
  sigactptr = ¬sigact;

  flock = record
           l_type   : short;
           l_whence : short;
           l_start  : integer;
           l_len    : integer;
           l_sysid  : usign;
           l_pid    : short;
         end;

  flockptr = @flock;
```

Calls to SIGVEC

The following definitions are used strictly with a call to **SIGVEC** to restore the process previous execution context, information pushed on the stack when a signal is delivered. This is used by the kernel to restore state following execution of the signal handler. It is also made available to the handler to allow it to properly restore state if a non-standard exit is performed.

```
  FP_STATUS = integer;       { holds the following information:
                               bit   1 : SIGFPE on exception
                               bit   2 : exception occurred
                               bit   3 : invalid operation occurred
```

```
                              bit   4 : exception on invalid operation
                              bit   5 : divide by zero occurred
                              bit   6 : exception on divide by zero
                              bit   7 : overflow occurred
                              bit   8 : exception on overflow
                              bit   9 : underflow occurred
                              bit  10 : exception on underflow
                              bits 11-21 : reserved
                              bits 22&23 : comparison result
                              bits 24&25 : rounding mode
                              bit  26    : inexact result occurred
                              bit  27    : exception on inexact result
                              bits 28&29 : reserved
                              bits 30-32 : machine communications type}

choice = 0..2;
fpreg = record
   case choice of
        0 : (hp : usign;
             lp : usign);
        2 : (freg : array[1..2] of real);
      end;

fptrapinfo = integer;

fptrap = record
         info              : fptrapinfo;
         designated_result : fpreg;
       end;

fpvmach = record
          fpregarray : array[1..8] of fpreg;
          statusreg  : FP_STATUS ;
          fptrapvar  : fptrap;
        end;

sigcontext = record
             sc_onstack : integer;   { Sigstack state to restore     }
             sc_mask    : integer;   { Signal mask to restore        }
             sc_sp      : integer;   { sp  to restore (ignored)      }
             sc_pc      : integer;   { pc to restore                 }
             sc_ps      : integer;   { psl to restore (ignored)      }
             fpvmp      : @fpvmach;  { pointer to virtual fp machine }
          end;
contextptr = @sigcontext;

char14 = array[1..14] of char;
int4   = array[1..4] of integer;

prof = record
       p_low     : integer;
       p_high    : integer;
       p_buff    : shrtptr;
       p_bufsize : integer;
       p_scale   : integer;
     end;

timeval= record
         tv_sec  : integer;
         tv_usec : integer;
```

```
       end;
timevalptr = @timeval;
timeval2   = array[1..2] of timeval;

timezone = record
            tz_minuteswest : integer;
            tz_dsttime     : integer;
          end;
timezoneptr = @timezone;

itimerval = record
             it_interval : timeval;
             it_value    : timeval;
           end;
itimervalptr = @itimerval;

rusage   = record
            ru_utime    : timeval;
            ru_stime    : timeval;
            ru_maxrss   : integer;
            ru_ixrss    : integer;
            ru_idrss    : integer;
            ru_isrss    : integer;
            ru_mainflt  : integer;
            ru_majflt   : integer;
            ru_nswap    : integer;
            ru_inblock  : integer;
            ru_outblock : integer;
            ru_msgsnd   : integer;
            ru_msgrcv   : integer;
            ru_nsignals : integer;
            ru_nvcsw    : integer;
            ru_cw       : integer;

            ru_steal    : integer;
            ru_swap     : integer;
            ru_file     : integer;
            ru_demand   : integer;
          end;
rusageptr = @rusage;

iovec = record
         iov_base : charptr;
         iov_len  : integer;
       end;
iovecptr = @iovec;

msghdr = record
          msg_name          : cstring;
          msg_namelen       : integer;
          msg_iov           : iovecptr;
          msg_iovlen        : integer;
          msg_accrights     : cstring;
          msg_accrightslen : integer;
        end;
msghdrptr = @msghdr;
int2 = array[1..2] of integer;

sockaddr = record
            sa_family : ushrt;
```

```
           sa_data   : packed array[1..14] of char;
         end;
sockaddrptr = @sockaddr;

mminfo = record
         m_nid    : usign;
         m_object : cstring;
         m_stub   : cstring;
         m_flag   : usign;
         m_date   : short;
       end;
minfoptr = @mminfo;

bheader = record
          nid      : integer;
          reserved : integer;
          size     : usign;
          minfo    : mminfo;
        end;
bheaderptr = @bheader;

ltable = record
         ttype    : char;
         id       : integer;
         mode     : char;
         nid      : integer;
         reserved : array[1..4] of integer;
       end;
ltableptr = @ltable;

idrow = record
        wireid  : integer;
        localid : short;
        pad     : short;
      end;

dsxlate = record
          rlv1    : short;
          gid     : short;
          uid     : short;
          flag    : char;
          pad1    : char;
          numuids : ushrt;
          numgids : ushrt;
          pad2    : short;
          idrow1  : idrow;
        end;

 ds_state = record
            i_state  : short;  { input state }
            i_kprocs : short;  { input number of kprocs }
            r_state  : short;  { result state }
            r_kprocs : short;  { result number of kprocs }
            reserved : array[1..4] of integer;   { reserved }
          end;
dsstateptr = @ds_state;

ddsipc = record
         inkey  : integer;
         nid    : integer;
```

```
          outkey : integer;
        end;
```

STAT and **FSTAT**

```
  statrec = record
            st_dev     : integer;
            st_ino     : integer;
            st_mode    : integer;
            st_nlink   : ushrt;
            spare0     : ushrt;
            st_uid     : integer;
            st_gid     : integer;
            st_rdev    : integer;
            st_size    : integer;
            st_atime   : integer;
            spare1     : integer;
            st_mtime   : integer;
            spare2     : integer;
            st_ctime   : integer;
            st_spare3  : integer;
            st_blksize : integer;
            st_blocks  : integer;
            st_gen     : integer;
            st_type    : integer;
        end;
  statptr = @statrec;
```

Time routines

```
  tms = record
        tms_utime  : integer;
        tms_stime  : integer;
        tms_cutime : integer;
        tms_cstime : integer;
      end;
```

UNAME

```
  unam = record
         sysname  : char32;
         nodename : char32;
         release  : char32;
         version  : char32;
         machine  : char32;
       end;
  unptr = @unam;

  xunam = record
          nid      : usign;
          reserved : array[1..3] of integer;
        end;
  xunptr = @xunam;
```

USTAT

```
  ustatrec = record
             f_tfree  : integer;
             f_tinode : usign;
             f_fname  : array[1..6] of char;
```

```
            f_fpack  : array[1..6] of char;
          end;
  ustatptr = @ustatrec;
  devkind = (disk, map, ether, mag);

  devinfo = record
            devtyp_flg : packed array[1..2] of char;
                                        { devinfo and flags chars needed... }
                                        { ...for proper allignment          }

    hold : short;
        case devkind of
        disk : (bytpsec : short;                  { bytes per sector }
                secptrk : short;                  { sectors per track }
                trkpcyl : short;                  { tracks per cylinder }
                numblks : integer);               { blocks this partition }
        map  : (capab : char;                     { capabilities }
                mode  : char;                     { current mode }
                hres  : short;                    { horizontal resolution }
                vres  : short);                   { vertical resolution }
        ether: (capabs : short;                   { capabilities }
                haddr  : array[1..6] of char);    { hardware address }
        mag  : (typ : short)                      { what flavor of tape }
    end;
  devptr = @devinfo;
```

UTIME

```
  utimbuf = record
            actime  : integer;
            modtime : integer;
          end;
  utimptr = @utimbuf;
```

D.0 Appendix D. Pascal Procedure and Function Declarations

The following declarations are required for Pascal calling sequences.

```
  function p_accept (s : integer; addr : sockaddrptr; var addrlen : integer) :
  function p_access (path : st80; amode : integer) : integer; external;
  function p_acct (path : st80) : integer; external;
  function p_adjtime (var delta, olddelta : timeval) : integer; external;
  function p_alarm (sec : usign) : usign; external;

  function p_bind (s : integer; name : sockaddrptr; namelen : integer) : integ
  function p_brk (endds : integer) : integer; external;

  function p_chdir (path : st80) : integer; external;
  function p_chhidden (path : st80; flag : integer) : integer; external;
  function p_chmod (path : st80; mode : integer) : integer; external;
  function p_chown (path : st80; owner, group : integer) : integer; external;
  function p_chownx (path : st80; owner, group, tflag : integer) : integer; ex
  function p_chroot (path : st80) : integer; external;
  function p_close (fildes : integer) : integer; external;
  function p_connect (s : integer; name : sockaddrptr; namelen : integer) : in
  function p_creat (path : st80; mode : integer) : integer; external;

  function p_dup (fildes : integer) : integer; external;
  function p_dup2 (oldfd, newfd : integer) : integer; external;

  function p_ercode : integer; external;
  function p_execl (path, arg0, arg1, arg2, arg3 : st80) : integer; external;
  function p_execle (path, arg0, arg1, arg2, arg3 : st80; envp : pasargv) : in
  function p_execlp (filename, arg0, arg1, arg2, arg3 : st80) : integer; exter
  function p_execv (path : st80; args : pasargv) : integer; external;
  function p_execve (path : st80; args, envp : pasargv) : integer; external;
  function p_execvp (filenm : st80; args : pasargv) : integer; external;
  function p_exit (status : integer) : integer; external;
  function p__exit (status : integer) : integer; external;

  function p_fabort (fildes : integer) : integer; external;
  function p_fclear (fildes : integer; nbytes : usign) : usign; external;
  function p_fcommit (fildes : integer) : integer; external;
  function p_ffullstat (fildes, cmd : integer; var buf : fullstatrec) : intege
  function p_fork : integer; external;
  function p_fstat (fildes : integer; var buf : statrec) : integer; external;
  function p_fstatx (fildes : integer; var buf : statrec; len, cmd : integer)
  function p_fsync (fildes : integer) : integer; external;
  function p_ftok (path : st80; id : char) : integer; external;
  function p_ftruncate (fildes : integer; len : usign) : integer; external;
  function p_fullstat (path : st80; cmd : integer; var buf : fullstatrec) : in

  function p_getdtablesize : integer; external;
  function p_getegid : ushrt; external;
  function p_geteuid : ushrt; external;
  function p_getgid : ushrt; external;
  function p_getgroups (ngrp :integer; var gidset : intgroup) : integer; exter
  function p_gethostid : integer; external;
  function p_gethostname (var name : st80; namelen : integer) : integer; exter
  function p_getitimer (which : integer; var vvalue : itimerval) : integer; ex
  function p_getlocal (var localname : st80; maxlength : integer) : integer; e
  function p_getpeername (s : integer; name : sockaddrptr; var namelen : integ
  function p_getpgrp : integer; external;
  function p_getpid : integer; external;
```

```
function p_getppid : integer; external;
function p_getsockname (s : integer; name : sockaddrptr; var namelen : integ
function p_gettimeofday (var tp : timevalptr; var tzp : timezone) : integer;
function p_getuid : ushrt; external;
function p_getxvers (var xvers : st80; length : integer) : integer; external

function p_ioctl (fildes, request : integer; argp : devptr) : integer; exter

function p_kill (pid, sig : integer) : integer; external;
function p_killpg (pgrp, sig : integer) : integer; external;

function p_link (path1, path2 : st80) : integer; external;
function p_listen (s, backlog : integer) : integer; external;
function p_loadtbl (cntl : ltableptr; buf : st80; size : integer) : integer;
function p_lockf (fildes, request, size : integer) : integer; external;
function p_lseek (fildes, offset, whence : integer) : integer; external;
function p_lstat (path : st80; var buf : statrec) : integer; external;

function p_mkdir (var path : st80; mode : integer) : integer; external;
function p_mknod (path : st80; mode, dev : integer) : integer; external;
function p_mount (dev, dir : st80; rwflag : integer) : integer; external;
function p_msgctl (msqid, cmd : integer; buf : mdsptr) : integer; external;
function p_msgget (key, msgflg : integer) : integer; external;
function p_msgrcv (msqid : integer; msgp : mbufptr; msgsz, msgtyp, msgflg :
function p_msgsnd (msqid : integer; msgp : mbufptr; msgsz, msgflg : integer)
function p_msgxrcv (msqid : integer; msgpt : msgxptr; msgsz, msgtyp, msgflg

function p_nice (incr : integer) : integer; external;

function p_open (oath : st80; oflag, mode : integer) : integer; external;

function p_pause : integer; external;
procedure p_perror (a : st80); external;
function p_pipe (var fildes : piparray) : integer; external;
function p_plock (op : integer) : integer; external;
function p_profil (var buff : intptr; bufsiz, offset, scale : usign) : integ
function p_ptrace (request, pid : integer; addr : intptr; data : integer; bu


function p_readlink (path : st80; var buf : st80; bufsiz : integer) : intege
function p_reboot (dev : integer) : integer; external;
function p_recvmsg (s : integer; msg : msghdrptr; flags : integer) : integer
function p_rename (var frompath, topath : st80) : integer; external;
function p_rmdir (var path : st80) : integer; external;

function p_sbrk (incr : integer) : integer; external;
function p_select (nfds : integer; var read, write, except : integer; timeou

function p_semctl (semid, semnum, cmd : integer; var arg : semrec) : integer
function p_semget (key, nsems, semflg : integer) : integer; external;
function p_semop (semid : integer; var sops : semopary; nsops : integer) : i
function p_sendmsg (s : integer; msg : msghdrptr; flags : integer) : integer
function p_setgid (uid : integer) : integer; external;
function p_setgroups (ngrp :integer; var gidset : intgroup) : integer; exter
function p_sethostid (hostid : integer) : integer; external;
function p_sethostname (var name : st80; namelen : integer) : integer; exter
function p_setitimer (which : integer; var vvalue, ovalue : itimerval) : int
function p_setlocal (var localname : st80) : integer; external;
function p_setpgid (pid : integer; pgid : integer) : integer; external;
function p_setpgrp (flag : integer) : integer; external;
```

```
function p_settimeofday (var tp : timeval; var tzp : timezone) : integer; ex
function p_setuid (uid : integer) : integer; external;
function p_setxvers (xvers : st80) : integer; external;
function p_shmat (shmid, shmadr, shmflg : integer) : integer; external;
function p_shmctl (shmid, cmd : integer; buf : smds) : integer; external;
function p_shmdt (shmadr : integer) : integer; external;
function p_shmget (key, size, shmflg : integer) : integer; external;
function p_shutdown (s, how : integer) : integer; external;
function p_sigaction (sig : integer; act,oact : sigactptr) : integer; extern
function p_sigblock (mask : integer) : integer; external;
function p_signal (sig : integer; func : integer) : integer; external;
function p_sigpause (sigmsk : integer) : integer; external;
function p_sigprocmask (how : integer; var sset, oset : sigset_t) : integer;
function p_sigsetmask (mask : integer) : integer; external;
function p_sigstack (instack, outstack : stackptr) : integer; external;
function p_sigsuspend (sigmask : sigset_t) : integer; external;
function p_sigvec (sig, code : integer; invec,outvec : sigvecptr) : integer;
function p_socket (domain, ttype, protocol : integer) : integer; external;
function p_socketpair (domain,  ttype, protocol : integer; var sv : int2) :
function p_stat (path : st80; var buf : statrec) : integer; external;
function p_statx (path : st80; var buf : statrec; len,cmd : integer) : integ
function p_stime (tp : integer) : integer; external;
function p_symlink (path1, path2 : st80) : integer; external;
function p_sync : integer; external;
function p_system (str : st80) : integer; external;

function p_time (var tloc : integer) : integer; external;
function p_times (var buf : tms) : integer; external;

function p_ulimit (cmd, newlimit : integer) : integer; external;
function p_umask (cmask : integer) : integer; external;
function p_umount (dev : st80; flag : integer) : integer; external;
function p_uname (var name : unam) : integer; external;
function p_unamex (xname : xunam) : integer; external;
function p_unlink (path : st80) : integer; external;
function p_usrinfo (cmd : integer; var buf : charinfsiz; count : integer) :
function p_ustat (dev : integer; var buf : ustatrec) : integer; external;
function p_utime (path : st80; times : utimptr) : integer; external;
function p_utimes (ffile : st80; tvp : timeval2) : integer; external;

function p_wait (status : integer) : integer; external;
function p_wait3 (var status : integer; options : integer; usage : rusageptr
```

E.0 Appendix E. The ftok System Subroutine

Description

The **ftok** system subroutine returns a *key* that can be used to obtain interprocess-communication identifiers.

Syntax

```
+--- Pascal ---------------------------------------------------------+
¦                                                                    ¦
¦   p_ftok (path, id);                                               ¦
¦                                                                    ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

```
+--- FORTRAN --------------------------------------------------------+
¦                                                                    ¦
¦   FFTOK (PATH, ID)                                                 ¦
¦                                                                    ¦
¦                                                                    ¦
¦                                                                    ¦
+--------------------------------------------------------------------+
```

Parameters

path
is the path name of an existing file that can be accessed by the calling process.

In Pascal, **path** is of type st80.

In FORTRAN, **path** is a string or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

id
is a character that uniquely identifies a project.

In Pascal, **id** is of type char.

In FORTRAN, **id** is of type CHARACTER.

Return Values

A *key* is returned upon successful completion of a call to **ftok**. The value -1 is returned and an error code set in **errno** if the call fails.

In Pascal, the return value is of type integer

In FORTRAN, the return value is of type INTEGER

Examples

The Pascal procedure and FORTRAN subroutine shown on the next page issue a call to the **ftok** system subroutine, which returns a key associated with the file /tmp/sample.

Pascal

```
  procedure ftok1;
```

```
type
  %include /usr/include/ailtypes.inc
var
  red : integer;
  blue : st80;
  green : char;

%include /usr/include/aildefs.inc

begin
  green := 'z';
  blue := '/tmp/sample';
  red := p_ftok (blue, green);
  writeln (red)
end;
```

FORTRAN

```
      SUBROUTINE FTOK1
      INTEGER RED
      CHARACTER*80 BLUE , GREEN
      GREEN = 'z'
      BLUE = '/tmp/sample '
      RED = FFTOK (BLUE, GREEN)
      PRINT *, RED
      END
```

F.0 Appendix F. The perror System Subroutine

Description

The **perror** system subroutine writes a message explaining a system-call error.

Syntax

```
+--- Pascal -----------------------------------------------------------+
¦                                                                      ¦
¦   p_perror (pmsg);                                                   ¦
¦                                                                      ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

```
+--- FORTRAN ----------------------------------------------------------+
¦                                                                      ¦
¦   FPERROR (PMSG)                                                     ¦
¦                                                                      ¦
¦                                                                      ¦
¦                                                                      ¦
+----------------------------------------------------------------------+
```

Parameters

pmsg
is a user-defined message that precedes the standard error message.

In Pascal, **pmsg** is of type st80.

In FORTRAN, **pmsg** is a string variable or constant of type CHARACTER*80. The terminating character of the string must be a blank space.

Return Values

There is no return value from a successful **perror** call.

Examples

The Pascal procedure and FORTRAN subroutine shown on the next page print an error code number and the associated error message if the **path** parameter (in the **CHDIR** call) specifies a nonexistent directory.

Pascal

```
%include /usr/include/aildefs.inc
procedure showerror;

var
  result, code : integer;
  pmsg : st80;

begin
  pmsg = 'MEANING OF ERROR';
  result := p_chdir ('/usr/nonexist');
    if result = -1 then
      begin
        code := p_ercode;
        writeln (code);
```

```
        p_perror (pmsg)
      end
end;
```

FORTRAN

```
      SUBROUTINE ERRORS
      INTEGER RESULT, CODE, ERCODE
      RESULT = CHDIR ('/usr/nonexist ')
      IF (RESULT .EQ. -1) THEN
      CODE = ERCODE ()
      PRINT *, CODE
      CALL FPERROR
      ENDIF
      END
```

E

F