OS/390

Using REXX and OS/390 OpenEdition

SC28-1905-02

OS/390

Using REXX and OS/390 OpenEdition

SC28-1905-02

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xi.

| **Third Edition (March 1998)**

| This is a major revision of SC28-1905-01.

| This edition applies to Version 2 Release 5 of OS/390 (5647-A01) and to all subsequent releases and modifications until otherwise
| indicated in new editions or technical newsletters.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):
Your International Access Code +1+914+432-9405

IBMLink (United States customers only): KGNVMC(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet e-mail: mhvrcfs@vnet.ibm.com
World Wide Web: http://www.s390.ibm.com/os390

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Tables

Notices

Programming Interface

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain services of OS/390 OpenEdition.

Trademarks

ANSI	American National Standards Institute
DFS	Transarc Corporation
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
Network File System	Sun Microsystems, Inc.
NFS	Sun Microsystems, Inc.
Notes	Lotus Development Corporation
POSIX	Institute of Electrical and Electronics Engineers

About This Book

This book presents the information you need to write REXX programs to access OS/390 OpenEdition services on an IBM system with OS/390 OpenEdition.

This book describes the features and usage requirements for the OS/390 OpenEdition REXX extensions, or *syscall commands*, which are interfaces between the OS/390 operating system and the functions specified in the POSIX.1 standard (ISO/IEC 9945-1:1990[E] IEEE Std 1003.1-1990: First edition 1990-12-07; Information technology—Portable Operating System Interface [POSIX] Part 1; System Application Program Interface [API] [C Language]). These functions are used by OS/390 OpenEdition. This book also describes syscall commands that are not related to the standards.

Who Should Use This Book

This book is for programmers who are already familiar with the REXX language and experienced with the workings of TSO/E and OS/390 OpenEdition. It describes how to include in a REXX program syscall commands that access OS/390 OpenEdition services.

Where to Find More Information

Where necessary, this book references information in other books about the elements and features of OS/390. For complete titles and order numbers for all OS/390 books, see *OS/390 Information Roadmap*.

Direct your request for copies of any IBM publication to your IBM representative or to the IBM branch office serving your locality.

There is also a toll-free customer support number (1-800-879-2755) available Monday through Friday from 6:30 a.m. through 5:00 p.m. Mountain Time. You can use this number to:

- Order or inquire about IBM publications
- Resolve any software manufacturing or delivery concerns
- Activate the program reorder form to provide faster and more convenient ordering of software updates

Softcopy Publications

The OpenEdition library is available on the *OS/390 Collection Kit*, SK2T-6700. This softcopy collection contains a set of OS/390 and related unlicensed product books. The CD-ROM collection includes the IBM Library Reader, a program that enables customers to read the softcopy books.

Softcopy OS/390 publications are also available for web-browsing at this URL:

`http://www.s390.ibm.com/os390/license.html`

IBM Systems Center Publications

IBM systems centers produce redbooks that can be helpful in setting up and using OpenEdition services. You can order these publications through normal channels, or you can view them with a web browser from this URL:

`http://www.redbooks.ibm.com`

These books have not been subjected to any formal review nor have they been checked for technical accuracy, but they represent current product understanding (at the time of their publication) and provide valuable information on a wide range of OpenEdition topics. You must order them separately. A selected list of these books follows:

- *Selecting a Server — The Value of S/390,* SG24-4812
- *OS/390 TCP/IP OpenEdition Implementation Guide,* SG24-2141. Written for OS/390 TCP/IP OpenEdition, a replacement for TCP/IP for MVS Version 3 Release 2 Application Feature.
- *Accessing OS/390 OpenEdition MVS from the Internet,* SG24-4721. Written for TCP/IP for MVS Version 3 Release 2 Application Feature.
- *MVS/ESA SP 5.2.2 OpenEdition MVS Installation and Customization Starter Kit,* SG24-4529
- *Porting Applications to the OpenEdition MVS Platform,* GG24–4473. This book was written for the OpenEdition MVS feature of MVS/ESA SP 5.1.

OpenEdition Courses

The following classroom course is available:

- *UNIX System Services for OS/390 Implementation,* ESP25

The availability of educational offerings changes. For current information on classroom courses and other offerings, see your IBM representative or call 1-800-IBM-TEACH (1-800-426-8322).

OpenEdition Home Page

The OpenEdition home page on the World Wide Web has the latest technical news, customer stories, tools, and FAQs (frequently asked questions). You can visit it at this URL:

`http://www.s390.ibm.com/unix/`

or

`http://www.s390.ibm.com/oe/`

Some of the tools available from the web site are ported tools, and some are home-grown tools designed for OpenEdition. All this code works in our environment at the time we make it available, but is not officially supported. Each tool has a README file that describes the tool and any restrictions on its use.

The simplest way to reach these tools is through the OpenEdition home page. From the home page, click on the **Tools and Toys** icon.

The code is also available from **www.s390.ibm.com** through **anonymous ftp**.To get access:

1. Log in as user **anonymous**.
2. Change the directory (**cd**) to **os390/oe/port** or **os390/oe/toys** to find the subdirectories that contain the tools.

Restrictions

Because the tools are not officially supported,

- There are no guaranteed enhancements.
- No APARs can be accepted.

IBM Talklink Conferencing Service

The OpenEdition forum, OPENMVS CFORUM, is available to customers through IBM's TalkLink offering. Customers and IBM developers participate in the discussion in this forum.

To preview or sign up for TalkLink, access this URL using a secure web browser:

```
http://www.ibmlink.ibm.com/talklink
```

Web Access:

1. Go to **http://www.ibmlink.ibm.com/talklink.**
2. Click on Logon.
3. After logon, select Martlink.
4. Choose S390 from the menu.
5. Choose MVS from the menu.
6. Choose MVSSYSTEM from the menu.
7. Choose MVSFORUMS from the menu.
8. Choose OPENMVS from the menu.

SNA Fastpath Access:

1. Logon to IBMLink.
2. From the IBMLink main menu, type Talklink on the command line.
3. Type S390 on the command line.
4. Type OPENMVS on the command line.

Contact your IBM representative for information on TalkLink, DialIBM, or equivalent offerings for your country.

Discussion List

Customers and IBM participants also discuss OpenEdition on the **mvs-oe discussion list**. This list is not operated or sponsored by IBM; it is run by Georgetown University.

To subscribe to the mvs-oe discussion so you can receive postings, send a note to:

```
listserv@listserv.georgetown.edu
```

Include the following line in the body of the note, substituting your first name and last name as indicated:

```
subscribe mvs-oe first_name last_name
```

After you are subscribed, you will receive further instructions on how to use the mailing list.

Finding More Information about REXX

The following publication is useful: *The REXX Language: A Practical Approach to Programming*, by Michael Cowlishaw (Englewood Cliffs, NJ: Prentice-Hall, 1990).

Summary of Changes

| **Summary of Changes**
| **for SC28-1905-02**
| **OS/390 Version 2 Release 5**

| This book contains information previously presented in *Using REXX and
| OS/390 OpenEdition* SC28-1905-01, which supports OS/390 OpenEdition with
| OS/390 Version 2 Release 4.

| This book includes terminology, maintenance, and editorial changes.
| Technical changes or additions to the text and illustrations are indicated by a
| vertical line to the left of the change.

Chapter 1. Using TSO/E REXX for OS/390 OpenEdition Processing

The set of OS/390 OpenEdition extensions to the TSO/E Restructured Extended Executor (REXX) language enable REXX programs to access OS/390 OpenEdition callable services. The OS/390 OpenEdition extensions, called *syscall commands*, have names that correspond to the names of the callable services that they invoke—for example, **access**, **chmod**, and **chown**.

You can run a REXX program with syscall commands only on a system with OS/390 OpenEdition installed.

A REXX program is recognized by the word *REXX* (not case-sensitive) as the first word on the first line and within a REXX comment. For example, the following is a simple REXX program:

```
/* rexx */
say 'hello world'
```

You can run an interpreted or compiled REXX program with syscall commands from TSO/E, from MVS batch, from the shell, or from a program.

Host Command Environments for OS/390 OpenEdition Processing

Host command environments and external function packages that are available in the MVS REXX environment can be used by a REXX program that has OS/390 OpenEdition extensions. Two additional host command environments are also available:

- **SYSCALL:** For a REXX program with syscall commands that will be run from TSO/E or MVS batch, you need to initialize the environment by beginning a REXX program with a **syscalls('ON')** call.
- **SH:** For a REXX program with syscall commands that will be run from the shell or from a program, SH is the initial host environment. The SYSCALL environment is automatically initialized as well, so you do not need to begin the REXX program with a **syscalls('ON')** call. Syscall commands within the REXX program (for example, **chmod**) are interpreted as shell commands, not as syscall commands.

When a REXX program is run from the shell or from a program, both the SH and SYSCALL host command environments are available to it. When a REXX program is run from TSO/E or MVS batch, only the SYSCALL environment is available.

For background information on the concept of a host command environment, see *OS/390 TSO/E User's Guide*.

The SYSCALL Environment

The SYSCALL environment can be used by any REXX program with syscall commands, whether it runs from TSO/E or the shell (where the environment is automatically initialized).

Running a REXX Program from TSO/E or Batch

To run a REXX program with syscall commands from TSO/E or MVS batch, use the **syscalls('ON')** function at the beginning of the REXX program. This function:

- Ensures that the SYSCALL command environment (ADDRESS syscall) is established.
- Ensures that the address space is an OS/390 OpenEdition process; this is known as *dubbing*.
- Initializes the predefined OS/390 OpenEdition variables in the current REXX variable pool.
- Sets the signal process mask to block all signals that can be blocked. See “Using the REXX Signal Services” on page 8 for more information on signals.
- Clears the __*argv.* and __*environment.* stems. For this reason, it is not recommended that you use **syscalls('ON')** in a shell environment.

For REXX programs run from TSO/E or MVS batch, you use the **syscalls()** function to control the SYSCALL host command environment. You control the SYSCALL environment by using:

- **syscalls('ON')** to establish the SYSCALL environment
- **syscalls('OFF')** to end the SYSCALL environment
- **syscalls('SIGON')** to establish the signal interface routine
- **syscalls('SIGOFF')** to delete the signal interface routine

Note: The words **ON**, **OFF**, **SIGON**, and **SIGOFF** must be in uppercase letters.

Establishing the SYSCALL Environment

The **syscalls('ON')** function establishes the SYSCALL environment. It sets up the REXX predefined variables and blocks all signals. The function sets this return value:

0	Successful completion.
4	The signal process mask was not set.
7	The process was dubbed, but the SYSCALL environment was not established.
8	The process could not be dubbed.

The following example shows how you can use the **syscalls('ON')** function at the beginning of a REXX program:

```
if syscalls('ON')>3 then
   do
   say 'Unable to establish the SYSCALL environment'
   return
   end
```

Ending the SYSCALL Environment

The **syscalls('OFF')** function ends the connection between the current task and OS/390 OpenEdition.

- If the REXX program was run from TSO/E or MVS batch, the task is undubbed, but the REXX program continues running.
- If the REXX program was run from the shell or a program, the REXX program is ended.

In general, it is not necessary to make a **syscalls('OFF')** call.

The **syscalls('OFF')** function has one return value:

0	Successful completion.

Establishing and Deleting the Signal Interface Routine

The **syscalls('SIGON')** function establishes the signal interface routine (SIR). After you establish the SIR, use the **sigaction** syscall command to catch the signals you want to process and the **sigprocmask** syscall command to unblock those signals.

Note: For a REXX program run from the shell or a program, the SIR is established by default.

The **syscalls('SIGON')** function has these return values:

0	Successful completion.
4	The SIR could not be established. The usual cause for this is that another SIR has already been established for the process.

If you are writing a REXX program that runs a program that requires a signal interface routine (for example, a program that uses the C runtime library), you must delete the SIR. The **syscalls('SIGOFF')** function deletes the SIR and uses **sigprocmask()** to reset the signal process mask so that it blocks all signals that can be blocked.

The **syscalls('SIGOFF')** function has two return values:

0	Successful completion.
4	The SIR could not be deleted. The usual cause for this is that a SIR did not exist for the process.

The SH Environment

The SH environment is the default host command environment when a REXX program is run from the shell or from a program using exec(). In this environment, a syscall command runs as a shell command that has been issued this way:

```
/bin/sh -c shell_command
```

If you are running the REXX program from the shell or a program, the SYSCALL environment is automatically initialized.

See Chapter 4 for some sample REXX programs that show how REXX and shell commands can work together—for example, a REXX program that can read output from a shell command. The **mount** and **unmount** sample programs in that chapter are shipped in the **/samples** directory as files **mountx** and **unmountx**.

Running a REXX Program from the Shell or from a Program

You can run a REXX program from the shell or you can call it from any program just as an executable program would be called. The REXX program runs as a separate process; it does not run in a TSO/E address space. You cannot use TSO/E commands in the REXX program.

A REXX program invoked from the shell or from a program must be a text file or compiled REXX program that resides in the hierarchical file system (HFS); it must have read and execute access permissions. Each line in the text file must be terminated by a newline character and should not exceed 2048 characters. Lines are passed to the REXX interpreter as is. Sequence numbers are not supported, so if you are using the ISPF editor to create the REXX program, be sure to set NUMBER OFF.

If you are working in the shell environment and use only a filename to invoke the REXX program, the PATH environment variable is used to locate it. For example, **myrexx** uses PATH to locate the program, but **./myrexx** searches only the working directory.

For a REXX program run from the shell or from a program, the SIR is established by default. If the REXX program calls a C program that is running POSIX(ON) or a program that requires an SIR, use the **syscalls('SIGOFF')** function to delete the SIR before calling that program.

CEXEC output from the REXX compiler is supported in the shell environment. To compile and put CEXEC output into the HFS, you can use the REXXOEC cataloged procedure; it compiles under TSO/E and then uses the TSO/E OCOPY command to copy the compiled program from a data set to a file in the file hierarchy.

Using External Functions and Subroutines

You can call external functions and subroutines from a REXX program that resides in the HFS. The search path for an external routine is similar to that used for a REXX program invoked from the shell or a program. If only the filename is used on the call to the function or subroutine, the PATH environment variable is used to locate it; otherwise, the function name determines the search. For an executable module, the link pack area (LPA), link list, and STEPLIB may also be searched. The default OS/390 environment searches for executable modules first. See "Customizing the Environment" on page 10.

The search order for modules and execs invoked as functions or subroutines is controlled by the FUNCSOFL flag in the REXX parameter module. For a description of that flag, see *OS/390 TSO/E REXX Reference*.

Two rules must be observed in naming and calling an external function or subroutine:

- If the name contains special characters or lowercase characters, you must enclose it in quotes—for example:

  ```
  ans='myfunc'(p1,p2)
  ```

If the name is not quoted, REXX folds the name to uppercase. The function call then fails, because the file is not found.

- If the function name is longer than 8 characters, REXX truncates it to 8 characters.

Executable external functions or subroutines written in a language other than interpreted REXX and located in the hierarchical file system are not supported.

Variable Scope

When the REXX program is initialized and the SYSCALL environment is established, the predefined variables are set up. If you call an internal subroutine that uses the PROCEDURE instruction to protect existing variables by making them unknown to that subroutine (or function), the predefined variables also become unknown. If some of the predefined variables are needed, you can either list them on the PROCEDURE EXPOSE instruction or issue another **syscalls('ON')** to reestablish the predefined variables. The predefined variables are automatically set up for external functions and subroutines. For example:

```
subroutine: procedure
junk = syscalls('ON')
parse arg dir
'readdir (dir) dir. stem.'
```

Writing setuid and setgid REXX Programs

Setting the set-group-ID-on-execution (setgid) permission means that when a file is run, the calling process's effective GID is set to the file's owner GID; the process seems to be running under the GID of the file's owner, rather than that of the actual invoker.

Setting the set-user-ID-on-execution (setuid) permission means that when a file is run, the calling process's effective UID is set to the file's owner UID; the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

Just like any other setuid or setgid program, a REXX program should not let the user of the program get control in your environment. Some examples of instructions that can let a user obtain control are:

- Interactive trace.
- Calling external functions or subroutines: Using a relative pathname can let the user get control if the user sets the PATH variable. External functions and subroutines run under the UID and GID of the main program, regardless of their setuid and setgid mode bits.

Input and Output for OS/390 OpenEdition Processing

When a REXX program runs, open file descriptors are inherited from the process that issued the **exec()**.

Using Standard Input, Output, and Error (File Descriptors 0, 1, and 2)

For a REXX program run from the shell or a program, file descriptors 0, 1, and 2 (conventionally, standard input, standard output, and standard error files) are typically inherited.

Attention: A read or write error on file descriptors 0, 1, or 2 results in a halt interruption if the read or write was from a PARSE EXTERNAL instruction, SAY instruction, or EXECIO.

If the REXX program issues a PARSE EXTERNAL instruction, either explicitly or implicitly (such as from a PARSE PULL instruction with an empty stack), it reads standard input for a single text record. The newline character is stripped from the record before it is returned to the REXX program. Standard input is assumed to be a text file, such as your terminal input.

If the REXX program issues a SAY instruction, the text is directed to standard output, and a newline character is appended to the end of the text. Messages issued by REXX, including error and trace messages, are similarly directed to standard output.

If PARSE EXTERNAL is used after standard input has reached the end of the file, null lines are returned. The end-of-file condition can be detected by EXECIO. For more information, see “Using EXECIO.”

Using SYSCALL Commands

The SYSCALL host command environment gives you more direct control over input and output. You can use:

- **readfile** to read an entire text file. See “readfile” on page 126 for more information.
- **writefile** to write an entire text file. See “writefile” on page 179 for more information.
- **read** to read bytes from any kind of file. See “read” on page 123 for more information.
- **write** to write bytes to any kind of file. See “write” on page 176 for more information.

Using EXECIO

EXECIO differs from **readfile** and **writefile** in that it operates on open files. To read or write a file in segments (for example, a line at a time), use this TSO/E REXX command:

```
address MVS "EXECIO"
```

The data can come from and go to the stack or a stem. You can also use it to read or write an entire file. As shown in the following diagram, OS/390 OpenEdition supports all the TSO/E REXX operands except OPEN, DISKRU, and *linenum*.

```
►►──EXECIO──┬─lines─┬──┬─DISKW──ddname──┬──────────────────────────────────────────┬──────────┬──►◄
            └─*─────┘  │                └─(──┬───────────────┬──┬───────┬──┬───┬──┘          │
                       │                     └─STEM var-name─┘  └─FINIS─┘  └─)─┘              │
                       └─DISKR──ddname──┬──────────────────────────────────────────────────┬──┘
                                        └─(──┬───────────────┬──┬───────┬──┬──────┬──┬───┬─┘
                                             ├─FIFO──────────┤  └─FINIS─┘  └─SKIP─┘  └─)─┘
                                             ├─LIFO──────────┤
                                             └─STEM var-name─┘
```

For the *ddname* operand, you can use the following pseudo-ddnames for OS/390 OpenEdition processing, when run from the shell or a program:

- File descriptors 0 to 7
- STDIN, STDOUT, STDERR

For information on EXECIO, see *OS/390 TSO/E REXX Reference*.

Exit Status from a REXX program

When a REXX program is run from the shell or a program it can return a return code. If the program returns a value in the range 0–255, that value is returned. Otherwise, a value of 255 is returned. If a program is terminated, REXX returns a value of 255.

Tokens Returned from the PARSE SOURCE Instruction

The tokens that are returned depend on where the REXX program is run: from the shell or from a program, from TSO/E, or from batch.

Running from the Shell or from a Program

When a REXX program runs in the shell environment or is called from a program, the PARSE SOURCE instruction returns nine tokens, in this order:

1. The string TSO
2. The string COMMAND, FUNCTION, or SUBROUTINE, depending on whether the program was invoked as a host command, from a function call in an expression, or using the CALL instruction
3. The first 8 characters of the name of the REXX program
4. The string PATH
5. The first 44 characters of the pathname of the REXX program
6. ? (question mark)
7. The name of the initial host command environment in uppercase: SH
8. The name of the address space in uppercase: OMVS
9. An 8-character user token: OpenMVS

To determine if the REXX program was run from the shell, use token 8 or 9.

Example

If **myexec** is invoked from the shell and resides in the working directory, and if PATH is set to **.:/bin**, a PARSE SOURCE instruction returns the following tokens:

```
TSO COMMAND  myexec  PATH  ./myexec  ?  SH  OMVS  OpenMVS
```

Running from TSO/E or Batch

If the REXX program runs from TSO/E or MVS batch, the PARSE SOURCE instruction returns the tokens described in *OS/390 TSO/E REXX Reference*.

Using the REXX Signal Services

The REXX signal services consist of the following syscall commands:

alarm
kill
pause
sigaction
sigpending
sigprocmask
sigsuspend
sleep

REXX does not include a service that allows you to attach your own signal catcher. Instead, you have the following options:

- To use the REXX signal catcher as the action for a signal, you can specify the SIG_CAT variable as the signal handler on **sigaction**.

  SIG_CAT can terminate various wait conditions without causing the process to end. If a signal arrives when the process is not currently waiting and the signal is not blocked, it may be lost.

  There are two primary uses for SIG_CAT: when you are using the **alarm** command, and when you want to avoid unexpected process termination for other unblocked signals.

  SIG_CAT causes a signal to interrupt conditions such as waits and blocks, but the application cannot determine which signal was delivered. It is not a traditional signal catcher, as implemented in the C language.

- To set the action to the default action, you can specify SIG_DFL as the signal handler on **sigaction**.
- To set the action to ignore the signal, you can specify SIG_IGN as the signal handler on **sigaction**.

POSIX.1 defines several C functions to manipulate signal sets. REXX does not define these functions; however, you can define each function using a single REXX statement, as shown in Table 1.

Table 1 (Page 1 of 2). REXX Statements for Defining Signal Sets

C Function	Equivalent REXX Statement
sigsetempty()	sigsetempty: return copies(0,64) Parameters: none Returns: signal set

Table 1 (Page 2 of 2). REXX Statements for Defining Signal Sets

C Function	Equivalent REXX Statement
sigfillset()	sigfillset: return copies(1,64) Parameters: none Returns: signal set
sigaddset()	sigaddset: return overlay(1,arg(1),arg(2)) Parameters: signal set, signal number Returns: signal set
sigdelset()	sigdelset: return overlay(0,arg(1),arg(2)) Parameters: signal set, signal number Returns: signal set
sigismember()	sigismember: return substr(arg(1),arg(2),1) Parameters: signal set, signal number Returns: 0 (not member) or 1 (is member)

Moving a REXX Program from TSO/E to the Shell

If you write a REXX program to run in TSO/E, it is likely that you will have to alter the REXX program to run it in the shell environment. Some of the differences between the two environments that you need to consider are:

- You cannot run TSO/E commands in the shell environment. Using the **spawn** syscall command, you can run shell commands from the TSO/E environment.
- Using the **syscalls('ON')** function at the beginning of the REXX program is required in TSO/E, but not in the shell environment. If you use **syscalls('ON')** in the shell environment, it clears the __*argv.* and __*environment.* stems. For this reason, it is not recommended that you use **syscalls('ON')** in a shell environment. Using **syscalls('ON')** in the shell environment also sets up the REXX predefined variables and blocks all signals.
- In TSO/E, the **syscalls('OFF')** function ends the OS/390 OpenEdition process, but the REXX program continues running. In the shell, the **syscalls('OFF')** function causes the REXX program to stop running.
- PARSE SOURCE returns different tokens in TSO/E and in the shell environment. A REXX program uses the tokens to determine how it was run.
- In TSO/E, the variables __*argv.0* and __*environment.0* are set to zero (0).

Using argv and environment Variables

The stem variables __argv and __environment are always set to the original values passed to the first-level REXX program, and they are visible to external REXX functions. You may want to use PARSE ARG instead of the __argv stem in external REXX programs. As the following two sample programs show, using the __argv stem from an external exec will return the same data as it did from the initial exec. In order for an external REXX program to get the arguments a caller is sending it, it must use **arg()** or PARSE ARG:

PGM1:

```
/* rexx */
say 'this is the main pgm'
say 'it was passed' _ _argv.0 'arguments:'
do i = 1 to _ _argv.0
  say '  Argument' i': "'_ _argv.i'"'
end

call 'pgm2' 'arguments', 'to pgm2'
```

PGM2:

```
/* rexx */
say 'This is pgm2'
say 'Using _ _argv stem, there are' _ _argv.0 'arguments.  They are:'
do i = 1 to _ _argv.0
  say '  Argument' i': "'_ _argv.i'"'
end

say 'Using arg(), there are' arg() 'arguments:'
do i = 1 to arg()
  say '  Argument' i': "'arg(i)'"'
end
```

Sample Execution

```
$ pgm1 'arguments to' 'pgm1'
this is the main pgm
it was passed 3 arguments:
  Argument 1: "pgm1"
  Argument 2: "arguments to"
  Argument 3: "pgm1"
This is pgm2
Using _ _argv stem, there are 3 arguments.  They are:
  Argument 1: "pgm1"
  Argument 2: "arguments to"
  Argument 3: "pgm1"
Using arg(), there are 2 arguments:
  Argument 1: "arguments"
  Argument 2: "to pgm2"
```

Customizing the Environment

When a REXX program is run from the shell or called from a program using **exec()**, the REXX environment that is established is created from the module BPXWRXEV. The source for this module is member BPXWRX01 in SYS1.SAMPLIB.

This environment is inherited from the default MVS REXX environment. However, the default handling of error messages from the REXX processor is overridden so that the messages are written to STDOUT. This is the same place to which output from the SAY instruction and trace information is sent.

You can further customize the sample member to alter the REXX environment for REXX programs running under OS/390 OpenEdition without affecting REXX programs running in the OS/390 environment. For detailed information on how to change the default values for initializing an environment, see *OS/390 TSO/E REXX Reference*.

Performance in the SYSCALL Environment

syscalls('ON') ensures that the SYSCALL host command environment is available in your REXX environment. If the call detects that SYSCALL is not available in your environment, it dynamically adds it.

Performance characteristics for dynamically added host commands are not as good as for host commands that are included in the initial environment: Every time a command is directed to the SYSCALL host command environment, the TSO/E REXX support loads the module for the SYSCALL host command.

To avoid this, include the SYSCALL host command in the three default TSO/E environments:

Module name	SYS1.SAMPLIB member name	REXX enviroment
IRXPARMS	TSOREXX1	MVS
IRXTSPRM	TSOREXX2	TSO
IRXISPRM	TSOREXX3	ISPF

Customizing IRXISPRM provides dramatic performance improvement for REXX programs that use syscall commands from TSO/E or MVS batch.

Make the following changes to the SYS1.SAMPLIB members to add the SYSCALL host command to that default environment:

1. Find the label SUBCOMTB_TOTAL and add 1 to its value. For example:

change	SUBCOMTB_TOTAL DC F'14'	to	SUBCOMTB_TOTAL DC F'15'

2. Find the label SUBCOMTB_USED and add 1 to its value. For example:

change	SUBCOMTB_USED DC F'14'	to	SUBCOMTB_USED DC F'15'

3. Find the end of the subcommand table, just before the label PACKTB or PACKTB_HEADER, and add the following lines:

```
SUBCOMTB_NAME_REXXIX    DC  CL8'SYSCALL '
SUBCOMTB_ROUTINE_REXXIX DC  CL8'BPXWREXX'
SUBCOMTB_TOKEN_REXXIX   DC  CL16' '
```

4. Assemble and link-edit the module and replace the default TSO/E module. These are normally installed in SYS1.LPALIB.

See *OS/390 TSO/E REXX Reference* for additional information on customizing the default environments.

Authorization

Users authorized to perform special functions are defined as having *appropriate privileges*, and they are called *superusers*. Appropriate privileges also belong to users with:

- A user ID of zero
- RACF-supported user privileges *trusted* and *privileged*, regardless of their user ID

A user can switch to superuser authority (with an effective UID of 0) if the user is permitted to the BPX.SUPERUSER FACILITY class profile within RACF. Either the OS/390 OpenEdition ISPF Shell or the **su** shell command can be used for switching to superuser authority.

Security

This book assumes that your operating system contains Resource Access Control Facility (RACF). You could use an equivalent security product updated to handle OS/390 OpenEdition security.

Chapter 2. OS/390 OpenEdition REXX Programming Services

An application that supports scripting or macro languages, such as an editor, can use REXX as the macro language. An application written in a programming language such as C can create an OS/390 OpenEdition REXX environment and run a REXX program directly. For information on using the TSO/E REXX programming services, such as IRXJCL and IRXEXEC, see *OS/390 TSO/E REXX Reference*.

Establishing an OS/390 OpenEdition REXX Environment from an Application

To create an OS/390 OpenEdition REXX environment, fetch and call the module BPXWRBLD from a key 8 problem state program. OS/390 linkage for C (that is, standard OS linkage) is required.

The BPXWRBLD module requires the following parameters:

16K area	A 16,000-byte environment area. This must persist for the life of the REXX environment.
arg count	The count of the number of REXX initialization arguments.
arg pointer array	An array of pointers to null-terminated strings, one for each REXX initialization argument. The array and the null-terminated strings must persist for the life of the REXX environment.
env count	The count of the number of environment variables to be exported to the REXX program.
env length pointer array	An array of pointers to fullwords, one for each environment variable. The fullword contains the length of the string that defines the environment variable, including the terminating null. The last element of the array must point to a fullword of `0`. The array and the fullwords must persist for the life of the REXX environment.
env pointer array	An array of pointers to null-terminated strings, one for each environment variable. Each string defines one environment variable. The array and the null-terminated strings must persist for the life of the REXX environment. The format of the string is `NAME=value` where `NAME` is the environment variable name, and `value` is the value for the environment variable followed by a null character.
REXX env addr	The address of a fullword where the address of the newly created REXX environment is returned.

If BPXWRBLD fails to create the environment, it returns the return code it received from the IRXINIT service. BPXWRBLD does not return any other codes.

The parameter list is a standard MVS variable-length parameter list. On entry, the following registers must be set:

Register 1	Address of the parameter list
Register 13	Address of a register save area
Register 14	Return address
Register 15	Entry point address

Register 1 contains the address of the parameter list:

```
R1 --> --------+
       |     --+----> 16K area
       --------+
       |     --+----> arg count
       --------+
       |     --+----> arg pointer array
       --------+
       |     --+----> env count
       --------+
       |     --+----> env length pointer array
       --------+
       |     --+----> env pointer array
       --------+
       |1    --+----> Rexx env addr
       --------+
```

When constructing arguments to the REXX program that are also passed to BPXWRBLD, keep in mind that:

- The only use of the the argument count and argument array is to populate the *__argv.* REXX variables. You can set the argument count to `0` if the REXX programs will always get their arguments using PARSE ARG or the **arg(1)** REXX function call. In this case, *__argv.0* is set to `0` when the REXX program is run.
- After the call to BPXWRBLD, do not alter the data that is pointed to by the environment pointer arrays or the arg pointer array.

Signals are not supported in this environment.

Running the REXX Program

Before calling a TSO/E REXX service to run the program, ensure that file descriptors `0`, 1, and 2 are open. The REXX program will fail if it attempts a PARSE EXTERNAL, EXECIO, or SAY and that function fails.

After the OS/390 OpenEdition REXX environment is established, the program can call either the IRXJCL or the IRXEXEC TSO/E REXX service to run the REXX program.

- If the IRXJCL service is used, the name of the REXX program is the first word of the IRXJCL parameter string. It is limited to 8 characters.

- If you request the IRXEXEC service to load the program, you must provide the name of the REXX program in the member field of the EXECBLK. Set the DDNAME field to spaces. This also limits the name of the REXX program to 8 characters. Names longer than 8 characters can be supported with additional programming effort. You would need to preload the program and build an INSTBLK instead of an EXECBLK for the IRXEXEC call. If the REXX program is compiled in CEXEC format, load it as a single-record program.

  If the name of the REXX program does not contain a slash (/), the PATH environment variable is used to locate the program.

The current REXX environment must be the OS/390 OpenEdition REXX environment. You cannot pass the environment to be used in Register `0`.

When the REXX program is being loaded, the IRXEXEC or the IRXJCL service uses one file descriptor to open the file, read it, and close it. If no file descriptor is available because the maximum number of file descriptors are already open, the program cannot be loaded.

Example: C/370 Program

This program creates a REXX environment and runs a REXX program:

```
#pragma strings(readonly)
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

typedef int EXTF();
#pragma linkage(EXTF,OS)

int main(int argc, char **argv) {
extern char **environ;              /* access environ variables     */
EXTF *irxjcl;                       /* pointer to IRXJCL routine    */
EXTF *bpxwrbld;                     /* pointer to BPXWRBLD routine  */
char *penvb;                        /* addr of REXX environment     */
int  i,j;                           /* temps                        */
long rcinit;                        /* return code                  */
int  **environlp;                   /* ptr to env length pointers   */
int  *environl;                     /* ptr to env lengths           */
char rxwork[16000];                 /* OE MVS env work area         */
char *execname="execname";          /* name of exec up to 8 chars   */
char *execparm="exec parameter string"; /* parm to exec             */
struct s_rxparm {                   /* parm to IRXJCL               */
   short len;                       /* halfword length of parm      */
   char name[8];                    /* area to hold exec name       */
   char space;                      /* one space                    */
   char text[253];                  /* big area for exec parm       */
   } *rxparm;

   /* if stdin or stdout are not open you might want to open file     */
   /* descriptors 0 and 1 here                                        */

   /* if no environ, probably tso or batch - make one                 */
   if (environ==NULL) {
      environ=(char **)malloc(8);         /* create one               */
      environ[0]="PATH=.";                /* set PATH to cwd          */
      environ[1]=NULL;                     /* env terminator         */
      };

   /* need to build the environment in the same format as expected by */
   /* the exec() callable service.  See                                */
   /* Assembler Callable Services for OpenEdition MVS.                 */

   /* the environ array must always end with a NULL element            */
   for (i=0;environ[i]!=NULL;i++);         /* count vars             */
   environlp=(int **)malloc(i*4+4);        /* get array for len ptrs */
   environl=(int *)malloc(i*4+4);          /* get words for len vals */
   for (j=0;j<i;j++) {
      environlp[j]=&environl[j];           /* point to len           */
      environl[j]=strlen(environ[j])+1;    /* set len word           */
      };
   environlp[j]=NULL;                      /* null entry at end      */
   environl[j]=0;

   /* load routines                                                    */
    irxjcl=(EXTF *)fetch("IRXJCL   ");
   bpxwrbld=(EXTF *)fetch("BPXWRBLD ");

   /* build the REXX environment                                       */
   rcinit=bpxwrbld(rxwork,
                  argc,argv,
                  i,environlp,environ,
               &penvb);
   if (rcinit!=0) {
      printf("environment create failed rc=%d\n",rcinit);
      return 255;
      };

   /* if you need to add subcommands or functions to the environment, */
   /* or create a new environment inheriting the current one, this is */
   /* the place to do it.  The user field in the environment is used  */
   /* by the OS/390 OpenEdition REXX support and must be preserved.      */

   /* run exec                                                         */
   rxparm=(struct s_rxparm *)malloc(strlen(execname)+
                                    strlen(execparm)+
                                    sizeof(struct s_rxparm));
   memset(rxparm->name,' ',sizeof(rxparm->name));
   memcpy(rxparm->name,execname,strlen(execname));
   rxparm->space=' ';
   memcpy(rxparm->text,execparm,i=strlen(execparm));
   rxparm->len=sizeof(rxparm->name)+sizeof(rxparm->space)+i;
   return irxjcl(rxparm);
}
```

Chapter 3. The Syscall Commands

Syscall commands invoke the OS/390 OpenEdition callable service that corresponds to the command verb (the first word of the command). The parameters that follow the command verb are specified in the same order as in POSIX.1 and the OS/390 OpenEdition callable services, where applicable.

For complete information about the processing of a particular syscall command, read about the callable service that it invokes, as described in *OS/390 OpenEdition Programming: Assembler Callable Services Reference*.

Specifying a Syscall Command

You must specify the syscall command parameters in the order indicated in the syscall command description.

syscall command name
The syscall command name is case-insensitive: you can specify it as uppercase, lowercase, or mixed case.

Parameters
You can specify several types of parameters, but most fall into the following categories:

pathname The pathname is case-sensitive, and it is specified as a string. The syscall commands can take a relative or absolute pathname as a parameter. The search for a relative pathname begins in your working directory:

- If you are running a REXX program from the shell, your working directory is inherited from your shell session.
- If you are running a REXX program in TSO/E, your working directory is typically your home directory.

Portable pathnames can use only the characters in the POSIX portable filename character set:

- Uppercase or lowercase A to Z
- Numbers 0 to 9
- Period (.)
- Underscore (_)
- Hyphen (-)

Do not include any nulls in a pathname.

mode The mode is a three- or four-digit number corresponding to the acc ess permission bits. Each digit must be in the range 0–7, and at least three digits must be specified. See Appendix B for more information on permissions.

stem The name of a *stem variable*. A stem can be used for input, output, or both. A stem is indicated by a . (period) at the end of the variable name.

- The variable name for the first value consists of the name of the stem variable with a 1 appended to it. The number is incremented for each value—for example, *vara.1*, *vara.2*, and *vara.3*.

- The variable name that contains the number of variables returned (excluding `0`) consists of the name of the stem variable with a `0` appended to it—for example, *vara.0*

If you omit the period from the end of the variable name, a numeric suffix is appended to the name (for example, *foo* would become *foo0*, *foo1*, and so on).

The name of a stem variable is case-insensitive.

variable The name of a REXX variable. The name is case-insensitive.

Specifying Numerics

All numbers are numeric REXX strings. Negative numbers can be preceded by a minus sign (–); others must be unsigned.

The SYSCALL environment supports a 10-digit field. If you are performing arithmetic on a field longer than 9 digits, you must set precision to 10. A range of up to 2^{31}–1 is supported.

Specifying Strings

You can specify a string in any of these ways:

String	Example
Any series of characters not containing a space. This example shows a pathname with no space in it.	`"creat /u/wjs/file 700"`
Any series of characters delimited by `'` and not containing `'`. This example shows a pathname with a space in it.	`"creat 'u/wjs/my file' 700"`
Any series of characters delimited by `"` and not containing `"`. This example shows a pathname with a space in it.	`'creat "u/wjs/my file" 700'`
A variable name enclosed in parentheses. Strings containing both the single and double quote characters must be stored in a variable, and you must use the variable name.	`file='/u/wjs/my file'` `"creat (file) 700"`

The following example uses a variable enclosed in parentheses to avoid problems with a blank in the filename:

```
file='/u/wjs/my file'
"creat (file) 700"
```

If you incorrectly coded the second line as:

```
"creat /u/wjs/my file 700"
```

it would contain four tokens instead of three.

Using Predefined Variables

Predefined variables make symbolic references easier and more consistent—for example, when you are specifying a flag or using a stem variable. Instead of coding a numeric value, you can specify the predefined variable that is used to derive that numeric value.

Appendix A lists all the predefined variables alphabetically and shows their numeric value and data type. If a variable is a stem variable, this shows the data type for the stem variable.

You can also use the index of this book as a reference: under the name of each syscall command are grouped the names of the predefined variables associated with it.

Return Values

A command can be issued to the SYSCALL environment or the SH environment, and the return values are different in the two environments.

Returned from the SYSCALL Environment

When a syscall command completes, the environment can set four reserved variables:

RC A numeric return code from the command execution.

Value Range	**Meaning**
0	The command finished successfully. If there is an error code for the requested function, it is returned in RETVAL and ERRNO.
>**0**	The command finished successfully, but a function-specific warning is indicated.
−3	The command environment has not been called. Probably the **syscalls('ON')** function did not end successfully, or the current address environment is not SYSCALL.
−20	The command was not recognized, or there was an improper number of parameters specified on the command.
−21,−22, ...	The first, second, ... parameter is in error. (The parameter is indicated by the second digit.)
<**0**	Other negative values may be returned by the REXX language processor. A negative value means that the command did not finish successfully.

RETVAL A numeric return value from the callable service. This variable is valid only if the return code (**RC**) is not negative.

ERRNO A hexadecimal error number from the callable service. This variable is valid only if the return code (**RC**) is not negative.

ERRNOJR A hexadecimal reason code from the callable service. This variable is valid only if the return code (**RC**) is not negative.

Returned from the SH Environment

When a command completes in the SH environment, the return code is set in the variable **RC**. Unusual situations cause the return code to be set to a negative value:

–**1xxx**	Terminated by signal xxx
–**2xxx**	Stopped by signal xxx
–**3xxx**	Fork failed with error number xxx
–**4xxx**	Exec failed with error number xxx
–**5xxx**	Wait failed with error number xxx

access

```
▶▶──access──pathname──flags──────────────────────────────────▶◀
```

Function

access invokes the access callable service to determine if the caller can access a file.

Parameters

pathname
The pathname of the file to be checked for accessibility.

flags
One or more numeric values that indicate the accessibility to be tested. You can specify a numeric value (see Appendix A) or the predefined variable used to derive the appropriate numeric value. The predefined variables you can specify are:

Variable	Description
F_OK	Test for file existence
R_OK	Test for permission to read
W_OK	Test for permission to write
X_OK	Test for permission to execute

For example, R_OK+W_OK tests for read and write permission.

Usage Notes

1. Testing for file permissions is based on the real user ID (UID) and real group ID (GID), not the effective UID or effective GID of the calling process.
2. The caller can test for the existence of a file or for access to the file, but not both.
3. In testing for permission, the caller can test for any combination of read, write, and execute permission. If the caller is testing a combination of permissions, a –1 is returned if any one of the accesses is not permitted.
4. If the caller has appropriate privileges, the access test is successful even if the permission bits are off, except when testing for execute permission. When the caller tests for execute permission, at least one of the execute permission bits must be on for the test to be successful.

Example

To test for permission to execute **grep**:

```
"access '/bin/grep'" x_ok
```

alarm

```
►►──alarm──seconds──────────────────────────────────────────►◄
```

Function

alarm invokes the alarm callable service to generate a **SIGALRM** signal after the number of seconds specified have elapsed.

Parameters

seconds
The number of seconds to pass between receipt of this request and generation of the **SIGALRM** signal.

Usage Notes

1. The default action for an alarm signal is to end a process.
2. The alarm callable service is always successful, and no return value is reserved to indicate an error.
3. An abend is generated when failures are encountered that prevent the alarm callable service from completing successfully.
4. Alarm requests are not stacked; only one **SIGALRM** can be scheduled to be generated at a time. If the previous alarm time did not expire and a new alarm is scheduled, the most recent alarm reschedules the time that **SIGALRM** is generated.
5. See “Using the REXX Signal Services” on page 8 for additional information on using signals.

Example

To generate a **SIGALRM** after 10 seconds:

```
"alarm 10"
```

catclose

```
►►──catclose──catalog_descriptor──────────────────────────────────►◄
```

Function

catclose closes a message catalog that was opened by **catopen**.

Parameters

catalog_descriptor
The catalog descriptor (a number) returned by **catopen** when the message catalog was opened.

Usage Notes

If unsuccessful, **catclose** returns –1 and sets ERRNO to indicate the error.

Example

See the example for “catgets” on page 24.

catgets

```
►►──catgets──catalog_descriptor──set_number──────────────────────►
►──message_number──variable─────────────────────────────────────►◄
```

Function

catgets locates and returns a message in a message catalog.

Parameters

catalog_descriptor
The catalog descriptor (a number) returned by **catopen** when a message catalog was opened earlier.

set_number
A number that identifies a message set in the message catalog.

message_number
A number that identifies a message in a message set in the message catalog.

variable
The name of the buffer in which the message string is returned.

Usage Notes

1. Set *variable* to a default message text prior to invoking the **catgets** command. If the message identified by *message_number* is not found, *variable* is not altered and can be used after the command has been invoked.
2. If the command is unsuccessful, *variable* is returned and ERRNO may be set to indicate the error.

Example

```
"catopen mymsgs.cat"
cd=retval
⋮
msg='error processing request'
"catgets (cd) 1 3 msg"
say msg
⋮
"catclose" cd
```

catopen

```
►►──catopen──catalog_name──────────────────────────────────────────►◄
```

Function

catopen opens a message catalog that has been built by the **gencat** utility. (For more information about **gencat**, see *OS/390 OpenEdition Command Reference*.) The catalog descriptor is returned in RETVAL.

Parameters

catalog name
The pathname for the message catalog. If the pathname contains a slash (/), the environment variables **NLSPATH** and **LANG** do not affect the resolution of the pathname.

Usage Notes

1. The catalog descriptor returned in RETVAL can be used with the **catgets** and **catclose** commands. Do not use the catalog descriptor with any other commands.
2. If unsuccessful, **catopen** returns a −1 and sets ERRNO to indicate the error.

Example

See the example for “catgets” on page 24.

chattr

```
►►──chattr──pathname──attribute_list──────────────────────────────►◄
```

Function

chattr invokes the chattr callable service to set the attributes associated with a file. You can change the file mode, owner, access time, modification time, change time, reference time, audit flags, general attribute flags, and file size.

Parameters

pathname
The pathname of the file.

attribute_list
A list of attributes to be set and their values. The attributes are expressed either as numeric values (see Appendix A), or as the predefined variables beginning with ST_, followed by arguments for that attribute. The attributes that may be changed and their parameters are:

Variable	Description
ST_MODE	1 argument: permission bits as 3 octal digits.
ST_UID	2 arguments: UID and GID numbers.
ST_SIZE	1 argument: new file size.
ST_ATIME	1 argument for access time: new time or −1 for TOD.
ST_MTIME	1 argument for modification time: new time or −1 for TOD.
ST_CTIME	1 argument for change time: new time or −1 for TOD.
ST_SETUID	No arguments.
ST_SETGID	No arguments.
ST_AAUDIT	1 argument: new auditor audit value.
ST_UAUDIT	1 argument: new user audit value.
ST_STICKY	No arguments.
ST_GENVALUE	2 arguments: names of two variables. The first variable contains the general attribute mask and the second contains the general attribute value.
ST_RTIME	1 argument for reference time: new time or −1 for TOD.
| ST_FILEFMT	Format of the file. To specify the format, you can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value:

| S_FFBINARY	Binary data
S_FFCR	Text data delimited by a carriage return character
S_FFCRLF	Text data delimited by carriage return and line feed characters
S_FFLF	Text data delimited by a line feed character
S_FFLFCR	Text data delimited by a line feed and carriage return characters
S_FFNA	Text data with the file format not specified
S_FFNL	Text data delimited by a newline character

Usage Notes

1. Some of the attributes changed by the chattr service can also be changed by other services.

2. When changing the mode:

   - The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.

   - Setting the set-group-ID-on-execution permission (in mode) means that when this file is run (through the exec service), the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

     The set-group-ID-on-execution permission is set to zero if both of the following are true:

     – The caller does not have appropriate privileges.

     – The GID of the file's owner does not match the effective GID, or one of the supplementary GIDs, of the caller.

   - Setting the set-user-ID-on-execution permission (in mode) means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

3. When changing the owner:

   - For changing the owner UID of a file, the caller must have appropriate privileges.

   - For changing the owner GID of a file, the caller must have appropriate privileges, or meet all of these conditions:

     – The effective UID of the caller matches the file's owner UID.

     – The owner UID value specified in the change request matches the file's owner UID.

     – The GID value specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.

   - When changing the owner, the set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.

   - When the owner is changed, both UID and GID must be specified as they are to be set. If you want to change only one of these values, you need to set the other to its present value for it to remain unchanged.

4. For general attribute bits to be changed, the calling process must have write permission for the file.

5. When changing the file size:

   - The change is made beginning from the first byte of the file. If the file was previously larger than the new size, the data from *file_size* to the original end of the file is removed. If the file was previously shorter than *file_size*, bytes between the old and new lengths are read as zeros. The file offset is not changed.

- If *file_size* is greater than the current file size limit for the process, the request fails with EFBIG, and the SIGXFSZ signal is generated for the process.
- Successful change clears the set-user-ID, the set-group-ID, and the save-text (sticky bit) attributes of the file, unless the caller is a superuser.

6. When changing times:
   - For the access time or the modification time to be set explicitly (using either *st_atime* or *st_mtime* with the new time), the effective ID must match that of the file's owner, or the process must have appropriate privileges.
   - For the access time or modification time to be set to the current time (using either *st_atime* or *st_mtime* with −1), the effective ID must match that of the file's owner, the calling process must have write permission for the file, or the process must have appropriate privileges.
   - For the change time or the reference time to be set explicitly (using either *st_ctime* or *st_rtime* with the new time), the effective ID must match that of the file's owner, or the process must have appropriate privileges.
   - For the change time or reference time to be set to the current time (using either *st_ctime* or *st_rtime* with −1), the calling process must have write permission for the file.
   - When any attribute field is changed successfully, the file's change time is updated as well.
7. For auditor audit flags to be changed, the user must have auditor authority. The user with auditor authority can set the auditor options for any file, even those to which they do not have path access or authority to use for other purposes.

   Auditor authority is established by issuing the TSO/E command ALTUSER AUDITOR.
8. For the user audit flags to be changed, the user must have appropriate privileges or be the owner of the file.

Example

To set permissions for **/u/project** to 775:

```
"chattr /u/project" st_mode 775
```

chaudit

```
►►──chaudit──pathname──audit_flags──option──────────────────────►◄
```

Function

chaudit invokes the chaudit callable service to change audit flags for a file.

Parameters

pathname
The pathname of the file.

audit_flags
One or more numeric values that indicate the type of access to be tested. You can specify a numeric value (see Appendix A) or the predefined variable used to derive the appropriate numeric value. The predefined variables you can specify are:

Variable	Description
AUD_FREAD	Audit failed read requests
AUD_SREAD	Audit successful read requests
AUD_FWRITE	Audit failed write requests
AUD_SWRITE	Audit successful write requests
AUD_FEXEC	Audit failed execute or search requests
AUD_SEXEC	Audit successful execute or search requests

option
A number indicating whether user-requested or auditor-requested auditing is being changed:

- `0` if user-requested auditing is being changed.
- `1` if auditor-requested auditing is being changed.

Usage Notes

1. If *option* indicates that the auditor audit flags are to be changed, you must have auditor authority for the request to be successful. If you have auditor authority, you can set the auditor options for any file, even those to which you do not have path access or authority to use for other purposes.

   You can get auditor authority by entering the TSO/E command ALTUSER AUDITOR.

2. If *option* indicates that the user audit flags are to be changed, you must have appropriate privileges or be the owner of the file.

Example

In the following example, assume that *pathname* was assigned a value earlier in the exec. To change user-requested auditing so that failed read, write, and execute attempts for *pathname* are audited:

```
"chaudit (pathname)" aud_fread+aud_fwrite+aud_fexec 0
```

chdir

```
►►──chdir──pathname──────────────────────────────────────────────►◄
```

Function

chdir invokes the chdir callable service to change the working directory.

Parameters

pathname
The pathname of the directory.

Usage Notes

If you use **chdir** to change a directory in a REXX program that is running in a TSO/E session, the directory is typically reset to your home directory when the REXX program ends. When a REXX program changes directories and then exits, the thread is undubbed. If this was the only thread dubbed in your TSO/E session, the working directory is reset to the home directory the next time a syscall command is issued. However, if there is more than one dubbed thread in the address space, the remaining threads keep the working directory even when the REXX program exits.

Example

To change the working directory to **/u/lou/dirb**:

```
"chdir /u/lou/dirb"
```

chmod

```
►►──chmod──pathname──mode──┬────────────────────────────┬──────────►◄
                           └─setuid──setgid──┬────────┬─┘
                                             └─sticky─┘
```

Function

chmod invokes the chmod callable service to change the mode of a file or directory.

Parameters

pathname
The pathname of the file or directory.

mode
A three- or four-digit number, corresponding to the access permission bits. Each digit must be in the range `0`–`7`, and at least three digits must be specified. For more information on permissions, see Appendix B.

setuid
Sets the set-user-ID-on-execution permission. Specify `1` to set this permission on, or `0` to set it off. The default is `0`.

setgid
Sets the set-group-ID-on-execution permission. Specify `1` to set this permission on, or `0` to set it off. The default is `0`.

sticky
The sticky bit for a file indicates where the file should be fetched from. If the file resides in the link pack area (LPA), link list, or STEPLIB, specify `1`. The default is `0`.

Setting the sticky bit for a directory to `1` indicates that to delete or rename a file, the effective user ID of the process must be the same as that of the directory owner or file owner, or that of a superuser. Setting the sticky bit for a directory to `0` indicates that anyone who has write permission to the directory can delete or rename a file.

Usage Notes

1. One bit sets permission for set-user-ID on access, set-group-ID on access, or the *sticky bit*. You can set this bit in either of two ways:
   - Specifying four digits on the *mode* parameter; the first digit sets the bit.
   - Specifying the *setuid*, *setgid*, or *sticky* parameters.
2. When a **chmod** or **fchmod** has occurred for an open file, **fstat** reflects the change in mode. However, no change in access authorization is apparent when the file is accessed through a previously opened file descriptor.
3. For mode bits to be changed, the effective UID of the caller must match the file's owner UID, or the caller must be a superuser.
4. When the mode is changed successfully, the file's change time is updated as well.
5. Setting the set-group-ID-on-execution permission means that when this file is run (through the exec service), the effective GID of the caller is set to the file's

owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

The set-group-ID-on-execution permission is set to zero if both of the following are true:

- The caller does not have appropriate privileges.
- The GID of the file's owner does not match the effective GID or one of the supplementary GIDs of the caller.

6. Setting the set-user-ID-on-execution permission means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

Example

In the following example, assume that *pathname* was assigned a value earlier in the exec. This example changes the mode of the file to read-write-execute for the owner, and read-execute for all others:

```
"chmod (pathname) 755"
```

chown

```
►►──chown──pathname──uid──gid──────────────────────────────────►◄
```

Function

chown invokes the chown callable service to change the owner or group for a file or directory.

Parameters

pathname
The pathname of a file or directory.

uid
The numeric UID for the new owner of the file or the present UID, or −1 if there is no change.

gid
The numeric GID for the group for the file or the present GID, or −1 if there is no change.

Usage Notes

1. The chown service changes the owner UID and owner GID of a file. Only a superuser can change the owner UID of a file.
2. The owner GID of a file can be changed by a superuser, or if a caller meets all of these conditions:
   - The effective UID of the caller matches the file's owner UID.
   - The *uid* value specified in the change request matches the file's owner UID.
   - The *gid* value specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both *uid* and *gid* must be specified as they are to be set. If you want to change only one of these values, the other must be set to its present value to remain unchanged.

Example

In the following example, assume that *pathname*, *uid*, and *gid* were assigned a value earlier in the exec:

```
"chown (pathname) (uid) (gid)"
```

close

```
►►──close──fd──────────────────────────────────────────────►◄
```

Function

close invokes the close callable service to close a file.

Parameters

fd The file descriptor (a number) for the file to be closed.

Usage Notes

1. Closing a file closes, or frees, the file descriptor by which the file was known to the process. The system can then reassign the file descriptor to the same file or to another file when it is opened.
2. Closing a file descriptor also unlocks all outstanding byte range locks that a process has on the associated file.
3. If a file has been opened by more than one process, each process has a file descriptor. When the last open file descriptor is closed, the file itself is closed. If the file's link count is zero at that time, the file's space is freed and the file becomes inaccessible. When the last open file descriptor for a pipe or FIFO special file is closed, any data remaining in the file is discarded.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec:

```
"close" fd
```

closedir

```
►►──closedir──fd──────────────────────────────────────────────►◄
```

Function

closedir invokes the closedir callable service to close a directory.

Parameters

fd The file descriptor (a number) for the directory to be closed.

Usage Notes

closedir closes a directory file descriptor opened by the **opendir** syscall command. The **rddir** command reads a directory in the readdir callable service format. You can use **opendir**, **rewinddir**, and **closedir** together with the **rddir** syscall command, but not with the **readdir** syscall command. Alternatively, you can simply use the **readdir** syscall command to read an entire directory and format it in a stem.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec:

```
"closedir" fd
```

creat

```
►►──creat──pathname──mode──────────────────────────────────────►◄
```

Function

creat invokes the open callable service to open a new file. The file descriptor is returned in RETVAL.

Parameters

pathname
The pathname of a file.

mode
A three- or four-digit number, corresponding to the access permission bits. Each digit must be in the range `0`–`7`, and at least three digits must be specified. For more information on permissions, see Appendix B.

Usage Notes

Using **creat** is the equivalent of using the open callable service with the create, truncate, and write-only options:

- When a file is created with the create option, the file permission bits as specified in *mode* are modified by the process's file creation mask (see “umask” on page 165) and then used to set the file permission bits of the file being created.
- The truncate option opens the file as though it had been created earlier, but never written into. The mode and owner of the file do not change (although the change time and modification time do), but the file's contents are discarded. The file offset, which indicates where the next write is to occur, points to the first byte of the file.

Example

To open a new file, **/u/lou/test.exec**, with read-write-execute permission for the owner only:

```
"creat /u/lou/test.exec 700"
```

dup

```
►►──dup──fd──────────────────────────────────────────────────────►◄
```

Function

dup invokes the fcntl callable service to duplicate an open file descriptor. The file descriptor is returned in RETVAL.

Parameters

fd An opened file descriptor (a number) to be duplicated.

Usage Notes

dup *fd* is equivalent to F_DUPFD *fd* `0`.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec:

```
"dup (fd)"
```

dup2

```
►►──dup2──fd──fd2──────────────────────────────────────────────►◄
```

Function

dup2 invokes the fcntl callable service to duplicate an open file descriptor to the file descriptor of choice. The file descriptor returned is equal to *fd2*. If *fd2* is already in use, it is closed and *fd* is duplicated. If *fd* is equal to *fd2*, *fd2* is returned without closing it. The file descriptor is returned in RETVAL.

Parameters

fd An opened file descriptor (a number) to be duplicated.

fd2
The file descriptor (a number) to be changed.

Usage Notes

dup *fd fd2* is equivalent to F_DUPFD *fd fd2*.

Example

In the following example, assume that *fd1* and *fd2* were assigned values earlier in the exec:

```
"dup2" fd1 fd2
```

exec

There is no **exec** syscall command. Instead of using **exec**, see “spawn” on page 151.

extlink

```
►►──extlink──extname──linkname──────────────────────────────────────►◄
```

Function

extlink invokes the extlink callable service to create a symbolic link to an external name. This creates a symbolic link file.

Parameters

extname
The external name of the file for which you are creating a symbolic link.

linkname
The pathname for the symbolic link.

Usage Notes

1. The object identified by *extname* need not exist when the symbolic link is created, and refers to an object outside a hierarchical file system.
2. The external name contained in an external symbolic link is not resolved. The *linkname* cannot be used as a directory component of a pathname.

Example

To create a symbolic link named **mydsn** for the file **WJS.MY.DSN**:

```
"extlink WJS.MY.DSN /u/wjs/mydsn"
```

fchattr

```
►►──fchattr──fd──attribute_list──────────────────────────────►◄
```

Function

fchattr invokes the fchattr callable service to modify the attributes that are associated with a file represented by a file descriptor. You can change the mode, owner, access time, modification time, change time, reference time, audit flags, general attribute flags, and file size.

Parameters

fd The file descriptor for the file.

attribute_list
A list of attributes to be set and their values. The attributes are expressed either as numeric values (see Appendix A), or as the predefined variables beginning with ST_ followed by arguments for that attribute. The attributes that may be changed and their parameters are:

Variable	Description
ST_MODE	1 argument: permission bits as 3 octal digits.
ST_UID	2 arguments: UID and GID numbers.
ST_SIZE	1 argument: new file size.
ST_ATIME	1 argument for access time: new time or −1 for TOD.
ST_MTIME	1 argument for modification time: new time or −1 for TOD.
ST_CTIME	1 argument for change time: new time or −1 for TOD.
ST_SETUID	No arguments.
ST_SETGID	No arguments.
ST_AAUDIT	1 argument: new auditor audit value.
ST_UAUDIT	1 argument: new user audit value.
ST_STICKY	No arguments.
ST_GENVALUE	2 arguments: names of two variables. The first variable contains the general attribute mask and the second contains the general attribute value.
ST_RTIME	1 argument for reference time: new time or −1 for TOD.
ST_FILEFMT	Format of the file. To specify the format, you can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value:

S_FFBINARY	Binary data
S_FFCR	Text data delimited by a carriage return character
S_FFCRLF	Text data delimited by carriage return and line feed characters
S_FFLF	Text data delimited by a line feed character
S_FFLFCR	Text data delimited by a line feed and carriage return characters
S_FFNA	Text data with the file format not specified
S_FFNL	Text data delimited by a newline character

Usage Notes

1. Some of the attributes changed by the fchattr service can also be changed by other services.
2. When changing the mode:
   - The effective UID of the calling process must match the file's owner UID, or the caller must have appropriate privileges.
   - Setting the set-group-ID-on-execution permission (in mode) means that when this file is run (through the exec service), the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

     The set-group-ID-on-execution permission is set to zero if both of the following are true:
     - The caller does not have appropriate privileges.
     - The GID of the file's owner does not match the effective GID, or one of the supplementary GIDs, of the caller.
   - Setting the set-user-ID-on-execution permission (in mode) means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.
3. When changing the owner:
   - For changing the owner UID of a file, the caller must have appropriate privileges.
   - For changing the owner GID of a file, the caller must have appropriate privileges, or meet all of these conditions:
     - The effective UID of the caller matches the file's owner UID.
     - The owner UID value specified in the change request matches the file's owner UID.
     - The GID value specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
   - When the owner is changed, the set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
   - When the owner is changed, both UID and GID must be specified as they are to be set. If you want to change only one of these values, you need to set the other to its present value for it to remain unchanged.
4. For general attribute bits to be changed, the calling process must have write permission for the file.
5. When changing the file size:
   - The change is made beginning from the first byte of the file. If the file was previously larger than the new size, the data from *file_size* to the original end of the file is removed. If the file was previously shorter than *file_size*, bytes between the old and new lengths are read as zeros. The file offset is not changed.

   - If *file_size* is greater than the current file size limit for the process, the request fails with EFBIG and the SIGXFSZ signal is generated for the process.
   - Successful change clears the set-user-ID, set-group-ID, and sa ve-text (sticky bit) attributes of the file unless the caller is a superuser.

6. When changing times:
   - For the access time or the modification time to be set explicitly (using either *st_atime* or *st_mtime* with the new time), the effective ID must match that of the file's owner, or the process must have appropriate privileges.
   - For the access time or modification time to be set to the current time (using either *st_atime* or *st_mtime* with −1), the effective ID must match that of the file's owner, the calling process must have write permission for the file, or the process must have appropriate privileges.
   - For the change time or the reference time to be set explicitly (using either *st_ctime* or *st_rtime* with the new time) the effective ID must match that of the file's owner, or the process must have appropriate privileges.
   - For the change time or reference time to be set to the current time (using either *st_ctime* or *st_rtime* with −1), the calling process must have write permission for the file.
   - When any attribute field is changed successfully, the file's change time is updated as well.
7. For auditor audit flags to be changed, the user must have auditor authority. The user with auditor authority can set the auditor options for any file, even those to which they don't have path access or authority to use for other purposes.

   Auditor authority is established by issuing the TSO/E command ALTUSER AUDITOR.
8. For the user audit flags to be changed, the user must have appropriate privileges or be the owner of the file.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec. This truncates a file to `0` bytes and sets the file permissions to 600:

```
"fchattr" fd st_size 0 st_mode 600
```

fchaudit

```
►►──fchaudit──fd──audit_flags──option──────────────────────────►◄
```

Function

fchaudit invokes the fchaudit callable service to change audit flags for a file identified by a file descriptor. The file descriptor is specified by a number.

Parameters

fd The file descriptor for the file.

audit_flags
One or more numeric values that indicate the type of access to be tested. You can specify a numeric value (see Appendix A) or the predefined variable used to derive the appropriate numeric value. The predefined variables you can specify are:

Variable	Description
AUD_FREAD	Audit failed read requests
AUD_SREAD	Audit successful read requests
AUD_FWRITE	Audit failed write requests
AUD_SWRITE	Audit successful write requests
AUD_FEXEC	Audit failed execute or search requests
AUD_SEXEC	Audit successful execute or search requests

option
A number indicating whether user-requested or auditor-requested auditing is being changed:

- `0` if user-requested auditing is being changed.
- `1` if auditor-requested auditing is being changed.

Usage Notes

1. If *option* indicates that the auditor audit flags are to be changed, you must have auditor authority for the request to be successful. If you have auditor authority, you can set the auditor options for any file, even those to which you do not have path access or authority to use for other purposes.

   You can get auditor authority by entering the TSO/E command ALTUSER AUDITOR.

2. If *option* indicates that the user audit flags are to be changed, you must have appropriate privileges or be the owner of the file.

Example

To change user-requested auditing so that failed read requests for the file identified by file descriptor `0` are audited:

```
"fchaudit 0 (aud_fread) 0"
```

fchmod

```
►►──fchmod──fd──mode──┬──────────────────────────┬──────────────►◄
                      └─setuid──setgid──┬────────┬┘
                                        └─sticky─┘
```

Function

fchmod invokes the fchmod callable service to change the mode of a file or directory indicated by a file descriptor. The file descriptor is specified by a number.

Parameters

fd The file descriptor for the file or directory.

mode
A three- or four-digit number, corresponding to the access permission bits. Each digit must be in the range `0`–`7`, and at least three digits must be specified. For more information on permissions, see Appendix B.

setuid
Sets the set-user-ID-on-execution permission. Specify `1` to set this permission on, or `0` to set it off. The default is `0`.

setgid
Sets the set-group-ID-on-execution permission. Specify `1` to set this permission on, or `0` to set it off. The default is `0`.

sticky
Sets the sticky bit to indicate where the file should be fetched from. If the file resides in the link pack area (LPA), link list, or STEPLIB, specify `1`. The default is `0`.

Usage Notes

1. One bit sets permission for set-user-ID on access, set-group-ID on access, or the *sticky bit*. You can set this bit in either of two ways:
   - Specifying four digits on the *mode* parameter; the first digit sets the bit.
   - Specifying the *setuid*, *setgid*, or *sticky* parameters.
2. When a **chmod** or **fchmod** has occurred for an open file, **fstat** reflects the change in mode. However, no change in access authorization is apparent when the file is accessed through a previously opened file descriptor.
3. For mode bits to be changed, the effective UID of the caller must match the file's owner UID, or the caller must be a superuser.
4. When the mode is changed successfully, the file's change time is updated as well.
5. Setting the set-group-ID-on-execution permission means that when this file is run, through the exec service, the effective GID of the caller is set to the file's owner GID, so that the caller seems to be running under the GID of the file, rather than that of the actual invoker.

   The set-group-ID-on-execution permission is set to zero if both of the following are true:
   - The caller does not have appropriate privileges.

- The GID of the file's owner does not match the effective GID or one of the supplementary GIDs of the caller.

6. Setting the set-user-ID-on-execution permission means that when this file is run, the process's effective UID is set to the file's owner UID, so that the process seems to be running under the UID of the file's owner, rather than that of the actual invoker.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec. This changes the mode for the file identified by the file descriptor so that only a superuser can access the file:

```
"fchmod (fd) 000"
```

fchown

```
►►──fchown──fd──uid──gid─────────────────────────────────────────►◄
```

Function

fchown invokes the fchown callable service to change the owner and group of a file or directory indicated by a file descriptor. The file descriptor is specified by a number.

Parameters

fd The file descriptor for a file or directory.

uid
The numeric UID for the new owner of the file or the present UID, or −1 if there is no change.

gid
The numeric GID for the new group for the file or the present GID, or −1 if there is no change.

Usage Notes

1. The fchown service changes the owner UID and owner GID of a file. Only a superuser can change the owner UID of a file.
2. The owner GID of a file can be changed by a superuser, or if a caller meets all of these conditions:
   - The effective UID of the caller matches the file's owner UID.
   - The *uid* value specified in the change request matches the file's owner UID.
   - The *gid* value specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
3. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
4. If the change request is successful, the change time for the file is updated.
5. Values for both *uid* and *gid* must be specified as they are to be set. If you want to change only one of these values, the other must be set to its present value to remain unchanged.

Example

In the following example, assume that *fd*, *uid*, and *gid* were assigned a value earlier in the exec:

```
"fchown" fd uid gid
```

f_closfd

```
►►──f_closfd──fd──fd2──────────────────────────────────────────────►◄
```

Function

f_closfd invokes the fcntl callable service to close a range of file descriptors.

Parameters

fd The file descriptor (a number) for a file. This is the first file descriptor to be closed.

fd2
The file descriptor (a number) for a file, which must be greater than or equal to *fd*. If a −1 is specified for *fd2*, all file descriptors greater than or equal to *fd* are closed.

Usage Notes

1. A process can use **f_closfd** to close a range of file descriptors. *fd2* must be greater than or equal to *fd*, or it can also be −1, which indicates that all file descriptors greater than or equal to *fd* are to be closed.
2. Use of **f_closfd** is meant to be consistent with the close callable service. You cannot close file descriptors that could not also be closed using the close service.
3. When a file descriptor cannot be closed, it is considered an error, but the request continues with the next file descriptor in the range. File descriptors that are not in use are ignored.

Example

In the following example, assume that *fd* and *fd2* were assigned values earlier in the exec:

```
"f_closfd" fd fd2
```

fcntl

Function

fcntl is supported as a set of syscall commands whose names begin with f_:

“f_closfd” on page 48
“f_dupfd” on page 50
“f_dupfd2” on page 51
“f_getfd” on page 52
“f_getfl” on page 53
“f_getlk” on page 54
“f_setfd” on page 61
“f_setfl” on page 62
“f_setlk” on page 63
“f_setlkw” on page 65

f_dupfd

```
►►──f_dupfd──fd──fd2──────────────────────────────────────────►◄
```

Function

f_dupfd invokes the fcntl callable service to duplicate the lowest file descriptor that is equal to or greater than *fd2* and not already associated with an open file. The file descriptor is returned in RETVAL.

Parameters

fd The file descriptor (a number) that you want to duplicate.

fd2
The file descriptor (a number) at which to start looking for an available file descriptor.

Example

In the following example, assume that *fd* and *fd2* were assigned values earlier in the exec:

```
"f_dupfd" fd fd2
```

f_dupfd2

```
►►──f_dupfd2──fd──fd2──────────────────────────────────────────────►◄
```

Function

f_dupf2d invokes the fcntl callable service to duplicate a file descriptor that is equal to *fd2*.

Parameters

fd An opened file descriptor (a number) to be duplicated.

fd2
The file descriptor of choice.

Usage Notes

If *fd2* is already in use, **f_dupfd2** closes it. If *fd* is equal to *fd2*, *fd2* is returned but not closed.

Example

In the following example, assume that *fd* and *fd2* were assigned values earlier in the exec:

```
"f_dupfd" fd fd2
```

f_getfd

```
►►──f_getfd──fd──────────────────────────────────────────────►◄
```

Function

f_getfd invokes the fcntl callable service to get the file descriptor flags for a file.

Parameters

fd The file descriptor (a number) for the file.

Usage Notes

The file descriptor flags are returned in RETVAL. The only POSIX-defined flag is FCTLCLOEXEC. To determine if this flag is set, use this expression:

```
(retval//2)=1
```

Example

To get the file descriptor flags for file descriptor `0`:

```
"f_getfd 0"
```

f_getfl

```
►►──f_getfl──fd──────────────────────────────────────────────────►◄
```

Function

f_getfl invokes the fcntl callable service to get the file status flags for a file.

Parameters

fd The file descriptor (a number) for the file.

Usage Notes

RETVAL returns the file status flags as a numeric value (see Appendix A):

Flag	Description
O_CREAT	Create the file if it does not exist.
O_EXCL	Fail if the file does exist and O_CREAT is set.
O_NOCTTY	Do not make this file a controlling terminal for the calling process.
O_TRUNC	If the file exists, truncate it to zero length.
O_APPEND	Set the offset to EOF before each write.
O_NONBLOCK	An open, read, or write on the file will not block (wait for terminal input).
O_RDWR	Open for read and write.
O_RDONLY	Open for read-only.
O_WRONLY	Open for write-only.
O_SYNC	Force synchronous updates.

You can use the open flags to test specific values. The easiest way to test a value is to convert both the RETVAL and the flags to binary data, and then logically AND them. For example, to test O_WRITE and O_TRUNC:

```
wrtr=D2C(O_WRITE+O_TRUNC,4))
If BITAND(D2C(retval,4),wrtr)=wrtr Then
   Do                                /* o_write and o_trunc are set */
   End
```

Example

In the following example, assume that *fd* was assigned a value earlier in the exec:

```
"f_getfl" fd
```

f_getlk

```
►►──f_getlk──fd──stem──────────────────────────────────────────►◄
```

Function

f_getlk invokes the fcntl callable service to return information on a file segment for which locks are set, cleared, or queried.

Parameters

fd The file descriptor (a number) for the file.

stem
The name of a stem variable that is the flock structure used to query, set, or clear a lock; or to return information. To specify the information, you can specify a numeric value (see Appendix A) or the predefined variables beginning with L_ used to derive the appropriate numeric value. For example, *stem.1* and *stem.l_type* are both the lock-type request:

Variable	Description
L_LEN	The length of the byte range to be set, cleared, or queried.
L_PID	The process ID of the process holding the blocking lock, if one was found.
L_START	The starting offset byte of the lock to be set, cleared, or queried.
L_TYPE	The type of lock being set, cleared, or queried. To specify the information, you can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value: **F_RDLCK** Shared or read lock. This type of lock specifies that the process can read the locked part of the file, and other processes cannot write on that part of the file in the meantime. A process can change a held write lock, or any part of it, to a read lock, thereby making it available for other processes to read. Multiple processes can have read locks on the same part of a file simultaneously. To establish a read lock, a process must have the file accessed for reading. **F_WRLCK** Exclusive or write lock. This type of lock indicates that the process can write on the locked part of the file, without interference from other processes. If one process puts a write lock on part of a file, no other process can establish a read lock or write lock on that same part of the file. A process cannot put a write lock on part of a file if there is already a read lock on an overlapping part of the file, unless that process is the only owner of that overlapping read lock. In such a case, the read lock on the overlapping section is replaced by the write lock being requested. To establish a write lock, a process must have the file accessed for writing.
L_TYPE (continued)	**F_UNLCK** Unlock. This is used to unlock all locks held on the given range by the requesting process.

Variable	Description
L_WHENCE	The flag for the starting offset. To specify the information, you can specify a numeric value (see Appendix A) or one of the predefined variables beginning with SEEK_ used to derive the appropriate numeric value. Valid values are: SEEK_CUR The current offset SEEK_END The end of the file SEEK_SET The specified offset

Example

In the following example, assume that *rec* was assigned a value earlier in the exec:

```
lock.l_len=40
lock.l_start=rec*40
lock.l_type=f_wrlck
lock.l_whence=seek_set
"f_getlk" fd "lock."
if lock.l_type=f_unlck then
 /* lock is available for the requested 40 byte record */
```

fork

There is no **fork** syscall command. Instead of using **fork**, see “spawn” on page 151.

forkexecm

```
►►──forkexecm──program──┬──────┬──────────────────────────────────►◄
                        └─parm─┘
```

Function

forkexecm invokes the fork and execmvs callable services to fork and exec a program to be executed from the LINKLIB, LPALIB, or STEPLIB library.

Parameters

program
The name of the program.

parm
A parameter to be passed to the program.

Usage Notes

1. If the exec fails, the child process ends with a **SIGABRT** signal.
2. If the **fork** succeeds, RETVAL contains the PID for the child process.
3. The child process has a unique process ID (PID) that does not match any active process group ID.
4. If a hierarchical file system (HFS) file has its FCTLCLOFORK flag set on, it is not inherited by the child process. This flag is set with the fcntl callable service.
5. The process and system utilization times for the child are set to zero.
6. Any file locks previously set by the parent are not inherited by the child.
7. The child process has no alarms set (similar to the results of a call to the alarm service with Wait_time specified as zero).
8. The child has no pending signals.
9. The following characteristics of the calling process are changed when the new executable program is given control by the execmvs callable service:
   - The prior process image is replaced with a new process image for the executable program to be run.
   - All open files marked close-on-exec and all open directory streams are closed.
   - All signals that have **sigaction** settings are reset to their default actions.
10. The input passed to the MVS executable file by the service is consistent with what is passed as input to MVS programs. On input, the MVS program receives a single-entry parameter list pointed to by register 1. The high-order bit of the single parameter entry is set to `1`.

    The single parameter entry is the address of a 2-byte length field followed by an argument string. The length field describes the length of the data that follows it. If a null argument and argument length are specified in the call, the length field specifies `0` bytes on input to the executable file.
11. The call can invoke both unauthorized and authorized MVS programs:

- Unauthorized programs receive control in problem program state, with PSW key 8.
- Authorized programs receive control in problem program state, with PSW key 8 and APF authorization.

12. The TASKLIB, STEPLIB, or JOBLIB DD data set allocations that are active for the calling task at the time of the call to the execmvs service are propagated to the new process image, if the data sets they represent are found to be cataloged. Uncataloged data sets are not propagated to the new process image. This causes the program that is invoked to run with the same MVS program search order as its invoker.

Example

In the following example, assume that *pgm* was assigned a value earlier in the exec:

```
"forkexecm (pgm) 'hello world'"
```

fpathconf

```
►►──fpathconf──fd──name──────────────────────────────────────────►◄
```

Function

fpathconf invokes the fpathconf callable service to let a program determine the current value of a configurable limit or variable associated with a file or directory. The value is returned in RETVAL.

Parameters

fd The file descriptor (a number) for the file or directory.

name
A numeric value that indicates which configurable limit will be returned. You can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value:

Variable	Description
PC_LINK_MAX	Maximum value of a file's link count.
PC_MAX_CANON	Maximum number of bytes in a terminal canonical input line.
PC_MAX_INPUT	Minimum number of bytes for which space will be available in a terminal input queue; therefore, the maximum number of bytes a portable application may require to be typed as input before reading them.
PC_NAME_MAX	Maximum number of bytes in a filename (not a string length; count excludes a terminating null).
PC_PATH_MAX	Maximum number of bytes in a pathname (not a string length; count excludes a terminating null).
PC_PIPE_BUF	Maximum number of bytes that can be written atomically when writing to a pipe.
PC_POSIX_CHOWN_RESTRICTED	Change ownership (“chown” on page 33) function is restricted to a process with appropriate privileges, and to changing the group ID (GID) of a file only to the effective group ID of the process or to one of its supplementary group IDs.
PC_POSIX_NO_TRUNC	Pathname components longer than 255 bytes generate an error.
PC_POSIX_VDISABLE	Terminal attributes maintained by the system can be disabled using this character value.

Usage Notes

1. If *name* refers to MAX_CANON, MAX_INPUT, or _POSIX_VDISABLE, then the following applies:
   - If *fd* does not refer to a terminal file, the function returns −1 and sets the ERRNO to EINVAL.
2. If *name* refers to NAME_MAX, PATH_MAX, or _POSIX_NO_TRUNC, the following applies:
   - If *fd* does not refer to a directory, the function still returns the requested information using the parent directory of the specified file.
3. If *name* refers to PC_PIPE_BUF, the following applies:

- If *fd* refers to a pipe or a FIFO, the value returned applies to the referred-to object itself. If *fd* refers to a directory, the value returned applies to any FIFOs that exist or that can be created within the directory. If *fd* refers to any other type of file, the function returns −1 in RETVAL and sets the ERRNO to EINVAL.

4. If *name* refers to PC_LINK_MAX, the following applies:
   - If *fd* refers to a directory, the value returned applies to the directory.

Example

To determine the maximum number of bytes that can be written atomically to the file identified by file descriptor 1:

```
"fpathconf 1 (pc_pipe_buf)"
```

f_setfd

```
►►──f_setfd──fd──close_exec──────────────────────────────────────►◄
```

Function

f_setfd invokes the fcntl callable service to set file descriptor flags.

Parameters

fd The file descriptor (a number) for the file.

close_exec
A numeric value to indicate whether this file descriptor should remain open after an exec:

- `0` indicates that it should remain open.
- 1 indicates that it should be closed.

Example

To set the flags for the file identified by file descriptor `0` and indicate that this file descriptor should remain open during an exec:

```
"f_setfd 0 0"
```

f_setfl

```
►►──f_setfl──fd──flags──────────────────────────────────────────►◄
```

Function

f_setfl invokes the fcntl callable service to set file status flags.

Parameters

fd The file descriptor (a number) for the file.

flags
A value that sets the file status flags. To specify the information, you can specify a numeric value (see Appendix A) or a predefined variable beginning with O_ used to derive the appropriate numeric value. The permitted values are:

Variable	Description
O_APPEND	Set offset to EOF on write.
O_NONBLOCK	Do not block an open, a read, or a write on the file (do not wait for terminal input).
O_SYNC	Force synchronous updates.

Example

To set the O_APPEND file status flag for the file identified by file descriptor 1:

```
"f_setfl 1" o_append
```

f_setlk

```
►►──f_setlk──fd──stem──────────────────────────────────────►◄
```

Function

f_setlk invokes the fcntl callable service to set or release a lock on part of a file.

Parameters

fd The file descriptor (a number) for the file.

stem
The name of a stem variable that is the flock structure used to set or release the lock. To access the information, you can specify a numeric value (see Appendix A) or the predefined variables beginning with L_ that derive the appropriate numeric value. For example, *stem.1* and *stem.l_type* are both the lock-type request.

Variable	Description
L_LEN	The length of the byte range to be set, cleared, or queried.
L_PID	The process ID of the process holding the blocking lock, if one was found.
L_START	The starting offset byte of the lock to be set, cleared, or queried.
L_TYPE	The type of lock being set, cleared, or queried. To specify the information, you can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value: **F_RDLCK** Shared or read lock. This type of lock specifies that the process can read the locked part of the file, and other processes cannot write on that part of the file in the meantime. A process can change a held write lock, or any part of it, to a read lock, thereby making it available for other processes to read. Multiple processes can have read locks on the same part of a file simultaneously. To establish a read lock, a process must have the file accessed for reading. **F_WRLCK** Exclusive or write lock. This type of lock indicates that the process can write on the locked part of the file, without interference from other processes. If one process puts a write lock on part of a file, no other process can establish a read lock or write lock on that same part of the file. A process cannot put a write lock on part of a file if there is already a read lock on an overlapping part of the file, unless that process is the only owner of that overlapping read lock. In such a case, the read lock on the overlapping section is replaced by the write lock being requested. To establish a write lock, a process must have the file accessed for writing.
L_TYPE (continued)	**F_UNLCK** Unlock. This is used to unlock all locks held on the given range by the requesting process.

Variable	Description
L_WHENCE	The flag for the starting offset. To specify the information, you can specify a numeric value (see Appendix A) or one of the predefined variables beginning with SEEK_ used to derive the appropriate numeric value. Valid values are:

SEEK_CUR	The current offset
SEEK_END	The end of the file
SEEK_SET	The specified offset

Usage Notes

If the lock cannot be obtained, a RETVAL of −1 is returned along with an appropriate ERRNO and ERRNOJR. You can also use F_SETLK to release locks already held, by setting *l_type* to F_UNLCK.

Multiple Lock Requests: A process can have several locks on a file simultaneously, but it can have only one type of lock set on any given byte. Therefore, if a process puts a new lock on part of a file that it had previously locked, the process has only one lock on that part of the file and the lock type is the one given by the most recent locking operation.

Releasing Locks: When an **f_setlk** or **f_setlkw** request is made to unlock a byte region of a file, all locks held by that process within the specified region are released. In other words, each byte specified on an unlock request is freed from any lock that is held against it by the requesting process.

All of a process's locks on a file are removed when the process closes a file descriptor for that file. Locks are not inherited by child processes created with the fork service.

Advisory Locking: All locks are advisory only. Processes can use locks to inform each other that they want to protect parts of a file, but locks do not prevent I/O on the locked parts. A process that has appropriate permissions on a file can perform whatever I/O it chooses, regardless of which locks are set. Therefore, file locking is only a convention, and it works only when all processes respect the convention.

Example

The following example locks a 40-byte record (*rec*). Assume that *rec* was assigned a value earlier in the exec:

```
lock.l_len=40
lock.l_start=rec*40
lock.l_type=f_wrlck
lock.l_whence=seek_set
"f_setlk (fd) lock."
```

f_setlkw

```
►►──f_setlkw──fd──stem──────────────────────────────────────────►◄
```

Function

f_setlkw invokes the fcntl callable service to set or release a lock on part of a file and, if another process has a lock on some or all of the requested range, wait until the specified range is free and the request can be completed.

fd The file descriptor (a number) for the file.

stem
The name of a stem variable that is the flock structure used to set or release the lock. To specify the lock information, you can specify a numeric value (see Appendix A) or the predefined variables beginning with L_ used to derive the appropriate numeric value. For example, *stem.1* and *stem.l_type* are both the lock-type request.

Variable	Description
L_LEN	The length of the byte range to be set, cleared, or queried.
L_PID	The process ID of the process holding the blocking lock, if one was found.
L_START	The starting offset byte of the lock to be set, cleared, or queried.
L_TYPE	The type of lock being set, cleared, or queried. To specify the information, you can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value: **F_RDLCK** Shared or read lock. This type of lock specifies that the process can read the locked part of the file, and other processes cannot write on that part of the file in the meantime. A process can change a held write lock, or any part of it, to a read lock, thereby making it available for other processes to read. Multiple processes can have read locks on the same part of a file simultaneously. To establish a read lock, a process must have the file accessed for reading. **F_WRLCK** Exclusive or write lock. This type of lock indicates that the process can write on the locked part of the file, without interference from other processes. If one process puts a write lock on part of a file, no other process can establish a read lock or write lock on that same part of the file. A process cannot put a write lock on part of a file if there is already a read lock on an overlapping part of the file, unless that process is the only owner of that overlapping read lock. In such a case, the read lock on the overlapping section is replaced by the write lock being requested. To establish a write lock, a process must have the file accessed for writing.
L_TYPE (continued)	**F_UNLCK** Unlock. This is used to unlock all locks held on the given range by the requesting process.

Variable	Description
L_WHENCE	The flag for the starting offset. To specify the information, you can specify a numeric value (see Appendix A) or one of the predefined variables beginning with SEEK_ used to derive the appropriate numeric value. Valid values are:

SEEK_CUR	The current offset
SEEK_END	The end of the file
SEEK_SET	The specified offset

Usage Notes

If the lock cannot be obtained because another process has a lock on all or part of the requested range, the **f_setlkw** request waits until the specified range becomes free and the request can be completed. You can also use **f_setlkw** to release locks already held, by setting *l_type* to F_UNLCK.

If a signal interrupts a call to the fcntl service while it is waiting in an **f_setlkw** operation, the function returns with a RETVAL of −1 and an ERRNO of EINTR.

f_setlkw operations have the potential for encountering deadlocks. This happens when process A is waiting for process B to unlock a region, and B is waiting for A to unlock a different region. If the system detects that a **f_setlkw** might cause a deadlock, the fcntl service returns with a RETVAL of −1 and an ERRNO of EDEADLK.

See “f_setlk” on page 63 for more information about locks.

Example

The following example locks a 40-byte record (*rec*). Assume that *rec* was assigned a value earlier in the exec:

```
lock.l_len=40
lock.l_start=rec*40
lock.l_type=f_wrlck
lock.l_whence=seek_set
"f_setlkw (fd) lock."
```

fstat

```
►►──fstat──fd──stem──────────────────────────────────────────────►◄
```

Function

fstat invokes the fstat callable service to get status information about a file that is identified by its file descriptor.

Parameters

fd The file descriptor (a number) for the file.

stem
The name of a stem variable used to return the status information. Upon return, *stem.0* contains the number of variables that are returned. To access the status values, you can use a numeric value (see Appendix A) or any of the predefined variables that begin with ST_. For example, *stem.9* and *stem.st_atime* are both the request for the time of last access.

Variable	Description
ST_AAUDIT	Auditor audit information.
ST_ATIME	Time of last access.
ST_AUDITID	RACF File ID for auditing.
ST_BLKSIZE	File block size.
ST_BLOCKS	Blocks allocated.
ST_CCSID	Coded character set ID.
ST_CRTIME	File creation time.
ST_CTIME	Time of last file status change.
ST_DEV	Device ID of the file.
ST_EXTLINK	External symbolic link flag, set to `0` or 1.
ST_FID	File identifier.
ST_FILEFMT	Format of the file. To specify the format, you can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value:

S_FFBINARY	Binary data
S_FFCR	Text data delimited by a carriage return character
S_FFCRLF	Text data delimited by carriage return and line feed characters
S_FFLF	Text data delimited by a line feed character
S_FFLFCR	Text data delimited by a line feed and carriage return characters
S_FFNA	Text data with the file format not specified
S_FFNL	Text data delimited by a newline character

Variable	Description
ST_GENVALUE	General attribute values.
ST_GID	Group ID of the group of the file.
ST_INO	File serial number.
ST_MAJOR	Major number for a character special file.
ST_MINOR	Minor number for a character special file.
ST_MODE	File mode, permission bits only.
ST_MTIME	Time of last data modification.
ST_NLINK	Number of links.
ST_RTIME	File backup time stamp (reference time).
ST_SETGID	Set Group ID on execution flag, set to `0` or 1.

Variable	Description
ST_SETUID	Set User ID on execution flag, set to `0` or 1.
ST_SIZE	File size for a regular file, in bytes. If file size exceeds $2^{31}-1$ bytes, size is expressed in megabytes, using an M (for example, 3123M).
ST_STICKY	Sticky bit flag (keep loaded executable in storage), set to `0` or 1.
ST_TYPE	Numeric value that represents the file type for this file. You can use a numeric value (see Appendix A) or any of the predefined variables that begin with S_ to determine the file type: S_ISCHR Character special file S_ISDIR Directory S_ISFIFO FIFO special file S_ISREG Regular file S_ISSYM Symbolic link
ST_UAUDIT	Area for user audit information.
ST_UID	User ID of the owner of the file.

Usage Notes

All time values returned in *stem* are in POSIX format. Time is in seconds since 00:00:00 GMT, January 1, 1970. You can use **gmtime** to convert it to other forms.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec:

```
"fstat (fd) st."
```

fstatvfs

```
►►──fstatvfs──fd──stem──────────────────────────────────────────►◄
```

Function

fstatvfs invokes the fstatvfs callable service to obtain information about a file system identified by a file descriptor that refers to a file in that file system.

Parameters

fd A file descriptor referring to a file from the desired file system.

stem
The name of a stem variable used to return the information. On return, *stem.0* contains the number of variables returned. You can use the predefined variables beginning with STFS_ or their equivalent numeric values to access the status values they represent. (See Appendix A for the numeric values.) For example, *stem.stfs_avail* accesses the number of blocks available in the file system.

Variable	Description
STFS_AVAIL	Space available to unprivileged users in block-size units.
STFS_BFREE	Total number of free blocks.
STFS_BLOCKSIZE	Block size.
STFS_FAVAIL	Number of free file nodes available to unprivileged users.
STFS_FFREE	Total number of free file nodes.
STFS_FILES	Total number of file nodes in the file system.
STFS_FRSIZE	Fundamental file system block size.
STFS_FSID	File system ID set by the logical file system.
STFS_INUSE	Allocated space in block-size units.
STFS_INVARSEC	Number of seconds the file system will remain unchanged.
STFS_NAMEMAX	Maximum length of file name.
STFS_NOSUID	SETUID and SETGID are not supported.
STFS_RDONLY	File system is read-only.
STFS_TOTAL	Total space in block-size units.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec:

```
"fstatvfs" fd st.
```

fsync

```
►►──fsync──fd──────────────────────────────────────────────►◄
```

Function

fsync invokes the fsync callable service to write changes on the direct access storage device that holds the file identified by the file descriptor.

Parameters

fd The file descriptor (a number) for the file.

Usage Notes

On return from a successful call, all updates have been saved on the direct access storage that holds the file.

Example

To invoke **fsync** for file descriptor 1:

```
"fsync 1"
```

ftrunc

```
►►──ftrunc──fd──file_size──────────────────────────────────────►◄
```

Function

ftrunc invokes the ftrunc callable service to change the size of the file identified by the file descriptor.

Parameters

fd The file descriptor (a number) for the file.

file_size
The new size of the file, in bytes.

Usage Notes

1. The ftrunc service changes the file size to *file_size* bytes, beginning at the first byte of the file. If the file was previously larger than *file_size*, all data from *file_size* to the original end of the file is removed. If the file was previously shorter than *file_size*, bytes between the old and new lengths are read as zeros.
2. Full blocks are returned to the file system so that they can be used again, and the file size is changed to the lesser of *file_size* or the current length of the file.
3. The file offset is not changed.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec. To truncate *fd* to `0` bytes:

```
"ftrunc" fd 0
```

getcwd

```
►►──getcwd──variable──────────────────────────────────────────►◄
```

Function

getcwd invokes the getcwd callable service to get the pathname of the working directory.

Parameters

variable
The name of the variable where the pathname of the working directory is to be stored.

Example

To get the pathname of the working directory:

```
"getcwd cwd"
```

getegid

```
►►──getegid──────────────────────────────────────────────────────►◄
```

Function

getegid invokes the getegid callable service to get the effective group ID (GID) of the calling process.

Usage Notes

1. The effective GID is returned in RETVAL.
2. If the service fails, the process abends.

geteuid

```
►►──geteuid─────────────────────────────────────────────────►◄
```

Function

geteuid invokes the geteuid callable service to get the effective user ID (UID) of the calling process.

Usage Notes

1. The effective UID is returned in RETVAL.
2. If the service fails, the process abends.

getgid

```
►►──getgid──────────────────────────────────────────────────►◄
```

Function

getgid invokes the getgid callable service to get the real group ID (GID) of the calling process.

Usage Notes

1. The GID is returned in RETVAL.
2. If the service fails, the process abends.

getgrent

```
►►──getgrent──stem──────────────────────────────────────────────►◄
```

Function

getgrent invokes the getgrent callable service to retrieve a group database entry.

stem
The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To access the status values, you can specify a numeric value (see Appendix A) or the predefined variable beginning with GR_ used to derive the appropriate numeric value. For example, you can specify *stem.2* or *stem.gr_gid* to access the group ID:

Variable	**Description**
GR_GID	The group ID.
GR_MEM	The stem index of the first member name returned.
GR_MEMBERS	The number of members returned.
GR_NAME	The name of the group.

Usage Notes

1. The service is intended to be used to search the group database sequentially. The first call to this service from a given task returns the first group entry in the group database. Subsequent calls from the same task return the next group entry found, until no more entries exist, at which time a RETVAL of `0` is returned.
2. The setgrent service can be used to reset this sequential search.

Example

To list all groups in the group data base:

```
do forever
   "getgrent gr."
   if retval=0 | retval=-1 then
      leave
   say gr.gr_name
end
```

getgrid

```
►►──getgrgid──gid──stem──────────────────────────────────►◄
```

Function

getgrgid invokes the getgrgid callable service to get information about a group and its members; the group is identified by its group ID (GID).

Parameters

gid

A numeric value.

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To access the status values, you can specify a numeric value (see Appendix A) or the predefined variable beginning with GR_ used to derive the appropriate numeric value. For example, you can specify *stem.2* or *stem.gr_gid* to access the group ID:

Variable	Description
GR_GID	The group ID.
GR_MEM	The stem index of the first member name returned.
GR_MEMBERS	The number of members returned.
GR_NAME	The name of the group.

Usage Notes

A RETVAL greater than zero indicates success. A RETVAL of −1 or `0` indicates failure. A RETVAL of `0` has an ERRNOJR, but no ERRNO.

Example

In the following example, assume that *gid* was assigned a value earlier in the exec. To get a list of names connected with a group ID:

```
"getgrgid (gid) gr."
say 'Users connected to group number' gid ':'
do i=gr_mem to gr.0
   say gr.i
end
```

getgrnam

```
►►──getgrnam──name──stem────────────────────────────────────►◄
```

Function

getgrnam invokes the getgrnam callable service to get information about a group and its members. The group is identified by its name.

Parameters

name
A string that specifies the group name as defined to the system.

stem
The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To access the status values, you can specify a numeric value (see Appendix A) or the predefined variable beginning with GR_ used to derive the appropriate numeric value. For example, you can specify *stem.2* or *stem.gr_gid* to access the group ID:

Variable	Description
GR_GID	The group ID.
GR_MEM	The stem index of the first member name returned.
GR_MEMBERS	The number of members returned.
GR_NAME	The name of the group.

Usage Notes

1. A RETVAL greater than zero indicates success. A RETVAL of −1 or `0` indicates failure. A RETVAL of `0` has an ERRNOJR, but no ERRNO.
2. The return values point to data that can change or disappear after the next getgrnam or getgrgid call from that task. Each task manages its own storage separately. Move data to your own dynamic storage if you need it for future reference.
3. The storage is key `0` non-fetch-protected storage that is managed by OS/390 OpenEdition.

Example

To get information about the group named SYS1:

```
"getgrnam SYS1 gr."
say 'Users connected to group SYS1:'
do i=gr_mem to gr.0
   say gr.i
end
```

getgroups

```
►►──getgroups──stem──────────────────────────────────────────►◄
```

Function

getgroups invokes the getgroups callable service to get the number of supplementary group IDs (GIDs) for the calling process and a list of those supplementary group IDs.

Parameters

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. *stem.1* to *stem.n* (where *n* is the number of variables returned) each contain a group ID.

Usage Notes

A RETVAL greater than zero indicates success. A RETVAL of −1 or `0` indicates failure. A RETVAL of `0` has an ERRNOJR, but no ERRNO.

Example

To invoke getgroups:

```
"getgroups grps."
```

getgroupsbyname

```
►►──getgroupsbyname──name──stem──────────────────────────────────►◄
```

Function

getgroupsbyname invokes the getgroupsbyname callable service to get the number of supplementary group IDs (GIDs) for a specified user name and, optionally, get a list of those supplementary group IDs.

Parameters

name
A string that specifies the name of the user as defined to the system.

stem
The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. *stem.1* to *stem.n* (where *n* is the number of variables returned) each contain a supplementary group ID for *name*.

Usage Notes

A RETVAL greater than zero indicates success. A RETVAL of −1 or `0` indicates failure. A RETVAL of `0` has an ERRNOJR, but no ERRNO.

Example

To get the number of supplementary group IDs for the user MEGA:

```
"getgroupsbyname MEGA supgrp."
```

getlogin

```
►►──getlogin──variable──────────────────────────────────────────►◄
```

Function

getlogin invokes the getlogin callable service to get the user login name associated with the calling process.

Parameters

variable
The name of the variable in which the login name is returned.

Usage Notes

If the service fails, the process abends.

Example

To invoke getlogin and store the login name in the variable *myid*:

```
"getlogin myid"
```

getment

```
►►──getment──length──variable──────────────────────────────────►◄
```

Function

getment invokes the w_getmntent callable service to get information about the mounted file systems and return the information in the format used by OS/390 OpenEdition callable services. Alternatively, you can use the getmntent syscall command to format the mount entries in a stem.

Parameters

length
The size of the specified variable. If the length is `0`, RETVAL contains the total number of mounted file systems; otherwise, RETVAL contains the number of entries returned.

variable
The name of the buffer where the information about the mount entries is to be stored. Clear the buffer on the first call and do not alter it between calls.

Usage Notes

1. Before a program calls **getment** for the first time, the variable should be dropped, set to blanks, or set to nulls.
2. If more than one call is made to **getment**, use the same variable on each call, because part of the information returned in the variable tells the file system where to continue retrieving its information.
3. **getment** normally returns information about as many file systems as are mounted, or as many as fit in the passed variable. The number of entries contained in the variable is returned. The caller must have a variable large enough to receive information about at least a single mount entry with each call. If a zero-length variable is passed, no information is returned, but the return value contains the total number of mounted file systems. This value could then be used to get enough storage to retrieve information on all these file systems in one additional call.
4. You could also retrieve all mount entries by setting up a loop that continues to call **getment** until a return value of either `−1` (in an error) or `0` (no more entries found) is returned.

Example

In the following example, assume that *buf* was assigned a value earlier in the exec. This example returns the number of mounted file systems in RETVAL:

```
"getment 0 buf"
```

getmntent

```
►►──getmntent──stem──┬───────┬──────────────────────────────────►◄
                     └─devno─┘
```

Function

getmntent invokes the w_getmntent callable service to get information about the mounted file systems, or a specific mounted file system, and return the information formatted in a stem.

Parameters

stem

The name of a stem variable used to return the mount table information. The stem has two dimensions: the field name, followed by a period and the number of the mount table entry. For example, you can access the file system name for the first entry in the mount table as *stem.mnte_fsname.1*.

For the field name, you can specify a numeric value (see Appendix A) or the predefined variable beginning with MNTE_ used to derive the appropriate numeric value. For example, you could use *stem.mnte_fsname.1* or *stem.6.1* to access the file system name for the first entry:

Variable	Description
MNTE_DD	The ddname specified on the mount.
MNTE_DEV	The device ID of the file system.
MNTE_FSNAME	The name of the HFS data set containing the file system.
MNTE_FSTYPE	The file system type; for example, HFS.
MNTE_MODE	The file system type mount method. You can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value: MNT_MODE_RDWR MNT_MODE_RDONLY
MNTE_PARDEV	The ST_DEV of the parent file system.
MNTE_PARM	The parameter specified with **mount()**.
MNTE_PATH	The mountpoint pathname.
MNTE_QJOBNAME	The job name of the quiesce requestor.
MNTE_QPID	The PID of the quiesce requestor.
MNTE_ROOTINO	The inode of the mountpoint.

Variable	Description	
MNTE_STATUS	The status of the file system. To specify the information, you can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value:	
	MNT_ASYNCHMOUNT	Asynchronous mount in progress for this file system.
	MNT_FILEACTIVE	File system is active.
	MNT_FILEDEAD	File system is dead.
	MNT_FILEDRAIN	File system is being unmounted with the drain option.
	MNT_FILEFORCE	File system is being unmounted with the force option.
	MNT_FILEIMMED	File system is being unmounted with the immediate option.
	MNT_FILENORM	File system is being unmounted with the normal option.
	MNT_FILERESET	File system is being reset.
	MNT_IMMEDTRIED	File system unmount with the immediate option failed.
	MNT_MOUNTINPROGRESS	Mount in progress for this file system
	MNT_QUIESCED	File system is quiesced.
MNTE_TYPE	The file system type.	

devno
An optional parameter, this is the device number for a specific file system for which you want the mount information. Specifying `0` is the equivalent of not specifying a device number.

Example

To invoke w_getmntent:

```
"getmntent mounts."
```

getpgrp

```
►►──getpgrp──────────────────────────────────────────────►◄
```

Function

getpgrp invokes the getpgrp callable service to get the process group ID (PGID) of the calling process.

Usage Notes

1. The PGID for the calling process is returned in RETVAL.
2. If the service fails, the process abends.

getpid

```
►►──getpid──────────────────────────────────────────────────────►◄
```

Function

getpid invokes the getpid callable service to get the process ID (PID) of the calling process.

Usage Notes

1. The PID for the calling process is returned in RETVAL.
2. If the service fails, the process abends.

getppid

```
►►──getppid──────────────────────────────────────────────────────►◄
```

Function

getppid invokes the getppid callable service to get the parent process ID (PPID) of the calling process.

Usage Notes

1. The parent PID for the calling process is returned in RETVAL.
2. If the service fails, the process abends.

getpsent

```
►►──getpsent──stem──────────────────────────────────────────────►◄
```

Function

getpsent invokes the w_getpsent callable service to provide data describing the status of all the processes to which you are authorized. The data is formatted in a stem. The PS_ variables, or their numeric equivalents, are used to access the fields.

Parameters

stem

Upon return, *stem.0* contains the number of processes for which information is returned. *stem.1* through *stem.n* (where *n* is the number of entries returned) each contain process informati on for the *n*th process.

You can use the predefined variables that begin with PS_ or their equivalent numeric values (see Appendix A) to access the information. For example, *stem.1.ps_pid* is the process ID for the first process returned.

Variable	Description
PS_CMD	Command.
PS_CONTTY	Controlling tty.
PS_EGID	Effective group ID.
PS_EUID	Effective user ID.
PS_FGPID	Foreground process group ID.
PS_MAXVNODES	Maximum number of vnode tokens allowed.
PS_PATH	Pathname.
PS_PGPID	Process Group ID.
PS_PID	Process ID.
PS_PPID	Parent Process ID.
PS_RGID	Real group ID.
PS_RUID	Real user ID.
PS_SERVERFLAGS	Server flags.
PS_SERVERNAME	Server name supplied on registration.
PS_SERVERTYPE	Server type (File=1; Lock=2).
PS_SGID	Saved set group ID.
PS_SID	Session ID (leader).
PS_SIZE	Total size.
PS_STARTTIME	Starting time, in POSIX format (seconds since the Epoch, 00:00:00 on 1 January 1970).
PS_STAT	Process status.

Variable	Description
PS_STATE	Process state. This value can be expressed as one of the following predefined variables or as an alphabetic value (see Appendix A):
	PS_CHILD — Waiting for a child process.
	PS_FORK — fork() a new process.
	PS_FREEZE — QUIESCEFREEZE
	PS_MSGRCV — IPC MSGRCV WAIT
	PS_MSGSND — IPC MSGSND WAIT
	PS_PAUSE — MVSPAUSE
	PS_QUIESCE — Quiesce termination wait.
	PS_RUN — Running, not in kernel wait.
	PS_SEMWT — IPC SEMOP WAIT
	PS_SLEEP — sleep() issued.
	PS_WAITC — Communication kernel wait.
	PS_WAITF — File system kernel wait.
	PS_WAITO — Other kernel wait.
	PS_ZOMBIE — Process cancelled.
	PS_ZOMBIE2 — Process terminated yet still the session or process group leader.
PS_SUID	Saved set user ID.
PS_SYSTIME	System CPU time, a value of the type clock_t, which needs to be divided by sysconf(_SC_CLK_TK) to convert it to seconds. For OS/390 OpenEdition, this value is expressed in hundredths of a second.
PS_USERTIME	User CPU time, a value of the type clock_t, which needs to be divided by sysconf(_SC_CLK_TK) to convert it to seconds. For OS/390 OpenEdition, this value is expressed in hundredths of a second.
PS_VNODECOUNT	Current number of vnode tokens.

Usage Notes

1. Information is returned for only those processes for which RACF allows the user access based on effective user ID, real user ID, or saved set user ID.
2. PS_STARTTIME is in seconds since the Epoch (00:00:00 on 1 January 1970).
3. PS_USERTIME and PS_SYSTIME are task-elapsed times in 1/100ths of seconds.

Example

This exec will produce output similar to the **ps -A** shell command, displaying information on all accessible processes:

```
/* rexx */
address syscall
say right('PID',12) left('TTY',10) '    TIME' 'COMMAND'
ps.0=0
'getpsent ps.'                          /* get process data           */
do i=1 to ps.0                          /* process each entry returned */
   t=(ps.i.ps_usertime + 50) % 100      /* change time to seconds     */
   'gmtime (t) gm.'                     /* convert to usable format   */
   if gm.tm_hour=0 then                 /* set hours: samp ignores day */
      h='   '
    else
      h=right(gm.tm_hour,2,0)':'
   m=right(gm.tm_min,2,0)':'            /* set minutes                */
   parse value reverse(ps.i.ps_contty),
         with tty '/'                   /* get tty filename           */
   tty=reverse(tty)
   say right(ps.i.ps_pid,12),           /* display process id         */
   say right(ps.i.ps_pid,12),           /* display process id         */
       left(tty,10),                    /* display controlling tty    */
       h || m || right(gm.tm_sec,2,0),  /* display process time       */
       ps.i.ps_cmd                      /* display command            */
end
return 0
```

getpwent

```
►►──getpwent──stem──────────────────────────────────────────────►◄
```

Function

getpwent invokes the getpwent callable service to retrieve a user database entry.

Parameters

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. You can use a numeric value (see Appendix A) or the predefined variables beginning with PW_ to access the values:

Variable	Description
PW_DIR	The initial working directory.
PW_GID	The group ID.
PW_NAME	The TSO/E user ID.
PW_SHELL	The name of the initial user program.
PW_UID	The user ID (UID) as defined to RACF.

Example

To list all users in the user database:

```
do forever
   "getpwent pw."
   if retval=0 | retval=-1 then
      leave
   say pw.pw_name
end
```

getpwnam

```
►►──getpwnam──name──stem──────────────────────────────────────────►◄
```

Function

getpwnam invokes the getpwnam callable service to get information about a user, identified by user name.

Parameters

name
The user name as defined to the system.

stem
The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To access the information, you can specify a numeric value (see Appendix A) or the predefined variable beginning with PW_ used to derive the appropriate numeric value. For example, to access the name of the user's initial working directory, you can specify *stem.4* or *stem.pw_dir*.

Variable	Description
PW_DIR	The initial working directory.
PW_GID	The group ID.
PW_NAME	The TSO/E user ID.
PW_SHELL	The name of the initial user program.
PW_UID	The user ID (UID) as defined to RACF.

Usage Notes

- A RETVAL greater than zero indicates success. A RETVAL of –1 or `0` indicates failure. A RETVAL of `0` has an ERRNOJR, but no ERRNO.
- If an entry for the specified *name* is not found in the user database, the RETVAL is `0`.

Example

To get information about the user JANET:

```
"getpwnam JANET pw."
```

getpwuid

```
►►──getpwuid──uid──stem──────────────────────────────────────────►◄
```

Function

getpwuid invokes the getpwuid callable service to get information about a user, identified by UID.

Parameters

uid

A numeric value that is the user's UID as defined to the system.

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. You can use a numeric value (see Appendix A) or the predefined variables beginning with PW_ to access the values:

Variable	Description
PW_DIR	The initial working directory.
PW_GID	The group ID.
PW_NAME	The TSO/E user ID.
PW_SHELL	The name of the initial user program.
PW_UID	The user ID (UID) as defined to RACF.

Usage Notes

A RETVAL greater than zero indicates success. A RETVAL of −1 or `0` indicates failure. A RETVAL of `0` has an ERRNOJR, but no ERRNO.

Example

To get information about the user with UID 42:

```
"getpwuid 42 pw."
```

getrlimit

```
►►──getrlimit──resource──stem──────────────────────────────────────►◄
```

Function

getrlimit invokes the getrlimit callable service to get the maximum and current resource limits for the calling process.

Parameters

resource

The resource whose limit is being requested. Resources that have limits with values greater than RLIM_INFINITY will return values of RLIM_INFINITY.

You can use the predefined variables beginning with RLIMIT_ or their equivalent numeric values to access the limits. (See Appendix A for the numeric values.)

Variable	Description
RLIMIT_AS	Maximum address space size for a process.
RLIMIT_CORE	Maximum size (in bytes) of a core dump created by a process.
RLIMIT_CPU	Maximum amount of CPU time (in seconds) used by a process.
RLIMIT_FSIZE	Maximum files size (in bytes) created by a process.
RLIMIT_NOFILE	Maximum number of open file descriptors for a process.

stem

The name of the stem variable used to return the limit. *stem.1* is the first word and *stem.2* is the second word. The first word contains the current limit; the second word contains the maximum limit. The values for each word are dependent on the *resource* specified.

Example

To print the maximum and current limit for number of open files allowed:

```
"getrlimit" rlimit_nofile r.
say 'maximum open limit is' r.2
say 'current open limit is' r.1
```

getuid

```
►►──getuid──────────────────────────────────────────────────►◄
```

Function

getuid invokes the getuid callable service to get the real user ID of the calling process.

Usage Notes

Upon return, RETVAL contains the UID. If the service fails, the process abends.

gmtime

```
►►──gmtime──time──stem──────────────────────────────────────────►◄
```

Function

gmtime converts time expressed in seconds since Epoch into month, day, and year time format.

Parameters

time

A numeric value, the time expressed as “POSIX time,” the number of seconds since the Epoch (00:00:00 on 1 January 1970).

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To access the status values, you can specify a numeric value (see Appendix A) or the predefined variable beginning with TM_ used to derive the appropriate numeric value. For example, to access the year, you can specify *stem.6* or *stem.tm_year*:

Variable	Description
TM_HOUR	The hour of the day.
TM_ISDST	The daylight saving time flag. This flag is always zero (−1) for Greenwich Mean Time (GMT).
TM_MDAY	The day of the month, 1 to 31.
TM_MIN	The minutes after the hour, 0 to 59.
TM_MON	The months since January, 0 to 11.
TM_SEC	The seconds after the minute, 0 to 59.
TM_WDAY	The days since Sunday, 0 to 6.
TM_YDAY	The days since January 1, 0 to 365.
TM_YEAR	The year.

Example

The *st_ctime* in the following example was set with a stat call:

```
"gmtime" st.st_ctime "tm."
```

ioctl

```
►►──ioctl──fd──command──buffer──┬────────┬────────────────────►◄
                                 └─length─┘
```

Function

ioctl invokes the w_ioctl service to issue ioctl commands to a file.

Note: Hierarchical file system (HFS) files and pipes do not support ioctl commands.

Parameters

fd A file descriptor for an open file.

Note: REXX does not support ioctl commands for sockets.

command
The numeric value for an ioctl command. The commands that can be specified vary by device and are defined by the device driver.

buffer
The name of a buffer containing the argument to be passed to the device driver. The argument is limited to 1024 bytes.

length
An optional numeric value indicating the length of the argument. If length is not specified, the length of the buffer is used.

isatty

```
►►──isatty──fd────────────────────────────────────────────────────►◄
```

Function

isatty invokes the isatty callable service to determine if a file is a terminal.

Parameters

fd The file descriptor (a number) for the file.

Usage Notes

On return, RETVAL contains either `0` (not a tty) or 1 (a tty).

Example

To test if file descriptor `0` is a terminal:

```
"isatty 0"
```

kill

```
►►──kill──pid──signal──────────────────────────────────────►◄
```

Function

kill invokes the kill callable service to send a signal to a process or process group.

Parameters

pid
A number, the process ID of the process or process group to which the caller wants to send a signal.

signal
The signal to be sent. You can set the value by using a numeric value (see Appendix A) or the predefined variable beginning with SIG used to derive the numeric value:

Variable	Description
SIGABND	Abend
SIGABRT	Abnormal termination
SIGALRM	Timeout
SIGBUS	Bus error
SIGCHLD	Child process terminated or stopped
SIGCONT	Continue if stopped
SIGDCE	Exclusive use by DCE
SIGFPE	Erroneous arithmetic operation, such as division by zero or an operation resulting in an overflow
SIGHUP	Hangup detected on the controlling terminal
SIGILL	Termination (cannot be caught or ignored)
SIGINT	Interactive attention
SIGIO	Completion of input or output
SIGIOERR	Error on input/output; used by the C runtime library
SIGKILL	Termination that cannot be caught or ignored
SIGPIPE	Write on a pipe with no readers
SIGPOLL	Pollable event
SIGPROF	Profiling timer expired
SIGQUIT	Interactive termination
SIGSEGV	Detection of an incorrect memory reference
SIGSTOP	Stop that cannot be caught or ignored
SIGSYS	Bad system call
SIGTERM	Termination
SIGTRAP	Trap used by the ptrace callable service
SIGTSTP	Interactive stop

Variable	Description
SIGTTIN	Read from a controlling terminal attempted by a member of a background process group
SIGTTOU	Write from a controlling terminal attempted by a member of a background process group
SIGURG	High bandwidth data is available at a socket
SIGUSR1	Reserved as application-defined signal 1
SIGUSR2	Reserved as application-defined signal 2
SIGVTALRM	Virtual timer expired
SIGXCPU	CPU time limit exceeded
SIGXFSZ	File size limit exceeded

Usage Notes

1. A caller can send a signal if the real or effective user ID of the caller is the same as the real or saved set user ID of the intended recipient. A caller can also send signals if it has appropriate privileges.
2. Regardless of user ID, a caller can always send a **SIGCONT** signal to a process that is a member of the same session as the sender.
3. A caller can also send a signal to itself. If the signal is not blocked, at least one pending unblocked signal is delivered to the sender before the service returns control. Provided that no other unblocked signals are pending, the signal delivered is the signal sent.

See “Using the REXX Signal Services” on page 8 for information on using signal services.

Example

In the following example, assume that *pid* was assigned a value earlier in the exec:

```
"kill" pid sighup
```

lchown

```
►►──lchown──pathname──uid──gid──────────────────────────────────►◄
```

Function

lchown invokes the lchown callable service to change the owner or group for a file, directory, or symbolic link.

Parameters

pathname
The pathname for a file, directory, or symbolic link.

uid
The numeric UID for the new owner or the present UID, or −1 if there is no change to the UID.

GID
The numeric GID for the new group or the present GID, or −1 if there is no change to the GID.

Usage Notes

1. If lchown's target is a symbolic link, it modifies the ownership of the actual symbolic link file instead of the ownership of the file pointed to by the symbolic link.
2. The lchown service changes the owner UID and owner GID of a file. Only a superuser can change the owner UID of a file.
3. The owner GID of a file can be changed by a superuser, or if a caller meets all of these conditions:
   - The effective UID of the caller matches the file's owner UID.
   - The *uid* value specified in the change request matches the file's owner UID.
   - The *gid* value specified in the change request is the effective GID, or one of the supplementary GIDs, of the caller.
4. The set-user-ID-on-execution and set-group-ID-on-execution permissions of the file mode are automatically turned off.
5. If the change request is successful, the change time for the file is updated.
6. Values for both *uid* and *gid* must be specified as they are to be set. If you want to change only one of these values, the other must be set to its present value to remain unchanged.

Example

In the following example, assume that *pathname*, *uid*, and *gid* were assigned a value earlier in the exec:

```
"lchown (pathname) (uid) (gid)"
```

link

```
►►──link──old_path──new_path──────────────────────────────────►◄
```

Function

link invokes the link callable service to create a hard link to a file (not a directory); that is, it creates a new name for an existing file. The new name does not replace the old one.

Parameters

old_path
A pathname, the current name for the file.

new_path
A pathname, the new name for the file.

Usage Notes

1. The link service creates a link named *new_path* to an existing file named *old_path*. This provides an alternate pathname for the existing file, so that the file can be accessed by the old name or the new name. The link can be stored in the same directory as the original file, or in a different directory.
2. The link and the file must be in the same file system.
3. If the link is created successfully, the service increments the link count of the file. The link count shows how many links exist for a file. If the link is not created successfully, the link count is not incremented.
4. Links are allowed only to files, not to directories.
5. If the link is created successfully, the change time of the linked-to file is updated and the change and modification times of the directory that holds the link are updated.

Example

To create the hard link **/usr/bin/grep** to the file **/bin/grep**:

```
"link /bin/grep /usr/bin/grep"
```

lseek

```
►►──lseek──fd──position──whence──────────────────────────────────►◄
```

Function

lseek invokes the lseek callable service to change the file offset of a file to a new position. The file offset is the position in a file from which data is next read or to which data is next written.

Parameters

fd The file descriptor (a number) for the file whose offset you want to change. The file descriptor is returned when the file is opened.

position
A number indicating the number of bytes by which you want to change the offset. If the number is unsigned, the offset is moved forward that number of bytes; if the number is preceded by a – (minus sign), the offset is moved backward that number of bytes.

whence
A numeric value that indicates the point from which the offset is calculated. You can specify a numeric value (see Appendix A) or the predefined variable beginning with SEEK_ used to derive the appropriate numeric value:

Variable	Description
SEEK_CUR	Set the file offset to current offset plus the specified offset.
SEEK_END	Set the file offset to EOF plus the specified offset.
SEEK_SET	Set the file offset to the specified offset.

Usage Notes

1. *position* gives the length and direction of the offset change. *whence* states where the change is to start. For example, assume that a file is 2000 bytes long, and that the current file offset is 1000:

Position Specified	Whence	New File Offset
80	SEEK_CUR	1080
1200	SEEK_SET	1200
–80	SEEK_END	1920
132	SEEK_END	2132

2. The file offset can be moved beyond the end of the file. If data is written at the new file offset, there will be a gap between the old end of the file and the start of the new data. A request to read data from anywhere within that gap completes successfully, and returns bytes with the value of zero in the buffer and the actual number of bytes read.

   Seeking alone, however, does not extend the file. Only if data is written at the new offset does the length of the file change.

Example

To change the offset of file descriptor *fd* (assuming that it was assigned a value earlier in the exec) to the beginning of the file (offset `0`):

```
"lseek" fd 0 seek_set
```

lstat

```
►►──lstat──pathname──stem──────────────────────────────────────►◄
```

Function

lstat invokes the lstat callable service to obtain status information about a file.

Parameters

pathname
A pathname for the file.

stem
The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To obtain the desired status information, you can use a numeric value (see Appendix A) or the predefined variables beginning with ST_ used to derive the numeric value. See “fstat” on page 67 for a list of those variables.

Usage Notes

1. If the pathname specified is a symbolic link, the status information returned relates to the symbolic link, rather than to the file to which the symbolic link refers.
2. All time values returned are in POSIX format. Time is in seconds since 00:00:00 GMT, January 1, 1970. You can use **gmtime** to convert it to other forms.

Example

In the following example, assume that *pathname* was assigned a value earlier in the exec:

```
"lstat (pathname) st."
```

mkdir

```
►►──mkdir──pathname──mode──────────────────────────────────────►◄
```

Function

mkdir invokes the mkdir callable service to create a new, empty directory.

Parameters

pathname
A pathname for the directory.

mode
A three- or four-digit number, corresponding to the access permission bits. Each digit must be in the range `0`–`7`, and at least three digits must be specified. For more information on permissions, see Appendix B.

Usage Notes

1. The file permission bits specified in *mode* are modified by the file creation mask of the calling process (see “umask” on page 165), and are then used to set the file permission bits of the new directory.
2. The new directory's owner ID is set to the effective user ID (UID) of the calling process.
3. The mkdir service sets the access, change, and modification times for the new directory. It also sets the change and modification times for the directory that contains the new directory.

Example

In the following example, assume that *pathname* and *mode* were assigned a value earlier in the exec:

```
"mkdir (pathname)" mode
```

mkfifo

```
►►──mkfifo──pathname──mode──────────────────────────────────────►◄
```

Function

mkfifo invokes the mknod callable service to create a new FIFO special file.

Parameters

pathname
A pathname for the file.

mode
A three- or four-digit number, corresponding to the access permission bits. Each digit must be in the range `0`–`7`, and at least three digits must be specified. For more information on permissions, see Appendix B.

Usage Notes

1. The file permission bits specified in *mode* are modified by the process's file creation mask (see “umask” on page 165), and then used to set the file permission bits of the file being created.
2. The file's owner ID is set to the process's effective user ID (UID). The group ID is set to the group ID (GID) of the directory containing the file.
3. The mknod service sets the access, change, and modification times for the new file. It also sets the change and modification times for the directory that contains the new file.

Example

In the following example, assume that *pathname* was assigned a value earlier in the exec. The mode `777` grants read-write-execute permission to everyone.

```
"mkfifo (pathname) 777"
```

mknod

```
►►──mknod──pathname──mode──major──minor──────────────────────────►◄
```

Function

mknod invokes the mknod callable service to create a new character special file. You must be a superuser to use this function.

Parameters

pathname
A pathname for the file.

mode
A three- or four-digit number, corresponding to the access permission bits. Each digit must be in the range 0–7, and at least three digits must be specified. For more information on permissions, see Appendix B.

major
The device major number corresponds to a device driver supporting a class of devices (for example, interactive terminals). For information on specifying the device major number, see *OS/390 OpenEdition Planning*.

minor
The number that corresponds to a specific device within the class of devices referred to by the device major number. For information on specifying the device minor number, see *OS/390 OpenEdition Planning*.

Usage Notes

1. The file permission bits specified in *mode* are modified by the process's file creation mask (see “umask” on page 165), and then used to set the file permission bits of the file being created.
2. The file's owner ID is set to the process's effective user ID (UID). The group ID is set to the group ID (GID) of the directory containing the file.
3. The mknod service sets the access, change, and modification times for the new file. It also sets the change and modification times for the directory that contains the new file.

Example

To create **/dev/null** with read-write-execute permission for everyone:

```
"mknod /dev/null 777 4 0"
```

mount

```
►►──mount──pathname──name──type──flags──┬──────┬──────────────────►◄
                                        └─parm─┘
```

Function

mount invokes the mount callable service to mount a file system, making the files in it available for use. You must be a superuser to use this function.

Parameters

pathname
The pathname for the mount point.

name
The name of the file system to be mounted. You must specify HFS data set names as fully qualified names in uppercase letters. Do not enclose the data set name in single quotes.

type
The type of file system, as defined by the FILESYSTYPE parameter on the BPXPRMxx parmlib member (for example, HFS). Specify this as it is specified on the parmlib member.

flags
A value indicating how the file system is to be mounted. To specify the information, you can specify a numeric value (see Appendix A), or one or more of the predefined variables beginning with MTM_. If more than one variable is specified, they are added together like open flags. (RDONLY and RDWR cannot be specified together.) The predefined variables used to derive the appropriate numeric value are:

MTM_NOSUID	This specifies that the SETUID and SETGID mode bits on any executable in this file system are to be ignored when the program is run.
MTM_RDONLY	Mount read-only
MTM_RDWR	Mount read-write
MTM_SYNCHONLY	Mount must be completed synchronously; that is, **mount()** must not return +1.

parm
The name of a variable that contains a parameter string to be passed to the physical file system. The format and content of the string are specified by the physical file system that is to perform the logical mount.

For an HFS file system, *parm* is not used.

Usage Notes

1. In order to mount a file system, the caller must be a superuser.
2. The mount service effectively creates a virtual file system. After a file system is mounted, references to the pathname that is mounted refer to the root directory on the mounted file system.
3. A file system can be mounted at only one point.
4. Parameter specifics for the HFS physical file system:

- The *name* value must be uppercase and must be the name of the data set.
- The *parm* parameter is not used.

Example

To mount the HFS data set HFS.BIN.V1R1M0 on the mountpoint **/v1r1m0** as a read-only file system:

```
"mount /v1r1m0 HFS.BIN.V1R1M0 HFS" mtm_rdonly
```

open

```
►►──open──pathname──o_flags──┬──────┬───────────────────────────────►◄
                             └─mode─┘
```

Function

open invokes the open callable service to access a file and create a file descriptor for it. The file descriptor is returned in RETVAL.

Parameters

pathname
A pathname for the file.

o_flags
One or more numeric values that describe how the file is to be opened. You can specify a numeric value (see Appendix A) or any of the predefined variables that begin with O_ used to derive the appropriate numeric value. For example, the numeric values 128+3 or 131 or the predefined variables *o_creat+o_rdwr* could be used to specify how the file is to be opened:

Variable	Description
O_APPEND	Set the offset to EOF before each write.
O_CREAT	Create the file if it does not exist.
O_EXCL	Fail if the file does exist and O_CREAT is set.
O_NOCTTY	Do not make this file a controlling terminal for the calling process.
O_NONBLOCK	Do not block an open, a read, or a write on the file (do not wait for terminal input).
O_RDONLY	Open for read-only.
O_RDWR	Open for read and write.
O_SYNC	Force synchronous updates.
O_TRUNC	Write, starting at the beginning of the file.
O_WRONLY	Open for write-only.

mode
A three- or four-digit number, corresponding to the access permission bits. If this optional parameter is not supplied, the mode is defaulted to `000`, which is useful for opening an existing file. For an explanation of how mode is handled if you are creating a file, see the usage notes below.

Each digit must be in the range `0`–`7`, and at least three digits must be specified. For more information on permissions, see Appendix B.

Usage Notes

When a file is created with the O_CREAT or O_EXCL options, the file permission bits as specified in the mode parameter are modified by the process's file creation mask (see “umask” on page 165), and then used to set the file permission bits of the file being created.

O_EXCL Option: If the O_EXCL bit is set and the create bit is not set, the O_EXCL bit is ignored.

O_TRUNC Option: Turning on the O_TRUNC bit opens the file as though it had been created earlier but never written into. The mode and owner of the file do not change (although the change time and modification time do); but the file's contents are discarded. The file offset, which indicates where the next write is to occur, points to the first byte of the file.

O_NONBLOCK Option: A FIFO special file is a shared file from which the first data written is the first data read. The O_NONBLOCK option is a way of coordinating write and read requests between processes sharing a FIFO special file. It works this way, provided that no other conditions interfere with opening the file successfully:

- If a file is opened read-only and O_NONBLOCK is specified, the open request succeeds. Control returns to the caller immediately.
- If a file is opened write-only and O_NONBLOCK is specified, the open request completes successfully, provided that another process has the file open for reading. If another process does not have the file open for reading, the request ends with RETVAL set to `−1`.
- If a file is opened read-only and O_NONBLOCK is omitted, the request is blocked (control is not returned to the caller) until another process opens the file for writing.
- If a file is opened write-only and O_NONBLOCK is omitted, the request is blocked (control is not returned to the caller) until another process opens the file for reading.
- If the O_SYNC update option is used, the program is assured that all data updates have been written to permanent storage.

Example

To open or create the file **/u/linda/my.exec** with read-write-execute permission for the owner, and to read and write starting at the beginning of the file:

```
"open /u/linda/my.exec" o_rdwr+o_trunc+o_creat 700
```

opendir

```
►►──opendir──pathname──────────────────────────────────────────►◄
```

Function

opendir invokes the opendir callable service to open a directory stream so that it can be read by **rddir**. The file descriptor is returned in RETVAL.

Parameters

pathname
A pathname for the directory.

Usage Notes

1. You can use **opendir** and **closedir** together with the **rddir** syscall command, but not with the **readdir** command. The **rddir** command reads a directory in the readdir callable service format. Alternatively, you can simply use the **readdir** syscall command to read an entire directory and format it in a stem.
2. The opendir service opens a directory so that the first rddir service (see "rddir" on page 121) starts reading at the first entry in the directory.
3. RETVAL is a file descriptor for a directory only. It can be used only as input to services that expect a directory file descriptor. These services are **closedir**, **rewinddir**, and **rddir**.

Example

To open the directory **/u/edman**:

```
"opendir /u/edman"
```

pathconf

```
►►──pathconf──pathname──name────────────────────────────────────►◄
```

Function

pathconf invokes the pathconf callable service to determine the current values of a configurable limit or option (variable) that is associated with a file or directory. The limit or option is returned in RETVAL.

Parameters

pathname
A pathname for a file or directory.

name
A numeric value that indicates which limit is returned. You can specify a numeric value (see Appendix A) or the predefined variable beginning with PC_ used to derive the appropriate numeric value.

Variable	Description
PC_LINK_MAX	Maximum value of a file's link count.
PC_MAX_CANON	Maximum number of bytes in a terminal canonical input line.
PC_MAX_INPUT	Minimum number of bytes for which space will be available in a terminal input queue; therefore, the maximum number of bytes a portable application may require to be typed as input before reading them.
PC_NAME_MAX	Maximum number of bytes in a filename (not a string length; count excludes a terminating null).
PC_PATH_MAX	Maximum number of bytes in a pathname (not a string length; count excludes a terminating null).
PC_PIPE_BUF	Maximum number of bytes that can be written atomically when writing to a pipe.
PC_POSIX_CHOWN_RESTRICTED	Change ownership (“chown” on page 33) function is restricted to a process with appropriate privileges, and to changing the group ID (GID) of a file only to the effective group ID of the process or to one of its supplementary group IDs.
PC_POSIX_NO_TRUNC	Pathname components longer than 255 bytes generate an error.
PC_POSIX_VDISABLE	Terminal attributes maintained by the system can be disabled using this character value.

Usage Notes

1. If *name* refers to MAX_CANON, MAX_INPUT, or _POSIX_VDISABLE, the following applies:
   - If *pathname* does not refer to a terminal file, the service returns −1 in RETVAL and sets ERRNO to EINVAL.
2. If *name* refers to NAME_MAX, PATH_MAX, or _POSIX_NO_TRUNC, the following applies:
   - If *pathname* does not refer to a directory, the service still returns the requested information using the parent directory of the specified file.
3. If *name* refers to PC_PIPE_BUF, the following applies:

- If *pathname* refers to a pipe or a FIFO, the value returned applies to the referred-to object itself. If *pathname* refers to a directory, the value returned applies to any FIFOs that exist or can be created within the directory. If *pathname* refers to any other type of file, the pathconf service returns `–1` in RETVAL and sets the ERRNO to EINVAL.

4. If *name* refers to PC_LINK_MAX, the following applies:

   - If *pathname* refers to a directory, the value returned applies to the directory.

Example

To determine the maximum number of bytes allowed in a pathname in the root directory:

```
"pathconf /" pc_name_max
```

pause

```
►►──pause──────────────────────────────────────────────────────►◄
```

Function

pause invokes the pause callable service to suspend execution of the calling thread until delivery of a signal that either executes a signal-catching function or ends the thread. See “Using the REXX Signal Services” on page 8 for more information.

Usage Notes

1. A thread that calls pause does not resume processing until a signal is delivered with an action to either process a signal-handling function or end the thread. Some signals can be blocked by the thread's signal mask; see “sigprocmask” on page 146 for details.
2. If an incoming unblocked signal ends the thread, pause never returns to the caller.
3. A return code is set when any failures are encountered that prevent this function from completing successfully.

pfsctl

```
►►──pfsctl──type──command──buffer──┬────────┬──────────────────────►◄
                                   └─length─┘
```

Function

pfsctl invokes the pfsctl service to issue physical file system (PFS) control commands to a PFS. The meaning of the command and argument are specific to and defined by the PFS.

Parameters

type
: The type of file system, as defined by the FILESYSTYPE parameter on the BPXPRMxx parmlib member—for example, HFS. Specify this as it is specified on the parmlib member.

command
: The name of a PFS control command.

buffer
: The name of a buffer containing the argument to be passed to the PFS. The argument is limited to 4095 bytes.

length
: An optional numeric value, indicating the length of the argument. If length is not specified, the length of the buffer is used. The maximum length allowed is 4095 bytes.

Usage Notes

1. This service is provided for communication between a program running in a user process and a physical file system.

   It is similar to **ioctl**, but the command is directed to the physical file system itself rather than to, or for, a particular file or device.

2. As an example of how you could use this function in writing a physical file system, consider the requirement to display status and performance statistics about the physical file system. You can collect this information in the physical file system, but you need a way to display it to the user.

   With **pfsctl**, your status utility program can easily fetch the information it needs from the physical file system.

pipe

```
►►──pipe──stem──────────────────────────────────────────────────►◄
```

Function

pipe invokes the pipe callable service to create a pipe; or an I/O channel that a process can use to communicate with another process, another thread (in this same process or another process), or, in some cases, with itself. Data can be written into one end of the pipe and read from the other end.

Parameters

stem

On return, *stem.0* contains the number of variables returned. Two stem variables are returned:

stem.1	The file descriptor for the end of the pipe that you read from
stem.2	The file descriptor for the end of the pipe that you write to

Usage Notes

When the pipe call creates a pipe, the O_NONBLOCK and FD_CLOEXEC flags are turned off on both ends of the pipe. You can turn these flags on or off by invoking:

- “f_setfl” on page 62 for the flag O_NONBLOCK
- “f_setfd” on page 61 for the flag FD_CLOEXEC

Example

To create a pipe:

```
"pipe pfd."
```

pt3270

```
►►──pt3270──fd──option──────────────────────────────────────────►◄
```

Function

pt3270 invokes the tcgetattr and tcsetattr callable services to query, set, and reset 3270 passthrough mode.

A REXX program running under a shell started from the OMVS TSO/E command can use this service to send and receive a 3270 data stream or issue TSO/E commands.

Parameters

fd The file descriptor for the file.

option
A number that identifies the service being requested:

Number	Service
1	Query 3270 passthrough support for this file. On return, RETVAL will contain a code for the current state of the file descriptor, or -1 if there is an error. 0 or -1 This file cannot support 3270 passthrough mode. 1 This file can support 3270 passthrough mode. 3 This file is currently in 3270 passthrough mode.
2	Set 3270 passthrough support for this file. If this is attempted on a file that does not support 3270 passthrough mode, on return RETVAL contains -1 and ERRNO contains the value for ENOSYS.
3	Reset 3270 passthrough support for this file.

Example

The following is an example of a REXX program that can accept a TSO/E command as its argument and issue the command through OMVS using 3270 passthrough mode. This REXX program would be located in the HFS and run as a command from the shell.

```
/* rexx */
parse arg cmd
if cmd=' then return
address syscall
'pt3270 1 2'                      /* set passthrough mode on stdout */
if retval=-1 then
   do
   say 'Cannot set passthrough mode' retval errno errnojr
   return
   end
buf='ff51000000010001'x ||,       /* OMVS passthrough command       */
    d2c(length(cmd),4) || cmd
"write 1 buf"                     /* send command to OMVS           */
"read 1 ibuf 1000"                /* discard the response           */
'pt3270 1 3'                      /* reset passthrough mode         */
return 0
```

quiesce

```
►►──quiesce──name──────────────────────────────────────────────►◄
```

Function

quiesce invokes the quiesce callable service to quiesce a file system, making the files in it unavailable for use. You must be a superuser to quiesce a file system.

Parameters

name
The name of the file system to be quiesced, specified as the name of an HFS data set. You must specify HFS data set names as fully qualified names in uppercase letters. Do not enclose the data set name in single quotes.

Usage Notes

1. After a quiesce service request, the file system is unavailable for use until a subsequent unquiesce service request is received.
2. Users accessing files in a quiesced HFS file system are suspended until an unquiesce request for the file system is processed. Other file systems may send an EAGAIN instead of suspending the user.
3. If a file system that is not mounted is quiesced, that file system cannot be mounted until the file system is unquiesced. This ensures that no one can use the file system while it is quiesced.

Example

To quiesce an HFS data set named HFS.USR.SCHOEN:

```
"quiesce HFS.USR.SCHOEN"
```

rddir

```
►►──rddir──fd──variable──length──────────────────────────────────►◄
```

Function

rddir invokes the readdir callable service to read multiple name entries from a directory and format it in the readdir callable service format. To format this type of information in a stem, see “readdir” on page 125. The number of entries read is returned in RETVAL.

Parameters

fd	The file descriptor (a number) for the directory to be read.
variable	The name of the buffer into which the directory entries are to be read.
length	The size of the buffer. After the read completes, the length of *variable* is the size of the buffer. The number of entries is returned in RETVAL.

Usage Notes

1. You can use this command only with file descriptors opened using the opendir syscall command. The **rddir** syscall command reads a directory in the readdir callable service format. You can use **opendir**, **rewinddir**, and **closedir** together with the **rddir** syscall command, but not with the **readdir** syscall command. Alternatively, you can simply use the **readdir** syscall command to read an entire directory and format it in a stem.
2. The buffer contains a variable number of variable-length directory entries. Only full entries are placed in the buffer, up to the buffer size specified, and the number of entries is returned.
3. Each directory entry returned has the following format:
   - 2-byte Entry_length. The total entry length, including itself.
   - 2-byte Name_length. Length of the following Member_name subfield.
   - Member_name. A character field of length Name_length. This name is not null-terminated.
   - File system specific data. If name_length + 4 = entry_length, this subfield is not present.

   The entries are packed together, and the length fields are not aligned on any particular boundary.
4. The buffer returned by one call to the readdir service must be used again on the next call to the readdir service to continue reading entries from where you left off. The buffer must not be altered between calls, unless the directory has been rewound.
5. The end of the directory is indicated in either of two ways:
   - A RETVAL of `0` entries is returned.
   - Some physical file systems may return a null name entry as the last entry in the caller's buffer. A null name entry has an Entry_length of `4` and a Name_length of `0`.

Both conditions should be checked for by the caller of the readdir service.

Example

To read the entries from the directory with file descriptor 4 into the buffer named *buf*, which is 300 bytes long:

```
"rddir 4 buf 300"
```

read

```
▶▶──read──fd──variable──length──────────────────────────────▶◀
```

Function

read invokes the read callable service to read a specified number of bytes from a file into a buffer that you provide. The number of bytes read is returned in RETVAL.

Parameters

fd	The file descriptor (a number) for the file to be read.
variable	The name of the buffer into which the data is to be read.
length	The maximum number of characters to read. After the read completes, the length of *variable* is the number of bytes read. This value is also returned in RETVAL.

Usage Notes

Length: The value of *length* is not checked against any OS/390 OpenEdition system limit.

Access Time: A successful read updates the access time of the file read.

Origin of Bytes Read: If the file specified by *fd* is a regular file, or any other type of file where a seek operation is possible, bytes are read from the file offset associated with the file descriptor. A successful read increments the file offset by the number of bytes read.

For files where no seek operation is possible, there is no file offset associated with the file descriptor. Reading begins at the current position in the file.

Number of Bytes Read: When a read request completes, the RETVAL field shows the number of bytes actually read—a number less than or equal to the number specified as *length*. The following are some reasons why the number of bytes read might be less than the number of bytes requested:

- Fewer than the requested number of bytes remained in the file; the end of file was reached before *length* bytes were read.
- The service was interrupted by a signal after some but not all of the requested bytes were read. (If no bytes were read, the return value is set to —1 and an error is reported.)
- The file is a pipe, FIFO, or special file and fewer bytes than *length* specified were available for reading.

There are several reasons why a read request might complete successfully with no bytes read (that is, with RETVAL set to 0). For example, zero bytes are read in these cases:

- The call specified a *length* of zero.
- The starting position for the read was at or beyond the end of the file.
- The file being read is a FIFO file or a pipe, and no process has the pipe open for writing.

- The file being read is a slave pseudoterminal and a zero-length canonical file was written to the master.

Nonblocking: If a process has a pipe open for reading with nonblocking specified, a request to read from the file ends with a return value of —1 and a “Resource temporarily unavailable” return code. But if nonblocking was not specified, the read request is blocked (does not return) until some data is written or the pipe is closed by all other processes that have the pipe open for writing.

Both master and slave pseudoterminals operate this way, too, except that how they act depends on how they were opened. If the master or the slave is opened blocking, the reads are blocked if there is no data. If it is opened nonblocking, EAGAIN is returned if there is no data.

SIGTTOU Processing: The read service causes signal **SIGTTIN** to be sent under the following conditions:

- The process is attempting to read from its controlling terminal, and
- The process is running in a background process group, and
- The **SIGTTIN** signal is not blocked or ignored, and
- The process group of the process is not orphaned.

If these conditions are met, **SIGTTIN** is sent. If **SIGTTIN** has a handler, the handler gets control and the read ends with the return code set to EINTRO. If **SIGTTIN** is set to default, the process stops in the read and continues when the process is moved to the foreground.

Example

In the following example, assume that *fd* was assigned a value earlier in the exec. This reads 1000 characters from the file *fd* into the buffer *buf*:

```
"read (fd) buf 1000"
```

readdir

```
►►──readdir──pathname──stem──┬───────┬──────────────────────────────►◄
                             └─stem2─┘
```

Function

readdir invokes the opendir, readdir, and closedir callable services to read multiple name entries from a directory and format the information in a stem.

Parameters

pathname	A pathname for a directory.
stem	Upon return, *stem.0* contains the number of directory entries returned. *stem.1* through *stem.n* (where *n* is the number of entries returned) each contain a directory entry.
stem2	If the optional *stem2* is provided, *stem2.1* through *stem2.n* (where *n* is the number of structures returned) each contain the stat structure for a directory entry.
	You can use the predefined variables that begin with ST_ or their equivalent numeric values (see Appendix A) to access the stat values. For example, *stem2.1.st_size* or *stem2.1.8* accesses the file size for the first directory entry. See “fstat” on page 67 for a list of the ST_ variables.

Usage Notes

The **rddir** command reads a directory in the readdir callable service format. You can use **opendir** and **closedir** together with the **rddir** syscall command, but not with the **readdir** syscall command. The **readdir** syscall command reads a directory and formats it in a stem.

Example

To read the root directory and return information about the directory entries and a stat structure for each directory entry:

```
"readdir / root. rootst."
```

readfile

```
►►──readfile──pathname──stem──────────────────────────────────────►◄
```

Function

readfile invokes the open, read, and close callable services to read from a text file and format it in a stem.

Parameters

pathname	A pathname for the file to be read.
stem	Upon return, *stem.0* contain the number of lines read. *stem.1* through *stem.n* (where *n* is the number of lines) each contains a line read.

Usage Notes

1. The maximum allowable length of a line in the file is 1024 characters; if there are lines longer than that, the **RC** is –23.
2. The newline characters that delimit the lines in a text file are stripped before the lines are saved in the stem.

Example

In the following example, assume that *pathname* was assigned a value earlier in the exec:

```
"readfile (pathname) file."
```

readlink

```
►►──readlink──pathname──┬──────────┬──────────────────────────────►◄
                        └─variable─┘
```

Function

readlink invokes the readlink callable service to read the contents of a symbolic link. A symbolic link is a file that contains the pathname for another file. The length, in bytes, of the contents of the link is returned in RETVAL. If a *variable* is specified, the pathname is read into it.

Parameters

pathname	A pathname for the symbolic link.
variable	The name of the variable to hold the contents of the symbolic link. After the link is read, the length of the variable is the length of the symbolic link.

Example

In the following example, assume that *sl* and *linkbuf* were assigned a value earlier in the exec:

```
"readlink (sl) linkbuf"
```

realpath

```
►►──realpath──pathname──variable──────────────────────────────────►◄
```

Function

realpath invokes the realpath callable service to resolve a pathname to a full pathname without any symbolic links.

Parameters

pathname
The pathname to resolve.

variable
The name of the variable to contain the resolved pathname that is returned.

Example

The following example retrieves the real pathname for the working directory:

```
"realpath . mycwd"
```

rename

```
►►──rename──old_pathname──new_pathname──────────────────────────►◄
```

Function

rename invokes the rename callable service to change the name of a file or directory.

Parameters

old_pathname	An existing pathname.
new_pathname	A new pathname.

Usage Notes

The rename service changes the name of a file or directory from *old_pathname* to *new_pathname*. When renaming finishes successfully, the change and modification times for the parent directories of *old_pathname* and *new_pathname* are updated.

The calling process needs write permission for the directory containing *old_pathname* and the directory containing *new_pathname*. The caller does not need write permission for the files themselves.

Renaming Files: If *old_pathname* and *new_pathname* are links referring to the same file, rename simply returns successfully.

If *old_pathname* is the name of a file, *new_pathname* must also name a file, not a directory. If *new_pathname* is an existing file, it is unlinked. Then the file specified as *old_pathname* is given *new_pathname*. The pathname *new_pathname* always stays in existence; at the beginning of the operation, *new_pathname* refers to its original file, and at the end, it refers to the file that used to be *old_pathname*.

Renaming Directories: If *old_pathname* is the name of a directory, *new_pathname* must also name a directory, not a file. If *new_pathname* is an existing directory, it must be empty, containing no files or subdirectories. If empty, it is removed, as described in "rmdir" on page 131.

new_pathname cannot be a directory under *old_pathname*; that is, the old directory cannot be part of the pathname prefix of the new one.

Example

In the following example, assume that *old* and *new* were assigned values earlier in the exec:

```
"rename (old) (new)"
```

rewinddir

```
►►──rewinddir──fd──────────────────────────────────────────────►◄
```

Function

rewinddir invokes the rewinddir callable service to “rewind,” or reset, to the beginning of an open directory. The next call to **rddir** reads the first entry in the directory.

Parameters

fd	The file descriptor (a number) returned from an **opendir** syscall command. This is the file descriptor for the directory to be reset.

Usage Notes

You can use this command only with file descriptors opened using the **opendir** syscall command. The **rddir** syscall command reads a directory in the readdir callable service format. You can use **opendir**, **rewinddir**, and **closedir** together with the **rddir** syscall command, but not with the **readdir** syscall command. Alternatively, you can simply use **readdir** syscall command to read an entire directory and format it in a stem.

If the contents of the directory you specify have changed since the directory was opened, a call to the rewinddir service will reset the pointer into the directory to the beginning so that a subsequent call to the readdir service will read the new contents.

Example

To rewind the directory associated with file descriptor 4:

```
"rewinddir 4"
```

rmdir

```
►►──rmdir──pathname──────────────────────────────────────────►◄
```

Function

rmdir invokes the rmdir callable service to remove a directory. The directory must be empty.

Parameters

pathname	A pathname for the directory.

Usage Notes

1. The directory must be empty.
2. If the directory is successfully removed, the change and modification times for the parent directory are updated.
3. If the link count of the directory becomes zero and no process has the directory open, the directory itself is deleted. The space occupied by the directory is freed for new use.
4. If any process has the directory open when the last link is removed, the directory itself is not removed until the last process closes the directory. New files cannot be created under a directory after the last link is removed, even if the directory is still open.

Example

To remove the directory **/u/ehk0**:

```
"rmdir /u/ehk0"
```

setegid

```
►►──setegid──gid──────────────────────────────────────────────►◄
```

Function

setegid invokes the setegid callable service to set the effective group ID (GID) of the calling process.

Parameters

gid
The numeric GID that the calling process is to assume.

Usage Notes

1. If *gid* is equal to the real group ID or the saved set group ID of the process, the effective group ID is set to *gid*.
2. If *gid* is not the same as the real group ID, and the calling process has the appropriate privileges, the effective group ID is set to *gid*.
3. The setegid service does not change any supplementary group IDs of the calling process.

Example

In the following example, assume that *gid* was assigned a value earlier in the exec:

```
"setegid" gid
```

seteuid

```
►►──seteuid──uid──────────────────────────────────────────────►◄
```

Function

seteuid invokes the seteuid callable service to set the effective user ID (UID) of the calling process.

Parameters

uid
The numeric UID that the calling process is to assume.

Usage Notes

1. A user can switch to superuser authority (with an effective UID of `0`) if the user is permitted to the BPX.SUPERUSER FACILITY class profile within RACF.
2. If *uid* is the same as the process's real or saved set UID, or the user has the appropriate privilege, the seteuid service sets the effective UID to be the same as *uid*.

Example

In the following example, assume that *uid* was assigned a value earlier in the exec:

```
"seteuid (uid)"
```

setgid

```
►►──setgid──gid──────────────────────────────────────────────────►◄
```

Function

setgid invokes the setgid callable service to set the real, effective, and saved set group IDs (GIDs) for the calling process.

Parameters

gid
The numeric GID that the calling process is to assume.

Usage Notes

1. If *gid* is equal to the real group ID or the saved set group ID of the process, the effective group ID is set to *gid*.
2. If *gid* is not the same as the real group ID, and the calling process has the appropriate privileges, then the real, saved set, and effective group IDs are set to *gid*.
3. The setgid service does not change any supplementary group IDs of the calling process.

Example

In the following example, assume that *gid* was assigned a value earlier in the exec:

```
"setgid (gid)"
```

setgrent

```
►►──setgrent──────────────────────────────────────────────►◄
```

Function

setgrent invokes the setgrent callable service to rewind, or reset to the beginning, the group database, allowing repeated searches. For more information, see "getgrent" on page 76.

setpgid

```
►►──setpgid──pid──pgid──────────────────────────────────────────►◄
```

Function

setpgid invokes the setpgid callable service to place a process in a process group. To identify the group, you specify a process group ID. You can assign a process to a different group, or you can start a new group with that process as its leader.

Parameters

pid
The numeric process ID (PID) of the process to be placed in a process group. If the ID is specified as `0`, the system uses the process ID of the calling process.

pgid
The ID of the process group. If the ID is specified as `0`, the system uses the process group ID indicated by *pid*.

Usage Notes

1. The process group ID to be assigned to the group must be within the calling process's session.
2. The subject process (the process identified by *pid*) must be a child of the process that issues the service, and it must be in the same session; but it cannot be the session leader. It can be the caller.

Example

In the following example, assume that *pid* and *pgid* were assigned values earlier in the exec:

```
"setpgid" pid pgid
```

setpwent

```
►►──setpwent──────────────────────────────────────────────►◄
```

Function

setpwent invokes the setpwent callable service to effectively rewind the user database to allow repeated searches. For more information, see "getpwent" on page 91.

setrlimit

```
►►──setrlimit──resource──stem──────────────────────────────►◄
```

Function

setrlimit invokes the setrlimit callable service to set resource limits for the calling process. A resource limit is a pair of values; one specifies the current limit and the other a maximum limit.

Parameters

resource

The resource whose limit is being set. The maximum resource limit is RLIM_INFINITY.

You can use the predefined variables beginning with RLIMIT_, or their equivalent numeric values, to specify the resource. (See Appendix A for the numeric values.)

Variable	Description	Allowable Range
RLIMIT_AS	Maximum address space size for a process.	10 485 760—2 147 483 647
RLIMIT_CORE	Maximum size (in bytes) of a core dump created by a process.	0—2 147 483 647
RLIMIT_CPU	Maximum amount of CPU time (in seconds) used by a process.	7—2 147 483 647
RLIMIT_FSIZE	Maximum files size (in bytes) created by a process.	0—2 147 483 647
RLIMIT_NOFILE	Maximum number of open file descriptors for a process.	5-65 535

stem

The name of the stem variable used to set the limit. *stem.1* is the first word, which sets the current limit, and *stem.2* is the second word, which sets the maximum limit. The values for each word depend on the *resource* specified. To specify no limit, use RLIM_INFINITY.

Usage Notes

1. The current limit may be modified to any value that is less than or equal to the maximum limit. For the RLIMIT_CPU, RLIMIT_NOFILE, and RLIMIT_AS resources, if the setrlimit service is called with a current limit that is lower than the current usage, the setrlimit service fails with an EINVAL errno.
2. The maximum limit may be lowered to any value that is greater than or equal to the current limit.
3. The maximum limit can only be raised by a process that has superuser authority.
4. Both the current limit and maximum limit can be changed via a single call to setrlimit.

5. If the setrlimit service is called with a current limit that is greater than the maximum limit, setrlimit returns an EINVAL errno.
6. The resource limit values are propagated across exec and fork. An exception exists for exec. If a daemon process invokes exec and it invoked setuid before invoking exec, the limit values are set based on the limit values specified in OS/390 OpenEdition parmlib member BPXPRMxx.
7. For a process that is not the only process within an address space, the RLIMIT_CPU and RLIMIT_AS limits are shared with all the processes within the address space. For RLIMIT_CPU, when the current limit is exceeded, action is taken on the first process within the address space. If the action is termination, all the processes within the address space are terminated.
8. In addition to the RLIMIT_CORE limit values, CORE dump defaults are set by SYSMDUMP defaults. See the *OS/390 MVS Initialization and Tuning Guide* for information on setting up SYSMDUMP defaults via the IEADMR00 parmlib member.
9. Core dumps are taken in 4160-byte increments. Therefore, RLIMIT_CORE values affect the size of core dumps in 4160-byte increments. For example, if the RLIMIT_CORE current limit value is 4000, core dumps will contain no data. If the RLIMIT_CORE current limit value is 8000, the maximum size of a core dump is 4160 bytes.
10. Limits may have an infinite value of RLIM_INFINITY.
11. When setting RLIMIT_NOFILE, the maximum limit cannot exceed the system defined limit of 65 535.
12. When setting RLIMIT_NOFILE, the current limit must be set higher than the value of the highest open file descriptor. Attempting to lower the current limit to a value less than or equal to the highest open file descriptor results in an error of EINVAL.
13. When setting RLIMIT_FSIZE, a limit of 0 prevents the creation of new files and the expansion of existing files.

Example

To reduce the maximum number of open files to 100 and current limit to 50:

```
r.2=100
r.1=50
"setrlimit" rlimit_nofile r.
```

setsid

```
►►──setsid──sid──────────────────────────────────────────────────►◄
```

Function

setsid invokes the setsid callable service to create a new session with the calling process as its session leader. The caller becomes the group leader of a new process group.

Parameters

sid
: The process ID of the calling process, which becomes the session or process group ID of the new process group.

Usage Notes

The calling process does not have a controlling terminal.

Example

In the following example, assume that *sid* was assigned a value earlier in the exec:

```
"setsid" sid
```

setuid

```
►►──setuid──uid──────────────────────────────────────────────────►◄
```

Function

setuid invokes the setuid callable service to set the real, effective, and saved set user IDs for the calling process.

Parameters

uid
The numeric UID the process is to assume.

Usage Notes

1. A user can switch to superuser authority (with an effective UID of `0`) if the user is permitted to the BPX.SUPERUSER FACILITY class profile within RACF.
2. If *uid* is the same as the process's real UID or the saved set UID, the setuid service sets the effective UID to be the same as *uid*.

   If *uid* is not the same as the real UID of the process, and the calling process has appropriate privileges, then the real, effective, and saved set UIDs are set to *uid*.

Example

In the following example, assume that *uid* was assigned a value earlier in the exec:

```
"setuid" uid
```

sigaction

```
►►──sigaction──signal──new_handler──new_flag──────────────────►
►──old_handler──old_flag─────────────────────────────────────►◄
```

Function

sigaction invokes the sigaction callable service to examine, change, or both examine and change the action associated with a specific signal for all the threads in the process.

Note: All threads within a process share the signal handlers (a set of additional signals to be masked) and the flags specified by the sigaction callable service.

Parameters

signal

The signal, as specified by a numeric value or a predefined variable beginning with SIG.

Variable	Description
SIGABND	Abend
SIGABRT	Abnormal termination
SIGALRM	Timeout
SIGBUS	Bus error
SIGCHLD	Child process terminated or stopped
SIGCONT	Continue if stopped
SIGDCE	Exclusive use by DCE
SIGFPE	Erroneous arithmetic operation, such as division by zero or an operation resulting in an overflow
SIGHUP	Hangup detected on the controlling terminal
SIGILL	Termination (cannot be caught or ignored)
SIGINT	Interactive attention
SIGIO	Completion of input or output
SIGIOERR	Error on input/output; used by the C runtime library
SIGKILL	Termination that cannot be caught or ignored
SIGPIPE	Write on a pipe with no readers
SIGPOLL	Pollable event
SIGPROF	Profiling timer expired
SIGQUIT	Interactive termination
SIGSEGV	Detection of an incorrect memory reference
SIGSTOP	Stop that cannot be caught or ignored
SIGSYS	Bad system call
SIGTERM	Termination
SIGTRAP	Trap used by the ptrace callable service

Variable	Description
SIGTSTP	Interactive stop
SIGTTIN	Read from a controlling terminal attempted by a member of a background process group
SIGTTOU	Write from a controlling terminal attempted by a member of a background process group
SIGURG	High bandwidth data is available at a socket
SIGUSR1	Reserved as application-defined signal 1
SIGUSR2	Reserved as application-defined signal 2
SIGVTALRM	Virtual timer expired
SIGXCPU	CPU time limit exceeded
SIGXFSZ	File size limit exceeded

new_handler
Specifies the new setting for handling the signal. The following predefined variables may be used for *new_handler*:

Variable	Description
SIG_CAT	Set signal handling to catch the signal.
SIG_DFL	Set signal handling to the default action.
SIG_IGN	Set signal handling to ignore the signal.
SIG_QRY	Query the handling for that signal.

new_flag
Used to modify the behavior of the specified signal. Specify one of these:

Variable	Description
`0`	Use the default behavior.
SA_NOCLDWAIT	Do not create zombie processes when child processes exit.
SA_RESETHAND	Reset the signal action to the default, SIG_DFL, when delivered.
SA_NOCLDSTOP	Do not generate **SIGCHLD** when the child processes stop.

old_handler
A variable name for the buffer where the system returns the old (current) signal handling action.

old_flag
The name of the variable that will store the old (current) signal action flags.

Usage Notes

1. If *new_handler* is set to the action SIG_DFL for a signal that cannot be caught or ignored, the sigaction request is ignored and the return value is set to `0`.
2. Setting a signal action to ignore for a signal that is pending causes the pending signal to be discarded.
3. Setting signal action SIG_IGN or catch for signals **SIGSTOP** or **SIGKILL** is not allowed.
4. Setting signal action SIG_IGN for **SIGCHLD** or **SIGIO** is not allowed.

5. The sigaction caller's thread must be registered for signals. You can register the thread by calling **syscalls('SIGON')**. If the thread is not registered for signals, the sigaction service fails with an ERRNO of EINVAL and ERRNOJR of JRNotSigSetup.

Example

To catch a **SIGALRM** signal:

```
"sigaction" sigalrm sig_cat 0 "prevhndlr prevflag"
```

sigpending

```
►►──sigpending──variable──────────────────────────────────────────►◄
```

Function

sigpending invokes the sigpending callable service to return the union of the set of signals pending on the thread and the set of signals pending on the process. Pending signals at the process level are moved to the thread that called the sigpending callable service.

Parameters

variable

The name of the variable that will store a string of 64 characters with values `0` or 1, representing the 64 bits in a signal mask.

Example

To invoke sigpending:

```
"sigpending sigset"
```

sigprocmask

```
►►──sigprocmask──number──new_mask──variable──────────────────────►◄
```

Function

sigprocmask invokes the sigprocmask callable service to examine or change the calling thread's signal mask.

Parameters

number

To specify the action to be taken on the thread's signal mask, you can specify a numeric value (see Appendix A) or the predefined variable beginning with SIG_ used to derive the appropriate numeric value. Use one of the following predefined variables:

Variable	Description
SIG_BLOCK	Add the signals in *new_mask* to those to be blocked for this thread.
SIG_SETMASK	Replace the thread's signal mask with *new_mask*.
SIG_UNBLOCK	Delete the signals in *new_mask* from those blocked for this thread.

new_mask

The new signal mask, a string of 64 characters with values `0` or 1. The first character represents signal number 1. A string shorter than 64 characters is padded on the right with zeros. Mask bits set on represent signals that are blocked. For more information on signals, see “Using the REXX Signal Services” on page 8.

variable

The name of the buffer that will store the old signal mask, a string of 64 characters with values `0` or 1, representing the 64 bits in a signal mask. Mask bits set on represent signals that are blocked. A zero indicates that no signal mask was returned.

Usage Notes

1. The sigprocmask service examines, changes, or both examines and changes the signal mask for the calling thread. This mask is called the thread's signal mask. If there are any pending unblocked signals, either at the process level or at the current thread's level after changing the signal mask, at least one of the signals is delivered to the thread before the sigprocmask service returns.
2. You cannot block the **SIGKILL** and the **SIGSTOP** signals. If you call the sigprocmask service with a request that would block those signals, that part of your request is ignored and no error is indicated.
3. A request to block signals that are not supported is accepted, and a return value of zero is returned.
4. All pending unblocked signals are moved from the process level to the current thread.

Example

In the following example, assume that *newsigset* was assigned a value earlier in the exec:

```
"sigprocmask" sig_setmask newsigset "oldsigset"
```

sigsuspend

```
►►──sigsuspend──mask──────────────────────────────────────────►◄
```

Function

sigsuspend invokes the sigsuspend callable service to replace a thread's current signal mask with a new signal mask; then it suspends the caller's thread until delivery of a signal whose action is either to process a signal-catching service or to end the thread.

Parameters

mask

The new signal mask, a string of up to 64 characters with the values `0` or 1. The first character represents signal number 1. A string shorter than 64 characters is padded on the right with zeros. For more information on signals, see “Using the REXX Signal Services” on page 8.

Usage Notes

1. The caller's thread starts running again when it receives one of the signals not blocked by the mask set by this call, or a system failure occurs that sets the return code to some value other than EINTR.
2. The signal mask represents a set of signals that will be blocked. Blocked signals do not “wake up” the suspended service. The signals **SIGSTOP** and **SIGKILL** cannot be blocked or ignored; they are delivered to the program no matter what the signal mask specifies.
3. If the signal action is to end the thread, the sigsuspend service does not return.
4. All pending unblocked signals are moved from the process level to the current thread.

Example

In the following example, assume that *sigmask* was assigned a value earlier in the exec:

```
"sigsuspend" sigmask
```

sleep

```
►►──sleep──number──────────────────────────────────────────────►◄
```

Function

sleep invokes the sleep callable service to suspend running of the calling thread (process) until either the number of seconds specified by *number* has elapsed, or a signal is delivered to the calling thread to invoke a signal-catching function or end the thread.

Parameters

number

The number of seconds to suspend the process. For more information on signals, see “Using the REXX Signal Services” on page 8.

Usage Notes

1. The suspension can actually be longer than the requested time, due to the scheduling of other activity by the system.
2. The sleep service suspends the thread running for a specified number of seconds, or until a signal is delivered to the calling thread that invokes a signal-catching function or ends the thread. An unblocked signal received during this time prematurely “wakes up” the thread. The appropriate signal-handling function is invoked to handle the signal. When that signal-handling function returns, the sleep service returns immediately, even if there is “sleep time” remaining.
3. The sleep service returns a zero in RETVAL if it has slept for the number of seconds specified. If the time specified by *number* has not elapsed when the sleep service is interrupted because of the delivery of a signal, the sleep service returns the unslept amount of time (the requested time minus the time actually slept when the signal was delivered) in seconds. Any time consumed by signal-catching functions is not reflected in the value returned by the sleep service.
4. The following are usage notes for a **SIGALRM** signal generated by the alarm or kill calls during the execution of the sleep call:
   - If the calling thread has **SIGALRM** blocked prior to calling the sleep service, the sleep service does not return when **SIGALRM** is generated, and the **SIGALRM** signal is left pending when sleep returns.
   - If the calling process has **SIGALRM** ignored when the **SIGALRM** signal is generated, then the sleep service does not return and the **SIGALRM** signal is ignored.
   - If the calling process has **SIGALRM** set to a signal-catching function, that function interrupts the sleep service and receives control. The sleep service returns any unslept amount of time, as it does for any other type of signal.
5. An EC6 abend is generated when the caller's PSW key or RB state prevents signals from being delivered.

Example

In the following example, assume that *timer* was assigned a value earlier in the exec:

```
"sleep (timer)"
```

spawn

```
►►──spawn──pathname──fd_count──fd_map──arg_stem──env_stem──────────►◄
```

Function

spawn invokes the spawn callable service to create a new process, called a *child process*, to run a hierarchical file system (HFS) executable file. It remaps the calling process's file descriptors for the child process.

Parameters

pathname

A pathname for the executable file. Pathnames can begin with or without a slash:

- A pathname that begins with a slash is an absolute pathname, and the search for the file starts at the root directory.
- A pathname that does not begin with a slash is a relative pathname, and the search for the file starts at the working directory.

fd_count

The number of file descriptors that can be inherited by the child process. In the new process, all file descriptors greater than or equal to *fd_count* are closed.

fd_map

A stem variable. The stem index specifies the child's file descriptor; for example, *stem.0* specifies the child's file descriptor `0`. This array selects the file descriptors to be inherited. The value assigned to the variable indicates the parent's file descriptor that will be mapped to the child's file descriptor. For example, if *stem.0* is 4, the child process inherits the parent's file descriptor 4 as its descriptor `0`. Any of the stem variables that contains a negative number or a nonnumeric value is closed in the child.

arg_stem

A stem variable. *stem.0* contains the number of arguments you are passing to the program. The first argument should always specify the absolute or relative pathname of the program to be executed. If a relative pathname is used and PATH is specified, PATH is used to resolve the name; otherwise, the name is processed as relative to the current directory. If a PATH environment variable is not passed, the first argument should specify the absolute pathname or a relative pathname for the program.

env_stem

A stem variable. *stem.0* contains the number of environment variables that you want the program to be run with. To propagate the current environment, pass `__environment` . Specify each environment variable as VNAME=*value*.

Usage Notes

1. The new process (called the *child process*) inherits the following attributes from the process that calls **spawn** (called the *parent process*):
   - Session membership.
   - Real user ID.
   - Real group ID.

- Supplementary group IDs.
- Priority.
- Working directory.
- Root directory.
- File creation mask.
- The process group ID of the parent is inherited by the child.
- Signals set to be ignored in the parent are set to be ignored in the child.
- The signal mask is inherited from the parent.

2. The new child process has the following differences from the parent process:
   - The child process has a unique process ID (PID) that does not match any active process group ID.
   - The child has a different parent PID (namely, the PID of the process that called **spawn**).
   - If the *fd_count* parameter specified a 0 value, the child has its own copy of the parent's file descriptors, except for those files that are marked FCTLCLOEXEC or FCTLCLOFORK. The files marked FCTLCLOEXEC or FCTLCLOFORK are not inherited by the child. If the filedesc_count parameter specifies a value greater than 0, the parent's file descriptors are remapped for the child as specified in the fd_map stem with a negative number or non-numeric value.
   - The FCTLCLOEXEC and FCTLCLOFORK flags are not inherited from the parent file descriptors to the child's.
   - The foreground process group of the session remains unchanged.
   - The process and system utilization times for the child are set to zero.
   - Any file locks previously set by the parent are not inherited by the child.
   - The child process has no alarms set (similar to the results of a call to the alarm service with Wait_time specified as zero) and has no interval timers set.
   - The child has no pending signals.
   - The child gets a new process image, which is not a copy of the parent process, to run the executable file.
   - Signals set to be caught are reset to their default action.
   - Memory mappings established by the parent via the shmem or mmap services are not inherited by the child.
   - If the set-user-ID mode bit of the new executable file is set, the effective user ID and saved set-user-ID mode of the process are set to the owner user ID of the new executable file.
   - If the set-group-ID mode bit of the new executable file is set, the effective group ID and saved set-group-ID bit of the process are set to the owner user ID of the new executable file.

   The last parameter that **spawn** passed to the executable file identifies the caller of the file as the exec or spawn service.

3. To control whether the spawned child process runs in a separate address space from the parent address space or in the same address space, you can specify the _BPX_SHAREAS environment variable. If _BPX_SHAREAS is not specified, is set to NO, or contains an unsupported value, the child process to be created will run in a separate address space from the parent process.

   _BPX_SHAREAS=YES indicates that the child process to be created is to run in the same address space as the parent. If the program to be run is a set-user-ID or set-group-ID program that will cause the effective user-ID or group-ID of the child process to be different from that of the parent process, the _BPX_SHAREAS=YES value is ignored and the child process runs in its own address space.
4. In addition to recognizing the _BPX_SHAREAS environment variable, spawn recognizes all of the environment variables that are recognized by the fork and exec callable services.
5. The fd_count parameter can be 0, which means that all file descriptors are inherited by the child.
6. The fd_count parameter is limited to a maximum value of 1000.

Example

In the following example, **/bin/ls** is run mapping its STDOUT and STDERR to file descriptors 4 and 5, which were previously opened in the exec, and STDIN is closed:

```
map.0=-1
map.1=4
map.2=5
parm.0=2
parm.1='/bin/ls'
parm.2='/'
'spawn /bin/ls 3 map. parm. _ _environment.'
```

spawnp

```
►►──spawnp──filename──fd_count──fd_map──arg_stem──env_stem──────────►◄
```

Function

spawnp invokes the spawn callable service and creates a new process, called a *child process*, to run a hierarchical file system (HFS) executable file. **spawnp** functions identically to the spawn function, except that it uses the **PATH** environment variable to resolve relative filenames.

See “spawn” on page 151 for more information.

stat

```
►►──stat──pathname──stem─────────────────────────────────────►◄
```

Function

stat invokes the stat callable service to obtain status about a specified file. You specify the file by its name. If the pathname specified refers to a symbolic link, the symbolic link name is resolved to a file and the status information for that file is returned. To obtain status information about a symbolic link, rather than the file it refers to, see “lstat” on page 105.

To use a file descriptor to obtain this information, see “fstat” on page 67.

Parameters

pathname
A pathname for the file.

stem
The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. You can use the predefined variables beginning with ST_ or their equivalent numeric values to access the values they represent. (See Appendix A for the numeric values.)

Variable	Description	
ST_AAUDIT	Auditor audit information.	
ST_ATIME	Time of last access.	
ST_AUDITID	RACF File ID for auditing.	
ST_BLKSIZE	File block size.	
ST_BLOCKS	Blocks allocated.	
ST_CCSID	Coded character set ID.	
ST_CRTIME	File creation time.	
ST_CTIME	Time of last file status change.	
ST_DEV	Device ID of the file.	
ST_EXTLINK	External symbolic link flag, set to 0 or 1.	
ST_FID	File identifier.	
ST_FILEFMT	Format of the file. To specify the format, you can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value:	
	S_FFBINARY	Binary data
	S_FFCR	Text data delimited by a carriage return character
	S_FFCRLF	Text data delimited by carriage return and line feed characters
	S_FFLF	Text data delimited by a line feed character
	S_FFLFCR	Text data delimited by a line feed and carriage return characters
	S_FFNA	Text data with the file format not specified
	S_FFNL	Text data delimited by a newline character
ST_GENVALUE	General attribute values.	
ST_GID	Group ID of the group of the file.	
ST_INO	File serial number.	
ST_MAJOR	Major number for a character special file.	
ST_MINOR	Minor number for a character special file.	

Variable	Description
ST_MODE	File mode, permission bits only.
ST_MTIME	Time of last data modification.
ST_NLINK	Number of links.
ST_RTIME	File backup time stamp (reference time).
ST_SETGID	Set Group ID on execution flag, set to 0 or 1.
ST_SETUID	Set User ID on execution flag, set to 0 or 1.
ST_SIZE	File size for a regular file, in bytes. If file size exceeds $2^{31}-1$ bytes, size is expressed in megabytes, using an M (for example, 3123M).
ST_STICKY	Sticky bit flag (keep loaded executable in storage), set to 0 or 1.
ST_TYPE	Numeric value that represents the file type for this file. You can use a numeric value (see Appendix A) or any of the predefined variables that begin with S_ to determine the file type: S_ISCHR Character special file; S_ISDIR Directory; S_ISFIFO FIFO special file; S_ISREG Regular file; S_ISSYM Symbolic link
ST_UAUDIT	Area for user audit information.
ST_UID	User ID of the owner of the file.

The stem variable *stem.st_type* is a number that represents the file type for this file. You can use the predefined variables beginning with S_ or their equivalent numeric values to determine the file type. For example, if *stem.st_type* is S_ISDIR, the file is a directory.

Usage Notes

All time fields in *stem* are in POSIX format. You can use **gmtime** to convert it to other forms.

Example

In the following example, assume that *path* was assigned a value earlier in the exec:

```
"stat (path) st."
```

statfs

```
►►──statfs──name──stem──────────────────────────────────────────►◄
```

Function

statfs invokes the statfs callable service to obtain status information about a specified file system.

Parameters

name

The name of the file system to be mounted, specified as the name of an HFS data set. You must specify the HFS data set name as a fully qualified name in uppercase letters. Do not enclose the data set name in single quotes.

stem

The name of a stem variable used to return the information. On return, *stem.0* contains the number of variables returned. You can use the predefined variables beginning with STFS_ or their equivalent numeric values to access the status values they represent. (See Appendix A for the numeric values.) For example, *stem.stfs_avail* accesses the number of blocks available in the file system.

Variable	Description
STFS_AVAIL	Space available to unprivileged users in block-size units.
STFS_BFREE	Total number of free blocks.
STFS_BLOCKSIZE	Block size.
STFS_FAVAIL	Number of free file nodes available to unprivileged users.
STFS_FFREE	Total number of free file nodes.
STFS_FILES	Total number of file nodes in the file system.
STFS_FRSIZE	Fundamental file system block size.
STFS_FSID	File system ID set by the logical file system.
STFS_INUSE	Allocated space in block-size units.
STFS_INVARSEC	Number of seconds the file system will remain unchanged.
STFS_NAMEMAX	Maximum length of file name.
STFS_NOSUID	SETUID and SETGID are not supported.
STFS_RDONLY	File system is read-only.
STFS_TOTAL	Total space in block-size units.

Example

In the following example, assume that *fsname* was assigned a value earlier in the exec:

```
"statfs (fsname) st."
```

statvfs

```
►►──statvfs──pathname──stem──────────────────────────────────────────►◄
```

Function

statvfs invokes the statvfs callable service to obtain status information about a file system, given the name of a file in the file system.

Parameters

pathname
The name of a file in a file system for which status information is to be obtained.

stem
The name of a stem variable used to return the information. On return, *stem.0* contains the number of variables returned. You can use the predefined variables beginning with STFS_ or their equivalent numeric values (see Appendix A) to access the status values they represent. For example, *stem.stfs_avail* accesses the number of blocks available in the file system.

Variable	Description
STFS_AVAIL	Space available to unprivileged users in block-size units.
STFS_BFREE	Total number of free blocks.
STFS_BLOCKSIZE	Block size.
STFS_FAVAIL	Number of free file nodes available to unprivileged users.
STFS_FFREE	Total number of free file nodes.
STFS_FILES	Total number of file nodes in the file system.
STFS_FRSIZE	Fundamental file system block size.
STFS_FSID	File system ID set by the logical file system.
STFS_INUSE	Allocated space in block-size units.
STFS_INVARSEC	Number of seconds the file system will remain unchanged.
STFS_NAMEMAX	Maximum length of file name.
STFS_NOSUID	SETUID and SETGID are not supported.
STFS_RDONLY	File system is read-only.
STFS_TOTAL	Total space in block-size units.

Example

In the following example, assume that *fsname* was assigned a value earlier in the exec:

```
"statvfs (fsname) st."
```

symlink

```
►►──symlink──pathname──linkname──────────────────────────────────────►◄
```

Function

symlink invokes the symlink callable service to create a symbolic link to a pathname. This creates a symbolic link file.

Parameters

pathname
A pathname for the file for which you are creating a symbolic link.

linkname
The pathname for the symbolic link.

Usage Notes

Like a hard link (described in “link” on page 102), a symbolic link allows a file to have more than one name. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link, however, provides no such assurance; in fact, the file identified by *pathname* need not exist when the symbolic link is created. In addition, a symbolic link can cross file system boundaries.

When a component of a pathname refers to a symbolic link rather than to a directory, the pathname contained in the symbolic link is resolved. If the pathname in the symbolic link begins with / (slash), the symbolic link pathname is resolved relative to the process root directory. If the pathname in the symbolic link does not begin with /, the symbolic link pathname is resolved relative to the directory that contains the symbolic link.

If the symbolic link is not the last component of the original pathname, remaining components of the original pathname are resolved from there. When a symbolic link is the last component of a pathname, it may or may not be resolved. Resolution depends on the function using the pathname. For example, a rename request does not have a symbolic link resolved when it appears as the final component of either the new or old pathname. However, an open request does have a symbolic link resolved when it appears as the last component. When a slash is the last component of a pathname, and it is preceded by a symbolic link, the symbolic link is always resolved.

Because the mode of a symbolic link cannot be changed, its mode is ignored during the lookup process. Any files and directories to which a symbolic link refers are checked for access permission.

Example

To create a symbolic link named **/bin** for the file **/v.1.1.0/bin**:

```
"symlink /v.1.1.0/bin /bin"
```

sysconf

```
►►──sysconf──name──────────────────────────────────────────►◄
```

Function

sysconf invokes the sysconf callable service to get the value of a configurable system variable.

Parameters

name

A numeric value that specifies the configurable variable to be returned. You can specify a numeric value (see Appendix A) or the predefined variable beginning with SC_ used to derive the appropriate numeric value:

Variable	Description
SC_ARG_MAX	The maximum length of all arguments and environment strings to exec()
SC_CHILD_MAX	The maximum number of simultaneous processes per real user ID
SC_CLK_TCK	The number of intervals per second used in defining the type clock_t, which is used to measure process execution times
SC_JOB_CONTROL	Support for job control
SC_NGROUPS_MAX	Maximum number of simultaneous supplementary group IDs per process
SC_OPEN_MAX	Maximum number of simultaneous open files per process
SC_SAVED_IDS	Support for saved set-user-IDs and set-group-IDs
SC_THREAD_TASKS_MAX_NP	Constant for querying the maximum number of threaded tasks per calling process
SC_THREADS_MAX_NP	Constant for querying the maximum number of threads per calling process
SC_TZNAME_MAX	The number of bytes supported for the name of a time zone
SC_VERSION	The integer value 199009L
SC_2_CHAR_TERM	Constant for querying whether the system supports at least one raw mode terminal

Usage Notes

SC_MAX_THREADS_NP and SC_MAX_THREAD_TASKS_NP return the limits defined for the caller's process, not the systemwide limits.

Example

To determine the maximum number of files that a single process can have open at one time:

```
"sysconf" sc_open_max
```

time

```
►►──time──────────────────────────────────────────────────────────►◄
```

Function

time returns in RETVAL the time in POSIX format (seconds since the Epoch, 00:00:00 on 1 January 1970). You can use **gmtime** to convert it to other forms.

times

```
►►──times──stem──────────────────────────────────────────────►◄
```

Function

times invokes the times callable service to collect information about processor time used by the current process or related processes. The elapsed time since the process was dubbed is returned in RETVAL. This value is of the type clock_t, which needs to be divided by sysconf(_SC_CLK_TK) to convert it to seconds. For OS/390 OpenEdition, this value is expressed in hundredths of a second.

Parameters

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. Four variables are returned in the stem. To access the stem variables, use a numeric value or the predefined variables beginning with TMS_ used to derive that numeric value. (For the numeric values, see Appendix A.) For example, you could specify *stem.4* or *stem.tms_cstime* to obtain system CPU values:

Variable	Description
TMS_CSTIME	The sum of system CPU time values and child system CPU time values for all waited-for child processes that have terminated. Zero if the current process has no waited-for children.
TMS_CUTIME	The sum of user CPU time values and child user CPU time values for all waited-for child processes that have terminated. Zero if the current process has no waited-for children.
TMS_STIME	The system CPU time of current process in hundredths of a second. This is the task control block (TCB) time accumulated while running in the kernel address space.
TMS_UTIME	The user CPU time of current process in hundredths of a second. This includes the TCB and service request block (SRB) time of the calling process minus the TCB time accumulated while running in the kernel address space.

Usage Notes

Processor times for a child process that has ended are not added to the TMS_CUTIME and TMS_CSTIME of the parent process until the parent issues a wait or waitpid for that child process.

Example

```
"times tm."
```

trunc

```
►►──trunc──pathname──file_size──────────────────────────────────────►◄
```

Function

trunc invokes the trunc callable service to change the size of the file identified by pathname.

Parameters

pathname
The pathname of the file.

file_size
The new size of the file, in bytes.

Usage Notes

1. The file specified must be a regular file to which the calling process has write access.
2. The file size changes beginning from the first byte of the file. If the file was previously larger than the new size, the data from *file_size* to the original end of the file is removed. If the file was previously shorter than *file_size*, bytes between the old and new lengths are read as zeros.
3. If *file_size* is greater than the current file size limit for the process, the request fails with EFBIG, and the SIGXFSZ signal is generated for the process.

Example

To set the file size of **/tmp/xxx** to 1000 bytes:

```
"trunc /tmp/xxx 1000"
```

ttyname

```
►►──ttyname──fd──variable──────────────────────────────────────►◄
```

Function

ttyname invokes the ttyname callable service to obtain the pathname of the terminal associated with the file descriptor.

Parameters

fd The file descriptor (a number) for the character special file for the terminal.

variable
The name of the variable that stores the pathname for the character special file for the terminal.

Usage Notes

This service does not return `−1` to indicate a failure. If the file descriptor is incorrect, a null string is returned.

Example

To obtain the pathname for file descriptor `0`:

```
"ttyname 0 path"
```

umask

```
►►──umask──mask──────────────────────────────────────────────►◄
```

Function

umask invokes the umask callable service to change your process's file mode creation mask. The file mode creation mask is used by the security package (RACF) to turn off permission bits in the mode parameter specified. Bit positions that are set in the file mode creation mask are cleared in the mode of the created file.

Parameters

mask

A permission bit mask that you specify as a three-digit number. Each digit must be in the range `0` to 7, and all digits must be specified. For more information on permissions, see Appendix B.

Usage Notes

1. The umask service changes the process's file creation mask. This mask controls file permission bits that are set whenever the process creates a file. File permission bits that are turned on in the file creation mask are turned off in the file permission bits of files created by the process. For example, if a call to the open service, BPX1OPN, specifies a *mode* argument with file permission bits, the process's file creation mask affects that argument: bits that are on in the mask will be turned off in the *mode* argument, and therefore in the mode of the created file.
2. Only the file permission bits of new mask are used.

Example

To create a mask that sets read-write-execute permission on for the owner of the file and off for everyone else:

```
"umask 077"
```

uname

```
►►──uname──stem──────────────────────────────────────────────────►◄
```

Function

uname invokes the uname callable service to obtain information about the OS/390 OpenEdition system you are running on.

Parameters

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To access the information, you can specify a numeric value (see Appendix A) or the predefined variables beginning with U_ that derive the appropriate numeric value. For example, both *stem.1* and *stem.u_sysname* access the name of the operating system.

Variable	**Description**
U_MACHINE	The name of the hardware type on which the system is running.
U_NODENAME	The name of this node within the communication network.
U_RELEASE	The current release level of this implementation.
U_SYSNAME	The name of this implementation of the operating system (OS/390).
U_VERSION	The current version level of this release.

Example

```
"uname uts."
```

unlink

```
►►──unlink──pathname──────────────────────────────────────────►◄
```

Function

unlink invokes the unlink callable service to remove a directory entry.

Parameters

pathname
A pathname for the directory entry. The directory entry could be identified by a pathname for a file, the name of a hard link to a file, or the name of a symbolic link.

Usage Notes

1. If the name specified refers to a symbolic link, the symbolic link file named by *pathname* is deleted.
2. If a file is deleted (that is, if the unlink service request is successful and the link count becomes zero), the contents of the file are discarded, and the space it occupied is freed for reuse. However, if another process (or more than one) has the file open when the last link is removed, the file is not removed until the last process closes it.
3. When the unlink service is successful in removing the directory entry and decrementing the link count, whether or not the link count becomes zero, it returns control to the caller with RETVAL set to `0`. It updates the change and modification times for the parent directory, and the change time for the file itself (unless the file is deleted).
4. Directories cannot be removed using unlink. To remove a directory, refer to “rmdir” on page 131.

Example

In the following example, assume that *file* was assigned a value earlier in the exec:

```
"unlink (file)"
```

unmount

```
►►──unmount──name──flags──────────────────────────────────────────►◄
```

Function

unmount invokes the umount callable service to unmount a file system; that is, it removes a file system from the file hierarchy. You must be a superuser to unmount a file system.

Parameters

name

The name of the file system to be unmounted, specified as the name of an HFS data set. You must specify the HFS data set name as a fully qualified name in uppercase letters. Do not enclose the data set name in single quotes.

flags

The unmount options, expressed as a numeric value. You can specify a numeric value (see Appendix A) or the predefined variable beginning with MTM_ used to derive the appropriate numeric value:

Variable	Description
MTM_DRAIN	An unmount drain request. All uses of the file system are normally ended before the file system is unmounted.
MTM_FORCE	An unmount force request. The file system is unmounted immediately, forcing any users of the named file system to fail. All data changes made up to the time of the request are saved. If there is a problem saving data, the unmount continues and the data may be lost. So that data will not be lost, you must issue an unmount immediate request before an unmount force request.
MTM_IMMED	An unmount immediate request. The file system is unmounted immediately, forcing any users of the named file system to fail. All data changes made up to the time of the request are saved. If there is a problem saving data, the unmount request fails.
MTM_NORMAL	A normal unmount request. If no one is using the named file system, the unmount request is done. Otherwise, the request is rejected.
MTM_RESET	A reset unmount request. This stops a previous unmount drain request.
MTM_REMOUNT	Unmounts the file system, changes the mount mode, and remounts the file system. A read/write mount mode changes to read-only. A read-only mount mode changes to read/write.

Usage Notes

1. A file system that has file systems mounted on it cannot be unmounted. Any child file systems must be unmounted first.
2. A reset request can stop only an unmount service drain request. There is no effect if it is issued when there is no umount request outstanding.

Example

To request a normal unmount of the file system HFS.USR.CRISP:

```
"unmount HFS.USR.CRISP" mtm_normal
```

unquiesce

```
►►──unquiesce──name──flag──────────────────────────────────────►◄
```

Function

unquiesce invokes the unquiesce callable service to unquiesce a file system, making the files in it available for use again. You must be a superuser to use this function.

Parameters

name

The name of the file system to be unquiesced. You must specify an HFS data set name as a fully qualified name in uppercase letters. Do not enclose the data set name in single quotes.

flag

A number specifying the type of unquiesce:

0	Normal unquiesce
1	Forced unquiesce. Request is allowed even if the requester process is not the process that made the quiesce request.

Usage Notes

An unquiesce service makes a file system available for use again following a previous quiesce request.

Example

To request a normal unquiesce of the file system HFS.USR.ELIZAB:

```
"unquiesce HFS.USR.ELIZAB 0'
```

utime

```
►►──utime──pathname──atime──mtime──────────────────────────────────►◄
```

Function

utime invokes the utime callable service to set the access and modification times of a file.

Parameters

pathname
A pathname for the file.

atime
A numeric value for the new access time for the file, specified as POSIX time (seconds since the Epoch, 00:00:00 1 January 1970).

mtime
A numeric value for the new modification time for the file, specified as POSIX time (seconds since the Epoch, 00:00:00 1 January 1970).

Example

In the following example, assume that *file*, *atm*, and *mtm* were assigned values earlier in the exec:

```
"utime (file)" atm mtm
```

wait

```
►►──wait──stem──────────────────────────────────────────────────►◄
```

Function

wait invokes the wait callable service to obtain the status of any child process that has ended or stopped. You can use the wait service to obtain the status of a process that is being debugged with the ptrace facilities. The term *child* refers to a child process created by a fork as well as a process attached by ptrace.

Parameters

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To access the information in the stem variables, you can specify a numeric value (see Appendix A) or the predefined variables beginning with W_ that derive the appropriate numeric value.

Variable	Description
W_CONTINUED	Process continued from stop.
W_EXITSTATUS	The exit status of the child process.
W_IFEXITED	The child process ended normally.
W_IFSIGNALED	The child process ended because of a signal that was not caught.
W_IFSTOPPED	Wait if the child process is stopped.
W_STAT3	Byte 3 of the BPXYWAST macro, documented in *OS/390 OpenEdition Programming: Assembler Callable Services Reference* .
W_STAT4	Byte 4 of the BPXYWAST macro, documented in *OS/390 OpenEdition Programming: Assembler Callable Services Reference* .
W_STOPSIG	The signal number that caused the child process to stop.
W_TERMSIG	The signal number that caused the child process to end.

Usage Notes

1. The wait service suspends execution of the calling thread until one of the requested child or debugged processes ends or until it obtains information about the process that ended. If a child or debugged process has already ended but its status has not been reported when wait is called, the routine immediately returns with that status information to the caller.
2. If the WUNTRACED option is specified, the foregoing also applies for stopped children or stopped debugged processes.
3. The wait service always returns status for stopped *debugged* processes, even if WUNTRACED is not specified.
4. If status is available for one or more processes, the order in which the status is reported is unspecified.

Note: A debugged process is one that is being monitored for debugging purposes with the ptrace service.

Example

See “Set Up a Signal to Enforce a Time Limit for a Program” on page 194 for an example of signal coding that interprets the stem this returns:

```
"wait wstat."
```

waitpid

```
►►──waitpid──pid──stem──option─────────────────────────────────────►◄
```

Function

waitpid invokes the wait callable service to obtain the status of a child process that has ended or stopped. You can use the wait service to obtain the status of a process that is being debugged with the ptrace facilities. The term *child* refers to a child process created by a fork as well as a process attached by ptrace.

Parameters

pid

A numeric value indicating the event the caller is waiting upon:

- A value greater than zero is assumed to be a process ID. The caller waits for the child or debugged process with that specific process ID to end or to stop.
- A value of zero specifies that the caller is waiting for any children or debugged processes with a process group ID equal to the caller's to end or to stop.
- A value of `-1` specifies that the caller is waiting for any of its children or debugged processes to end or to stop.
- If the value is negative and less than `-1`, its absolute value is assumed to be a process group ID. The caller waits for any children or debugged processes with that process group ID to end or to stop.

stem

The name of a stem variable used to return the information. Upon return, *stem.0* contains the number of variables returned. To access the information in the stem variables, you can use numeric values (see Appendix A), or the predefined variables beginning with W_ (see their description in "wait" on page 172).

options

A numeric value or its equivalent predefined variable beginning with W_ that indicates the wait options for this invocation of the wait service. These options can be specified separately or together. (For the numeric values, see Appendix A.)

Variable	Description
`0`	Wait for the child process to end (default processing)
W_NOHANG	The wait service does not suspend execution of the calling process if status is not immediately available for one of the child processes specified by Process_ID.
W_UNTRACED	The wait service also returns the status of any child processes specified by Process_ID that are stopped, and whose status has not yet been reported since they stopped. If this option is not specified, the wait service returns only the status of processes that end.

Usage Notes

1. Use **waitpid** when you want to wait for a specified child process. The *pid* argument specifies a set of child processes for which status is requested; **waitpid** returns the status of a child process from this set.
2. The waitpid service suspends execution of the calling thread until one of the requested child or debugged processes ends or until it obtains information about the process that ended. If a child or debugged process has already ended but its status has not been reported when waitpid is called, the routine immediately returns with that status information to the caller.
3. If the WUNTRACED option is specified, the foregoing also applies for stopped children or stopped debugged processes. A debugged process is one that is being monitored for debugging purposes with the ptrace service.
4. The wait service always returns status for stopped *debugged* processes, even if WUNTRACED is not specified.
5. If status is available for one or more processes, the order in which the status is reported is unspecified.

Example

See “Set Up a Signal to Enforce a Time Limit for a Program” on page 194 for an example of signal coding that interprets the stem this returns:

```
"waitpid -1 wst. 0"
```

write

```
►►──write──fd──variable──┬────────┬──────────────────────────────►◄
                         └─length─┘
```

Function

write invokes the write callable service to copy data to a buffer and then write it to an open file. The number of bytes written is returned in RETVAL.

Parameters

fd The file descriptor (a number) for a file.

variable
The name of the variable that is to store the data to be written to the file.

length
The number of bytes to be written to the file identified by *fd*. If you want a length longer than 4096 bytes, specify the *length* parameter. If you do not specify *length*, the length of *variable* is used, up to a maximum length of 4096 bytes. A *variable* longer than 4096 bytes is truncated to 4096, and RC is set to 4.

Usage Notes

1. Within the variable, you can use the predefined variables beginning with ESC_ the same way you use C language escape sequences to avoid code page dependence with some control characters. For example:

   ```
   buf='line 1' || esc_n
   ```

   appends a newline character to the string `'line 1'`.

ESC_A	Alert (bell)
ESC_B	Backspace
ESC_F	Form feed (new page)
ESC_N	Newline
ESC_R	Carriage return
ESC_T	Horizontal tab
ESC_V	Vertical tab

2. Return codes:

   - 4 indicates one of these:

     – If *length* was specified, the number of characters specified by *length* is not the same as the length of *variable*. The data is truncated or padded as required. The characters used for padding are arbitrarily selected.

     – If *length* was not specified, 4 indicates that a variable longer than 4096 bytes was truncated to 4096.

   - –24 indicates that storage could not be obtained for the buffer.

3. **File Offset:** If *fd* specifies a regular file or any other type of file on which you can seek, the write service begins writing at the file offset associated with that file descriptor. A successful write operation increments the file offset by the number of bytes written. If the incremented file offset is greater than the previous length of the file, the file is extended: the length of the file is set to the new file offset.

If the file descriptor refers to a file on which you cannot seek, the service begins writing at the current position. No file offset is associated with such a file.

If the file was opened with the “append” option, the write routine sets the file offset to the end of the file before writing output.

4. **Number of Bytes Written:** Ordinarily, the number of bytes written to the output file is the number you specify in the *length* parameter. (This number can be zero. If you ask to write zero bytes, the service simply returns a return value of zero, without attempting any other action.)

   If the *length* you specify is greater than the remaining space on the output device, fewer bytes than you requested are written. When at least 1 byte is written, the write is considered successful. The return value shows the number of bytes written. An attempt to write again to the same file, however, causes an ENOSPC error unless you are using a pseudoterminal. With a pseudoterminal, if there is not enough room in the buffer for the whole write, the number of bytes that can fit are written, and the number of bytes written is returned. However, on the next write (assuming the buffer is still full) there is a block or EAGAIN is returned, depending on whether the file was opened blocking or nonblocking.

   Similarly, fewer bytes are written if the service is interrupted by a signal after some but not all of the specified number of bytes are written. The return value shows the number of bytes written. If no bytes were written before the routine was interrupted, the return value is −1 and an EINTR error is reported.

5. The write service causes signal **SIGTTOU** to be sent under all the following conditions:

   - The process is attempting to write to its controlling terminal.
   - TOSTOP is set as a terminal attribute.
   - The process is running in a background process group.
   - The **SIGTTOU** signal is not blocked or ignored.
   - The process is not an orphan.

   If all the conditions are met, **SIGTTOU** is sent.

6. Write requests to a pipe (FIFO) are handled in the same as write requests to a regular file, with the following exceptions:

   - There is no file offset associated with a pipe; each write request appends to the end of the pipe.

   - If the size of the write request is less than or equal to the value of the PIPE_BUFF variable (described in the pathconf service), the write is guaranteed to be atomic. The data is not interleaved with data from other write processes on the same pipe. If the size of the write request is greater than the value of PIPE_BUFF, the data can be interleaved, on arbitrary boundaries, with writes by other processes, whether or not the O_NONBLOCK flag is set.

Example

In the following example, assume that *fd* and *buf* were assigned values earlier in the exec:

```
"write" fd "buf"
```

writefile

```
►►──writefile──pathname──mode──stem──────────────────────────────►◄
```

Function

writefile invokes the open, write, and close callable services to write text files, with lines up to 1024 characters long.

Parameters

pathname
A pathname for the file.

mode
A three- or four-digit number, corresponding to the access permission bits. Each digit must be in the range 0–7, and at least three digits must be specified. For more information on permissions, see Appendix B.

stem
The name of a stem variable used to return the information written to the file. Upon return, *stem.0* contains the number of variables that are written. *stem.1* through *stem.n*, where *n* is the total number of variables written, each contain a line to be written. A newline is written to the file following each line.

Within the stem, you can use the predefined variables beginning with ESC_ the same way you use C language escape sequences to avoid code page dependence with some control characters. See the usage notes for “write” on page 176.

Usage Notes

File I/O stops when **writefile** sets a return code. **writefile** can set the following return codes:

4	A line is longer than 1024 characters.
8	A write was attempted, but failed. RETVAL, ERRNO, and ERRNOJR contain the return values from the write callable service.

For further usage notes, see:

- “close” on page 34
- “open” on page 111
- “write” on page 176

Example

In the following example, assume that *fname* and the stem *file.* were assigned values earlier in the exec:

```
"writefile (fname) 600 file."
```

Chapter 4. Examples: REXX Coding with Syscall Commands

The examples in this chapter are provided to assist you with coding REXX programs that use OS/390 OpenEdition syscall commands.

The first three examples can run in TSO/E, batch, or from the shell; they begin with the call **syscalls('ON')**.

Read the Root Directory into a Stem and Print It

This example prints the contents of the root directory to standard output:

```
/*   rexx   */
call syscalls 'ON'
address syscall
'readdir / root.'
do i=1 to root.0
   say root.i
end
```

The following line saves the results from the previous example in a text file:

```
   'writefile /u/schoen/root.dir 777 root.'
```

Open, Write, and Close a File

```
/*   rexx   */
call syscalls 'ON'
address syscall
path='/u/schoen/my.file'
'open' path,
       O_rdwr+O_creat+O_trunc,
       660
if retval=-1 then
   do
say 'file not opened, error codes' errno errnojr
   return
   end
fd=retval
rec='hello world' || esc_n
'write' fd 'rec' length(rec)
if retval=-1 then
   say 'record not written, error codes' errno errnojr
'close' fd
```

Open a File, Read from It, and Close It

```
   /*   rexx   */
call syscalls 'ON'
address syscall
path='/u/schoen/my.file'
'open  (path)',
       O_rdonly,
       000
if retval=-1 then
   do
   say 'file not opened, error codes' errno errnojr
   return
   end
fd=retval
'read' fd 'bytes 80'
if retval=-1 then
   say 'bytes not read, error codes' errno errnojr
else
   say bytes
'close' fd
```

Display the Working Directory and List a Specified Directory

This REXX program runs in the shell; it uses both the SH and SYSCALL environments. The program identifies your working directory and lists the contents of a directory that you specify as a parameter or after a prompt.

```
/*   rexx  */
parse arg dir
address syscall 'getcwd cwd'
say 'current directory is' cwd
if dir=' then
   do
   say 'enter directory name to list'
   parse pull dir
   end
'ls -l' dir
return
```

Parse Arguments Passed to a REXX program: The getopts Function

The following simple utility function is used by some of the examples to parse the arguments to a REXX program that is run from the shell. This function parses the stem `__argv` for options in the format used by most POSIX commands.

```
/*********************************************************************/
/* Function: getopts                                                 */
/*  This function parses __argv. stem for options in the format      */
/*  used by most POSIX commands.  This supports simple option        */
/*  letters and option letters followed by a single parameter.       */
/*  The stem OPT. is setup with the parsed information.  The         */
/*  options letter in appropriate case is used to access the         */
/*  variable:  op='a'; if opt.op=1 then say 'option a found'         */
/*  or, if it has a parameter:                                       */
/*     op='a'; if opt.op<>'' then say 'option a has value' opt.op    */
/*                                                                   */
/* Parameters: option letters taking no parms                        */
/*             option letters taking 1 parm                          */
/*                                                                   */
/* Returns: index to the first element of __argv. that is not        */
/*          an option.  This is usually the first of a list of files.*/
/*          A value of 0 means there was an error in the options and */
/*          a message was issued.                                    */
/*                                                                   */
/* Usage:  This function must be included in the source for the exec */
/*                                                                   */
/* Example:  the following code segment will call GETOPTS to parse   */
/*           the arguments for options a, b, c, and d.  Options a    */
/*           and b are simple letter options and c and d each take   */
/*           one argument.  It will then display what options were   */
/*           specified and their values.  If a list of files is      */
/*           specified after the options, they will be listed.       */
/*                                                                   */
/*   parse value 'a   b   c   d' with,                               */
/*               lca lcb lcc lcd .                                   */
/*   argx=getopts('ab','cd')                                         */
/*   if argx=0 then exit 1                                           */
/*   if opt.0=0 then                                                 */
/*      say 'No options were specified'                              */
/*    else                                                           */
/*      do                                                           */
/*      if opt.lca<>'' then say 'Option a was specified'             */
/*      if opt.lcb<>'' then say 'Option b was specified'             */
/*      if opt.lcc<>'' then say 'Option c was specified as' opt.lcc  */
/*      if opt.lcd<>'' then say 'Option d was specified as' opt.lcd  */
/*      end                                                          */
/*   if __argv.0>=argx then                                          */
/*      say 'Files were specified:'                                  */
/*    else                                                           */
/*      say 'Files were not specified'                               */
/*   do i=argx to __argv.0                                           */
/*      say __argv.i                                                 */
/*   end                                                             */
/*                                                                   */
/*********************************************************************/
```

```
getopts: procedure expose opt. __argv.
   parse arg arg0,arg1
   argc=__argv.0
   opt.='
   opt.0=0
   optn=0
   do i=2 to argc
      if substr(__argv.i,1,1)<>'-' then leave
      if __argv.i='-' then leave
         opt=substr(__argv.i,2)
      do j=1 to length(opt)
         op=substr(opt,j,1)
         if pos(op,arg0)>0 then
            do
            opt.op=1
            optn=optn+1
            end
         else
         if pos(op,arg1)>0 then
            do
            if substr(opt,j+1)<>' then
               do
               opt.op=substr(opt,j+1)
               j=length(opt)
               end
             else
               do
               i=i+1
               if i>argc then
                  do
                  say 'Option' op 'requires an argument'
                  return 0
                  end
               opt.op=__argv.i
               end
            optn=optn+1
            end
         else
            do
            say 'Invalid option =' op
            say 'Valid options are:' arg0 arg1
            return 0
            end
      end
   end
   opt.0=optn
   return i
```

Count Newlines, Words, and Bytes

This is an example of a REXX program that can run as a shell command or filter. It is a REXX implementation of the **wc** (word count) utility supporting the options –c, –w, and –l.

The program uses **open**, **close**, and EXECIO. To read standard input, it accesses the file **/dev/fd0**.

```
/* rexx */
parse value 'l   w   c' with,
             lcl lcw lcc .         /* init lower case access vars    */
argx=getopts('lwc')                /* parse options                  */
if argx=0 then return 1            /* return on error                */
if opt.0=0 then                    /* no opts, set defaults          */
   parse value '1       1       1' with,
                opt.lcl opt.lcw opt.lcc .
if __argv.0>argx then              /* multiple files specified       */
   single=0
else
if __argv.0=argx then              /* one file specified             */
   single=1
else
   do                              /* no files specified, use stdin  */
   single=2
   __argv.argx='/dev/fd0'          /* handle it like fd0 specified   */
   __argv.0=argx
   end
parse value '0   0   0' with,
             twc tcc tlc .         /* clear total counters           */
address syscall
do i=argx to __argv.0              /* loop through files             */
   fi=__argv.i                     /* get file name                  */
   parse value '0  0  0' with,
                wc cc lc .         /* clear file counters            */
   'open (fi)' o_rdonly 000        /* open the file                  */
   fd=retval
   if fd=-1 then
      do                           /* open failed                    */
      say 'unable to open' fi
      iterate
      end
   do forever                      /* loop reading 1 line at a time  */
      address mvs 'execio 1 diskr' fd '(stem LN.'
      if rc<>0 | ln.0=0 then leave  /* error or end of file          */
      if opt.lcw=1 then
         wc=wc+words(ln.1)         /* count words in line            */
      if opt.lcc=1 then
         cc=cc+length(ln.1)+1      /* count chars in line + NL char  */
      if opt.lcl=1 then
         lc=lc+1                   /* count lines                    */
   end
   'close' fd                      /* close file                     */
   twc=twc+wc                      /* accumulate total words         */
   tlc=tlc+lc                      /* accumulate total lines         */
   tcc=tcc+cc                      /* accumulate total chars         */
   if opt.lcw<>1 then wc=''        /* format word count              */
                 else wc=right(wc,7)
   if opt.lcl<>1 then lc=''        /* format line count              */
                 else lc=right(lc,7)
   if opt.lcc<>1 then cc=''        /* format char count              */
                 else cc=right(cc,7)
   if single=2 then fi=''          /* if stdin used clear filename   */
   say lc wc cc '  'fi              /* put out counts message        */
end
if single=0 then                    /* if multiple files specified    */
   do                               /* format and output totals line  */
   if opt.lcw<>1 then twc='
                 else twc=right(twc,7)
   if opt.lcl<>1 then tlc='
                 else tlc=right(tlc,7)
   if opt.lcc<>1 then tcc='
                 else tcc=right(tcc,7)
   say tlc twc tcc '  total'
   end
return 0
```

Obtain Information about the Mounted File System

This REXX program uses **getmntent** and **statfs** to list all mount points, the name of the mounted file system, and the available space in the file system.

```
/* rexx */
address syscall
numeric digits 12
'getmntent m.'
do i=1 to m.0
   'statfs' m.mnte_fsname.i 's.'
   j=s.stfs_avail * s.stfs_blocksize
   if length(m.mnte_path)>20 then
      say m.mnte_path.i'    'strip(m.mnte_fsname.i) '('j')'
    else
      say left(m.mnte_path.i,20) strip(m.mnte_fsname.i) '('j')'
end
```

Mount a File System

This REXX program uses **mount** to mount a file system, an action that requires superuser authority. The name of this program is **mountx**, and it is in **/samples**.

The syntax is:

```
 mountx pathname  fsn options
```

where *pathname* is the name of the directory where the file system is to be mounted, and *fsn* is the name of the HFS data set. The *options* are:

-p parm	Parameter data, in the form of a single string.
-r	Mount file system as read-only.
-t type	File system type (for example, HFS). The default file system type to HFS. If the file system type is HFS, the program changes the file system name to uppercase.

The pathname and the file system name can be specified in any order. **stat** is used to determine which name is the pathname.

For the **getopts** function called in this program, see “Parse Arguments Passed to a REXX program: The getopts Function” on page 182.

```
/* rexx */
/*********************************************************************/
/* Determine which options were specified.                           */
/*********************************************************************/
lcp='p'
lcr='r'
lct='t'
ix=getopts('r','pt')
/*********************************************************************/
/* If -r specified, mount file system read-only.                     */
/*********************************************************************/
if opt.lcr<>' then
   mtm=mtm_rdonly
 else
   mtm=mtm_rdwr
/*********************************************************************/
/* If -t 'name' specified, direct mount request to file system 'name'.*/
/* Otherwise, direct mount request to file system named HFS.         */
/*********************************************************************/
if opt.lct<>' then
   type=translate(opt.lct)
 else
   type='HFS'
```

```
/*******************************************************************/
/* Complain if required parameters are missing.                    */
/*******************************************************************/
if __argv.0<>ix+1 then
   do
   say 'Pathname and file system name are required, and '
   say 'they must follow the options: MOUNT <options> <pathname> <fsn>'
   return 1
   end
/*******************************************************************/
/* Direct function calls to REXX OpenEdition services.             */
/*******************************************************************/
address syscall
/*******************************************************************/
/* Determine which of the parameters is the mount point name       */
/* (it must be a pathname), and which is the file system to be     */
/* mounted.                                                        */
/*******************************************************************/
'stat (__argv.ix) st.'
if st.st_type<>s_isdir then
   do
   fsn=__argv.ix
   ix=ix+1
   path=__argv.ix
   end
 else
   do
   path=__argv.ix
   ix=ix+1
   fsn=__argv.ix
   end
'stat (path) st.'
if st.st_type<>s_isdir then
   do
   say "Can't figure out pathname, neither name is a directory:"
   say path
   say fsn
   return 1
   end
```

```
/*********************************************************************/
/* HFS file system requires mounted file systems to be data sets,    */
/* so translate the file system name to upper case.                  */
/*********************************************************************/
if type='HFS' then
   fsn=translate(fsn)
/*********************************************************************/
/* Mount the file system.                                            */
/* Return code  Return code  Meaning                                 */
/* from system  to caller                                            */
/*      0          0         Success, file system now mounted        */
/*      1          2         Success, file system mount in progress  */
/*     -1          1         Error, explained by ERRNO and reason code*/
/*********************************************************************/
"mount (path) (fsn) (type) (mtm) (opt.lcp)"
select
 when retval= 0 then
   do
   say fsn 'is now mounted at'
   say path
   return 0
   end
 when retval= 1 then
   do
   say fsn 'will be mounted asynchronously at'
   say path
   return 2
   end
 otherwise
   do
   say 'Mount failed:'
   say '  Error number was' errno'x('x2d(errno)')'
   say '  Reason code was ' right(errnojr,8,0)
   return 1
   end
end
```

Unmount a File System

This REXX program uses **unmount** to unmount a file system, an action that requires superuser authority. The name of this program is **unmountx**, and it is in **/samples**.

The syntax is:

unmountx *name*

or

unmountx -*t filesystype*

where:

- *name* is the pathname where the file system is mounted, or the name of an HFS data set.
- *filesystype* is the type name of the physical file system (PFS).

The program assumes that the case of the requested file system name is entered correctly in uppercase. If the unmount fails, it folds the file system name to uppercase and retries the unmount.

```
/*******************************************************************/
/* Direct commands to REXX OpenEdition MVS services.               */
/*******************************************************************/

address syscall

/*******************************************************************/
/* Verify that exactly one operand or a pfs type was specified.    */
/*******************************************************************/
if __argv.0=3 & __argv.2='-t' then
   return fstype(__argv.3)
if __argv.0<>2 then
   do
   say 'Syntax:   unmount <name>   or   unmount -t <filesystype>'
   return 2
   end
/*******************************************************************/
/* Determine if the name is a pathname.  If so, determine file system */
/* name via stat().  Otherwise, use the name as entered.           */
/*******************************************************************/
'stat (__argv.2) st.'
if retval =0 & st.st_type=s_isdir then
   do
   getmntent mnt. x2d(st.st_dev)
   fsn=mnt.mnte_fsname.1
   end
else
   fsn=__argv.2
```

```
/**********************************************************************/
/* Unmount the file system, trying both the name as entered           */
/* and the name uppercased, since the HFS file system requires        */
/* mounted file systems to be data sets.                              */
/*                                                                    */
/* Return code  Return code  Meaning                                  */
/* from system  to caller                                             */
/*  Not -1         0         Success, file system unmount complete    */
/*      -1         1         Error, explained by ERRNO and reason code*/
/**********************************************************************/
"unmount (fsn)" mtm_normal      /* unmount name as specified          */
if retval⅛=0 then
   do
   fsn=translate(fsn)           /* if fails, upcase name and retry    */
   "unmount (fsn)" mtm_normal
   end
if retval<>-1 then
   do
   say 'Unmount complete for' fsn
   return 0
   end
else
   do
   say 'Unmount failed:'
   say '  Error number was' errno'x('x2d(errno)')'
   say '  Reason code was ' right(errnojr,8,0)
   return 1
   end
/**********************************************************************/
/* Unmount all file systems for a PFS type                            */
/**********************************************************************/
fstype:
   arg name .                       /* make upper case                */
   /*******************************************************************/
   /* Loop through mount table until unable to do any unmounts.       */
   /* This handles cascaded mounts.                                   */
   /*******************************************************************/
   do until didone=0
      didone=0
      'getmntent m.'
      do i=1 to m.0
         if m.mnte_fstype.i=name then
            do
            "unmount (m.mnte_fsname.i)" mtm_normal
            if retval<>-1 then
               do
               didone=1
               say 'unmounted:' m.mnte_fsname.i
               end
            end
      end
   end
   /*****************************************************************/
   /* Make one more pass and show errors for failed unmounts         */
   /*****************************************************************/
   goterr=0
   'getmntent m.'
   do i=1 to m.0
      if m.mnte_fstype.i=name then
         do
         "unmount (m.mnte_fsname.i)" mtm_normal
         if retval=-1 then
            do
            say 'Unmount failed for' m.mnte_fsname.i':'
            say '  Error number was' errno'x('x2d(errno)')'
            say '  Reason code was ' right(errnojr,8,0)
            goterr=1
            end
         else
            say 'unmounted:' m.mnte_fsname.i
         end
   end
   return goterr
```

Run a Shell Command and Read Its Output into a Stem

This REXX program runs the **ls** shell command and reads the output from the command into a stem. The program uses **pipe**, **close**, and EXECIO. It accesses **/dev/fd***n*, where *n* is a number that the exec concatenates to **/dev/fd**.

| **Note:** You can use this example to trap output from commands when the output is
| relatively small (less than 100KB). For command output that could be larger,
| you should use the spawn service.

```
/* rexx */
address syscall 'pipe p.'                  /* make a pipe           */
'ls>/dev/fd' || p.2                        /* run the ls command and
                                              redirect output to the
                                              write end of the pipe */
address syscall 'close' p.2                /* close output side     */
address mvs 'execio * diskr' p.1 '(stem s.'  /* read data in pipe     */
do i=1 to s.0                              /* process the data      */
   say s.i
end
```

Print the Group Member Names

This REXX program uses **getgrgid**, **getgrgnam**, and **write** to print the names of the users that are connected to a group. The group can be specified as either a GID or a group name.

```
/* rexx */
arg group .
address syscall
if datatype(group,'W')=1 then
   'getgrgid (group) gr.'   /* use getgrgid if GID specified          */
 else
   'getgrnam (group) gr.'   /* use getgrgnam if group name specified  */
if retval<=0 then           /* check for error                        */
   do
   say 'No information available for group' group
   return 1
   end
say 'Connected users for group' strip(gr.gr_name)'('gr.gr_gid')'
j=1
do i=gr_mem to gr.0
   buf='   ' gr.i
   if j//5=0 | i=gr.0 then
      buf=buf || esc_n      /* write newline every 5 groups or at end */
   j=j+1
   'write 1 buf'
end
return 0
```

Obtain Information about a User

This REXX program prints information from the user database for a user. The user can be specified as either a UID or a user name. The program uses **getpwuid**, **getpwnam**, **getgrgid**, **getgroupsbyname**, and **write**.

```
/* rexx */
arg user .
address syscall
if datatype(user,'W')=1 then
   'getpwuid (user) pw.'    /* use getpwuid if UID specified         */
 else
   'getpwnam (user) pw.'    /* use getpwnam if user name specified   */
if retval<=0 then           /* check for error                       */
   do
   say 'No information available for user' user
   return 1
   end
say 'Information for user' strip(pw.pw_name)'('pw.pw_uid')'
say '  Home directory: ' strip(pw.pw_dir)
say '  Initial program:' strip(pw.pw_shell)
'getgrgid' pw.pw_gid 'gr.'
if retval<=0 then           /* check for error                       */
   do
   say '  Group information not available'
   return 1
   end
say '  Primary group:  ' strip(gr.gr_name)'('gr.gr_gid')'
'getgroupsbyname' pw.pw_name 's.'
if retval<=0 then           /* check for error                       */
   do
   say '  Supplemental group information not available'
   return 1
   end
say '  Supplemental GIDs:'
do i=1 to s.0
   buf=right(s.i,12)
   if i//5=0 | i=s.0 then
      buf=buf || esc_n      /* write newline every 5 groups or at end */
   'write 1 buf'
end
return 0
```

Set Up a Signal to Enforce a Time Limit for a Program

This REXX program runs **/bin/ls** to list files in the **/bin** directory, and sets up a signal that enforces a time limit of 10 seconds for the program to run:

```
/* REXX                                                      */
address syscall
/* initialize file descriptor map (see note 1)               */
fd.0=-1
fd.2=-1
'creat /tmp/dirlist 600'
fd.1=retval
/* initialize parameter stem  (see note 2)                   */
parm.1='/bin/ls'
parm.2='-l'
parm.3='/bin'
parm.0=3
/* spawn new process    (see note 3)                         */
'spawn /bin/ls 3 fd. parm. __environment.'
pid=retval
/* set up signals to wait up to 10 seconds   (see note 4)    */
call syscalls 'SIGON'
'sigaction' sigalrm sig_cat 0 'oldh oldf'
'sigprocmask' sig_unblock sigaddset(sigsetempty(),sigalrm) 'mask'
'alarm' 10                  /* set alarm                     */
'waitpid (pid) st. 0'       /* wait for process term or alarm */
srv=retval
'alarm 0'                   /* make sure alarm is now off    */
if srv=-1 then              /* if alarm went off             */
   do
   'kill' pid sigkill       /* cancel child process          */
   'waitpid (pid) st. 0'    /* wait for completion           */
   end
call syscalls 'SIGOFF'      /* turn off signals              */
/* determine process status code  (see note 5)               */
select
   when st.w_ifexited then
      say 'exited with code' st.w_exitstatus
   when st.w_ifsignaled then
      say 'terminated with signal' st.w_termsig
   when st.w_ifstopped then
      say 'stopped by signal' st.w_stopsig
end
return
sigsetempty: return copies(0,64)
sigaddset: return overlay(1,arg(1),arg(2))
```

Notes:

1. The file descriptor map is set up so that the file descriptor for the new file being created is remapped to file descriptor 1 (standard output) for the new process. File descriptors 0 and 2 will not be opened in the new process.
2. The first parameter is set to the pathname for the file being spawned. Additional parameters are set in the format the program expects. In this case, they specify a long directory listing for the **/bin** directory.
3. The new process is spawned to run **/bin/ls**. File descriptors greater than or equal to 3 are not available to the new process. *fd.0* to *fd.2* are used in remapping file descriptors from the parent. The current environment for the parent is passed to the new process.

4. The **syscalls** function is called to enable signals. If this program were to be exec'd (run from the shell), this would not be necessary, and, similarly, the call later on to turn off signals would also be unnecessary.

   sigaction is used to set the action for **sigalrm** to be caught. **sigprocmask** is used to unblock **sigalrm**. This call also uses **sigaddset** and **sigsetempty** to create a signal mask with the **sigalrm** bit.

   The alarm is set by using the **alarm** service, and the process waits for completion of the child or the alarm. `"ALARM 0"` is used just to make sure the alarm is off.

5. A SELECT instruction is used on the status stem returned by the **waitpid** service. This determines if the process was terminated by a signal, if it exited, or if the process is stopped. If it exited, the exit status is available; otherwise, the signal number is available.

Chapter 5. OpenMVS Virtual File System (VFS) Server Syscall Commands

A number of syscall commands are intended for file system server applications, such as a Network File System server. Although it is unlikely that you would implement a Network File System server using REXX, you can access the server callable services using OS/390 OpenEdition REXX syscall commands. Since it is possible for a server to create a file in the file hierarchy with a name that cannot be accessed through conventional C functions—for example, a filename that has a slash (/) in it—these syscall commands may be useful if you need to obtain local access to such a file.

For detailed information about these services, see *OS/390 OpenEdition File System Interface Reference*.

Security

The file system server services are available only to a registered server. Only a superuser can use the callable service v_reg to register the process as a server.

The server services can bypass system security for file access.

For examples of REXX coding using these commands, see Chapter 6.

Tokens

Many tokens flow across the server interface. The types of tokens are:

VFS	Represents a mounted file system.
vnode	Represents a file or directory that is currently in use. This identifier is valid only until the token is released, using the v_rel callable service.
FID	Uniquely identifies a file or directory in a particular mounted file system; the file or directory may or may not be currently in use. This identifier is valid across mounting and unmounting of the file system, as well as OS/390 OpenEdition restarts.

You must specify tokens as variable names, not as strings. See “Specifying a Syscall Command” on page 17 for information on specifying variable names. When a token is returned to the exec, the value of the token is stored in the variable. When a token variable is used as a parameter on a syscall command, the token value is extracted from the variable. The format for a token is 8 bytes of binary data.

v_create

```
►►──v_create──vntoken──filename──type──mode──stem──vntoken2──────────►

►──┬─────────────┬───────────────────────────────────────────────────►◄
   └─major──minor─┘
```

Function

v_create invokes the v_create callable service to create a new file in the directory represented by *vntoken*. The file can be a regular, FIFO, or character special file.

Parameters

vntoken
A variable name that contains the vnode token for the directory in which the file *filename* is to be created.

filename
The name of the file. It must not contain null characters.

type
A number used to specify the type of file to be created: a regular, FIFO, or character special file. You can specify one of the predefined variables beginning with S_ to set the value—for example, S_ISREG. For a list of the variables, see “fstat” on page 67.

mode
A three-digit number, corresponding to the access permission bits. Each digit must be in the range `0`–`7`, and all three digits must be specified. For more information on permissions, see Appendix B.

stem
The name of a stem variable used to return the status information. Upon return, *stem.0* contains the number of variables that are returned. To obtain the desired status information, you can use a numeric value or the predefined variables beginning with ST_ used to derive the numeric value. See “stat” on page 155 for a list of the predefined variables or Appendix A for the numeric values.

vntoken2
A variable name that will contain the vntoken of the created file on return.

major
For a character special file (S_ISCHR), the device major number. For a complete description, see “mknod” on page 108.

minor
For a character special file (S_ISCHR), the device minor number. For a complete description, see “mknod” on page 108.

Usage Notes

1. If the file named in *filename* already exists, the v_create service returns a failing return code, and no vnode token is returned.
2. The caller is responsible for freeing vnode tokens returned by the service by calling to the v_rel service when they are no longer needed.

Example

In the following example, assume that *filenm* and *vnod* were assigned values earlier in the exec:

```
"v_create vnod (filenm)" s_isreg 777 "st. filetok"
```

v_fstatfs

```
►►──v_fstatfs──vntoken──stem───────────────────────────────────►◄
```

Function

v_fstatfs invokes the v_fstatfs callable service to return file system status for the file system containing the file or directory represented by the specified *vntoken*.

Parameters

vntoken
: A variable name that contains the vnode token for a file or directory in the file system whose status is to be checked.

stem
: The name of a stem variable used to return the status information. Upon return, *stem.0* contains the number of variables that are returned. To obtain the desired status information, you can use a numeric value or the predefined variables beginning with STFS_ used to derive the numeric value. For example, *stem.stfs_avail* accesses the number of blocks available in the file system. See “statfs” on page 157 for a list of the predefined variables, or Appendix A for the numeric values.

Example

In the following example, assume that *vnod* was assigned a value earlier in the exec:

```
" v_fstatfs vnod st."
```

v_get

```
►►──v_get──vfstoken──fid──vntoken─────────────────────────────►◄
```

Function

v_get invokes the v_get callable service to return a vnode token for the file or directory represented by the input FID within the mounted file system represented by the input VFS token.

Parameters

vfstoken
A variable name that contains the VFS token for the file system where the file identified by *fid* resides.

fid A variable that contains a file ID. File IDs are returned in the file attribute structure in the stem index ST_FID.

vntoken
A variable name that stores the vnode token for the requested file.

Usage Notes

1. The FID (file identifier) uniquely identifies a file in a particular mounted file system, and its validity persists across mounting and unmounting of the file system, as well as OS/390 OpenEdition restarts. This distinguishes the FID from the vnode token, which relates to a file in active use, and whose validity persists only until the token is released via the v_rel callable service.

   A server application uses **v_get** to convert an FID to a vnode token when it is preparing to use a file, since the vnode token identifies the file to the other services.

2. The FID for a file is returned in a stem variable by such services as v_rpn and v_lookup.
3. The caller is responsible for freeing vnode tokens returned by **v_get** by calling to the v_rel service when they are no longer needed.

Example

In the following example, assume that *vfs* and *st.st_fid* were assigned values earlier in the exec:

```
"v_get vfs st.st_fid vnod"
```

v_getattr

```
►►──v_getattr──vntoken──stem──────────────────────────────────►◄
```

Function

v_getattr invokes the v_getattr callable service to get the attributes of the file represented by *vntoken*.

Parameters

vntoken
: A variable name that contains the vnode token of the file for which the attributes are returned.

stem
: The name of a stem variable used to return the file attribute information. Upon return, *stem.0* contains the number of attribute variables returned. To obtain the desired information, you can use a numeric value or the predefined variables beginning with ST_ used to derive the numeric value. See "stat" on page 155 for a list of the variables, or Appendix A for the numeric values.

Example

In the following example, assume that *vnod* was assigned a value earlier in the exec:

```
"v_getattr vnod attr."
```

v_link

```
►►──v_link──vntoken──filename──vntoken2──────────────────────────►◄
```

Function

v_link invokes the v_link callable service to create a link to the file specified by *vntoken* in the directory specified by *vntoken2*. The link is a new name, *filename*, identifying an existing file.

Parameters

vntoken
A variable name that contains the vnode token for the file being linked to.

filename
The new name for the existing file.

vntoken2
A variable name that contains the vnode token for the directory to which *filename* is to be added.

Usage Notes

1. **v_link** creates a link named *filename* to an existing file specified by *vntoken*. This provides an alternative pathname for the existing file, so that you can access it by the old name or the new name. You can store the link under the same directory as the original file, or under a different directory on the same file system.
2. If the link is created successfully, the service routine increments the link count of the file. The link count shows how many links to a file exist. (If the link is not created successfully, the link count is not incremented.)
3. Links are not allowed to directories.
4. If the link is created successfully, the change time of the linked-to file is updated, as are the change and modification times of the directory that contains *filename*—that is, the directory that holds the link.

Example

In the following example, assume that *filetok*, *name*, and *dirtok* were assigned values earlier in the exec:

```
"v_link filetok (name) dirtok"
```

v_lockctl

```
►►──v_lockctl──v_command──stem──────────────────────────────►◄
```

Function

v_lockctl invokes the v_lockctl callable service to control advisory byte-range locks on a file.

Note: All locks are advisory only. Client and local processes can use locks to inform each other that they want to protect parts of a file, but locks do not prevent I/O on the locked parts. A process that has appropriate permissions on a file can perform whatever I/O it chooses, regardless of which locks are set. Therefore, file locking is only a convention, and it works only when all processes respect the convention.

Parameters

v_command

The name of a predefined command variable that is used to control the lock. You can specify a numeric value (see Appendix A) or a predefined VL_command variable that derives the appropriate numeric value. The command variables are

Variable	Description
VL_REGLOCKER	Register the lock server (locker).
VL_UNREGLOCKER	Unregister the locker.
VL_LOCK	Set a lock in a specified byte range.
VL_LOCKWAIT	Set a lock in a specified byte range or wait to set the lock until the byte range is free.
VL_UNLOCK	Unlock all locks in a specified byte range.
VL_QUERY	Query for lock information about a file.
VL_PURGE	Release all locks on all files, held by a locker or a group of lockers.

stem

The name of a stem variable that is the structure used to obtain information about the lock. To access the information, you can specify a numeric value (see Appendix A) or the predefined variables beginning with VL_ or L_ that derive the appropriate numeric value—for example, *stem.vl_serverpid*. The variables beginning with VL_ are:

Variable	Description
VL_SERVERPID	Server's PID.
VL_CLIENTPID	Server's client's process ID (PID).
VL_LOCKERTOK	Token for locker.
VL_CLIENTTID	Client's thread's TID. The TID is the individual lock owner within a locker.
VL_OBJCLASS	The class for an object (a single locked file)—for example, HFS for an HFS file, MVS for an MVS data set, LFSESA for a LAN file server.
VL_OBJID	The unique ID for an object (locked file) within its class. For an HFS file, the VL_OBJID contains the device number and FID of the file.
VL_OBJTOK	A token to identify the object (locked file) on a subsequent lock request.

Variable	Description
VL_DOSMODE	DOS file-sharing field.
VL_DOSACCESS	DOS file-sharing field.

For a description of the variables beginning with L_, see “f_getlk” on page 54.

Usage Notes

1. The v_lockctl service locks out other cooperating lockers from part of a file, so that the locker can read or write to that part of the file without interference from others.

2. Each locker must be registered before issuing any lock requests. On a REGLOCKER command, the caller must provide stem variables with these suffixes:

   VL_SERVERPID
   VL_CLIENTPID

   The VL_LOCKERTOK variable is returned to the caller; it is a token to identify the locker on subsequent lock requests.

3. On a QUERY, LOCK, LOCKWAIT, or UNLOCK command, the caller provides stem variables with these suffixes:

   VL_LOCKERTOK
   VL_CLIENTTID
   VL_OBJCLASS
   VL_OBJID
   VL_OBJTOK (This is optional, but it will improve performance for multiple lock requests)

   To describe the byte range for the command, the caller must also provide stem variables with the following L_ suffixes:

QUERY	L_TYPE, L_START, L_LEN, L_WHENCE
LOCK	L_TYPE, L_START, L_LEN, L_WHENCE
LOCKWAIT	L_TYPE, L_START, L_LEN, L_WHENCE
UNLOCK	L_START, L_LEN, L_WHENCE

   The L_ variables are described in “f_getlk” on page 54.

   VL_OBJTOK is returned to the caller; it is a token to identify the object on a subsequent lock request. On a QUERY, lock information describing a lock that would prevent the proposed lock from being set is returned to the caller.

4. Stem variables with the suffixes L_TYPE, L_START, and L_LEN are needed whether the request is for setting a lock, releasing a lock, or querying a particular byte range for a lock. L_WHENCE is always treated as SEEK_SET, the start of the file.

   The L_TYPE variable is used to specify the type of lock to be set or queried. (L_TYPE is not used to unlock.) You can use a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate value:

Type	**Description**
F_RDLCK	A *read lock*, also known as a *shared lock*. This type of lock specifies that the locker can read the locked part of the file, and other lockers cannot write on that part of the file in the meantime. A locker can change a held write lock, or any part of it, to a read lock, thereby making it available for other lockers to read. Multiple lockers can have read locks on the same part of a file simultaneously.
F_WRLCK	A *write lock*, also known as an *exclusive lock*. This type of lock indicates that the locker can write on the locked part of the file, without interference from other lockers. If one locker puts a write lock on part of a file, no other locker can establish a read lock or write lock on that same part of the file. A locker cannot put a write lock on part of a file if there is already a read lock on an overlapping part of the file, unless that locker is the only owner of that overlapping read lock. In such a case, the read lock on the overlapping section is replaced by the write lock being requested.
F_UNLCK	Returned on a query, when there are no locks that would prevent the proposed lock operation from completing successfully.

The L_WHENCE variable specifies how the byte range offset is to be found within the file; L_WHENCE is always treated as SEEK_SET, which stands for the start of the file.

The L_START variable is used to identify the part of the file that is to be locked, unlocked, or queried. The part of the file affected by the lock begins at this offset from the start of the file. For example, if L_START has the value `10`, a lock request attempts to set a lock beginning 10 bytes past the start of the file.

Note: Although you cannot request a byte range that begins or extends beyond the beginning of the file, you can request a byte range that begins or extends beyond the end of the file.

The L_LEN variable is used to give the size of the locked part of the file, in bytes. The value specified for L_LEN cannot be negative. If a negative value is specified for L_LEN, a RETVAL of −1 and an EINVAL ERRNO are returned. If L_LEN is zero, the locked part of the file begins at L_START and extends to the end of the file.

The L_PID variable identifies the VL_CLIENTPID of the locker that holds the lock found on a query request, if one was found.

5. You can set locks by specifying a VL_LOCK as the command parameter. If the lock cannot be obtained, a RETVAL of −1 is returned along with an appropriate ERRNO and ERRNOJR.

   You can also set locks by specifying a VL_LOCKWAIT as the command parameter. If the lock cannot be obtained because another process has a lock on all or part of the requested range, the LOCKWAIT request waits until the specified range becomes free and the request can be completed.

   If a signal interrupts a call to the v_lockctl service while it is waiting in a LockWait operation, the function returns with a RETVAL of −1, and the ERRNO EINTR.

LockWait operations have the potential for encountering deadlocks. This happens when locker A is waiting for locker B to unlock a byte range, and B is waiting for A to unlock a different byte range. If the system detects that a LockWait might cause a deadlock, the v_lockctl service returns with a RETVAL of −1 and the ERRNO EDEADLK.

6. A process can determine locking information about a file using VL_QUERY as the command parameter. The stem should describe a lock operation that the caller would like to perform. When the v_lockctl service returns, the structure is modified to describe the first lock found that would prevent the proposed lock operation from finishing successfully.

   If a lock is found that would prevent the proposed lock from being set, the query request returns a stem whose L_WHENCE value is always SEEK_SET, whose L_START value gives the offset of the locked portion from the beginning of the file, whose L_LEN value is set to the length of the locked portion of the file, and whose L_PID value is set to the ClientProcessID of the locker that is holding the lock. If there are no locks that would prevent the proposed lock operation from finishing successfully, the returned structure is modified to have an L_TYPE of F_UNLCK, but otherwise it remains unchanged.

7. A locker can have several locks on a file simultaneously but can have only one type of lock set on any given byte. Therefore, if a locker sets a new lock on part of a file that it had previously locked, the locker has only one lock on that part of the file, and the lock type is the one given by the most recent locking operation.

8. When a VL_UNLOCK command is issued to unlock a byte range of a file, all locks held by that locker within the specified byte range are released. In other words, each byte specified on an unlock request is freed from any lock that is held against it by the requesting locker.

9. Each locker should be unregistered when done issuing lock requests. On a VL_UNLOCKER command, the caller provides the stem variable VL_LOCKERTOK to identify the locker to unregister.

10. The VL_PURGE command releases all locks on all files, held by a locker or a group of lockers. The following stem variables are provided by the caller:

    VL_SERVERPID
    VL_CLIENTPID
    VL_CLIENTTID

Example

This example illustrates several calls to **v_lockctl** to register a locker, lock a range, unlock a range, and unregister a locker:

```
/* rexx */
address syscall
'v_reg 2 RxLocker'                /* register server as a lock server */
/*******************************************************************/
/* register locker                                                  */
/*******************************************************************/
lk.vl_serverpid=0                 /* use my pid as server pid        */
lk.vl_clientpid=1                 /* set client process id           */
'v_lockctl' vl_reglocker 'lk.'    /* register client as a locker     */
c1tok=lk.vl_lockertok             /* save client locker token        */
/*******************************************************************/
/* lock a range                                                     */
/*******************************************************************/
lk.vl_lockertok=c1tok             /* set client locker token         */
lk.vl_clienttid='thread1'         /* invent a thread id              */
lk.vl_objclass=1                  /* invent an object class          */
lk.vl_objid='objectname'          /* invent an object name           */
lk.vl_objtok=''                   /* no object token                 */
lk.l_len=40                       /* set length of range to lock     */
lk.l_start=80                     /* set start of range to lock      */
lk.l_whence=seek_set              /* start of range is absolute      */
lk.l_type=f_wrlck                 /* set write lock                  */
'v_lockctl' vl_lock 'lk.'         /* try to do the lock              */
obj1=lk.vl_objtok                 /* keep returned object token      */
/*******************************************************************/
/* unlock a range                                                   */
/*******************************************************************/
lk.vl_lockertok=c1tok             /* set client locker token         */
lk.vl_clienttid='thread1'         /* invent a thread id              */
lk.vl_objclass=1                  /* invent an object class          */
lk.vl_objid='objectname'          /* invent an object name           */
lk.vl_objtok=obj1                 /* set object token                */
lk.l_len=40                       /* set length of range to lock     */
lk.l_start=80                     /* set start of range to lock      */
lk.l_whence=seek_set              /* start of range is absolute      */
lk.l_type=f_unlck                 /* set unlock                      */
'v_lockctl' vl_unlock 'lk.'       /* unlock the range                */
/*******************************************************************/
/* unregister locker                                                */
/*******************************************************************/
lk.vl_lockertok=c1tok             /* set client locker token         */
'v_lockctl' vl_unreglocker 'lk.'  /* unregister client as a locker   */
return
```

v_lookup

```
►►──v_lookup──vntoken──filename──stem──vntoken2──────────────────►◄
```

Function

v_lookup invokes the v_lookup callable service to search a directory for a file. v_lookup accepts a vnode token representing a directory and a name identifying a file. If the file is found in the directory, a vnode token for the file and the attributes of the file are returned.

Parameters

vntoken
A variable name that contains the vnode token for the directory in which *filename* is looked up.

filename
The name of the file.

stem
The same file attribute information is returned in *stem* as if a v_getattr had been used on the file looked up. Upon return, *stem.0* contains the number of attribute variables returned. To access the attribute values, you can use a numeric value or the predefined variables beginning with ST_ used to derive the numeric value. See “stat” on page 155 for a list of the predefined variables, or Appendix A for the numeric values.

vntoken2
The variable name for the buffer that, when returned, will contain the vnode token for the looked-up file.

Usage Notes

The caller is responsible for freeing vnode tokens returned by the v_lookup service by calling to the v_rel service when they are no longer needed.

Example

In the following example, assume that *dirtok* and *file* were assigned values earlier in the exec:

```
"v_lookup dirtok (file) st. outtok"
```

v_mkdir

```
►►──v_mkdir──vntoken──directoryname──mode──stem──vntoken2──────────►◄
```

Function

v_mkdir invokes the v_mkdir callable service to create a new empty directory in the directory represented by *vntoken*, with the permission specified in *mode*.

Parameters

vntoken
A variable name that contains the vnode token for the directory in which *filename* is to be created.

directoryname
The name of the directory.

mode
A three-digit number, corresponding to the access permission bits for the directory. Each digit must be in the range `0–7`, and all three digits must be specified. For more information on permissions, see Appendix B.

stem
The same file attribute information is returned in *stem* as if a v_getattr had been used on the file specified. Upon return, *stem.0* contains the number of attribute variables returned. To access the attribute information, you can use a numeric value or the predefined variables beginning with ST_ used to derive the numeric value. See “stat” on page 155 for a list of the predefined variables, or Appendix A for the numeric values.

vntoken2
The variable name for the buffer that will contain the vnode token for the newly created directory.

Usage Notes

1. If the directory named in *directoryname* already exists, the v_mkdir service returns a failing return code, and no *vntoken2* is returned.
2. The caller is responsible for freeing vnode tokens returned by the v_mkdir service by calling to the v_rel service when they are no longer needed.

Example

In the following example, assume that *dirtok*, *file*, and *perm* were assigned values earlier in the exec:

```
"v_mkdir dirtok (file)" perm "st. newtok"
```

v_read

```
►►──v_read──vntoken──variable──length──position──stem──────────────►

►──┬──────────────┬────────────────────────────────────────────────►◄
   └─access_check─┘
```

Function

v_read invokes the v_rdwr callable service to accept a vnode token representing a file and to read data from the file. The file attributes are returned when the read completes. The number of bytes read is returned in RETVAL.

Parameters

vntoken
A variable name that contains the vnode token for the file to be read.

variable
The name of the buffer into which data will be read.

length
The maximum number of characters to read. After the read completes, the length of *variable* is the number of bytes read. This value is also returned in RETVAL.

position
The file offset where the read is to begin, specified in bytes.

stem
The name of a stem variable used to return the file attribute information. Upon return, *stem.0* contains the number of attribute variables returned. The same information is returned in *stem* as if a v_getattr had been used on the file. To obtain the attribute information, you can use a numeric value or the predefined variables beginning with ST_ used to derive the numeric value. See "stat" on page 155 for a list of the variables, or Appendix A for the numeric values.

access_check
Specify a 0 for no access check, or 1 to indicate the system is to check the user for read access to the file. The user is defined by the effective UID and GID. Setting the effective GID does not affect supplemental groups. Also, there is no support for altering the MVS user identity of the task using the ACEE (Accessor Environment Element).

Example

In the following example, assume that *filetok*, *bytes*, and *pos* were assigned values earlier in the exec:

```
"v_read filetok buffer" bytes pos stem. 0
```

v_readdir

```
►►──v_readdir──vntoken──stem──start──┬──────────────┬──────────────►◄
                                     └─access_check─┘
```

Function

v_readdir invokes the v_readdir callable service to accept a vnode token representing a directory and return directory entries from this directory.

Parameters

vntoken
A variable name that contains the vnode token for the directory to be read.

stem
The name of a stem variable used to return the directory entries. Upon return, *stem.0* contains the number of directory entries returned. *stem.1* through *stem.n* (where *n* is the number of entries returned) each contain a directory entry.

Note: Only small directories can be read in a single call. To ensure that you read to the end of the directory, make calls to v_readdir until no entries are returned.

start
The number of the first directory entry to be returned. The numbers `0` and 1 both indicate that the read should start at the beginning of the directory.

access_check
Specify a `0` for no access_check, or 1 to specify that the system is to check the user for read access to the file. The user is defined by the effective UID and GID. Setting the effective GID does not affect supplemental groups. Also, there is no support for altering the MVS user identity of the task using the ACEE.

Example

In the following example, assume that *dirtok* was assigned a value earlier in the exec:

```
"v_readdir dirtok dir. 0"
```

v_readlink

```
►►──v_readlink──vntoken──variable──────────────────────────────────►◄
```

Function

v_readlink invokes the v_readlink callable service to read the symbolic link file represented by the vnode token and return its contents in *variable*.

Parameters

vntoken

A variable name that contains the vnode token for the symbolic link to be read.

The attribute stem returned on call to another function (for example, v_getattr) identifies whether the symbolic link is a link to an external name in the stem index ST_EXTLINK. An external name is the name of an object outside the HFS.

variable

The name of the buffer that, on return, contains the contents of the symbolic link.

Example

In the following example, assume that *symtok* was assigned a value earlier in the exec:

```
"v_readlink symtok link"
```

v_reg

```
►►──v_reg──type──name──────────────────────────────────────────────►◄
```

Function

v_reg invokes the v_reg callable service to register a process as a server. A process must be registered using this service before it can use any other vnode interface services.

Parameters

type
A numeric value that defines the type of server. You can specify:

1 to indicate a file server
2 to indicate a lock server

name
The name of the server, a character string up to 32 bytes in length. There are no restrictions on the name; for example, it does not have to be unique in the system.

Usage Notes

1. Only a superuser can register a process as a server.
2. The D OMVS command uses the values supplied in *type* and *name* fields to display information about the currently active servers.

Example

To register a server:

```
'v_reg 1 "My REXX server"'
```

v_rel

```
►►──v_rel──vntoken──────────────────────────────────────────►◄
```

Function

v_rel invokes the v_rel callable service to accept a vnode token representing a file descriptor for a file or directory, and to release that token.

Parameters

vntoken
A variable name that contains the vnode token for the file descriptor to be released.

Usage Notes

1. The vnode token is no longer valid and cannot be used for subsequent requests after **v_rel** has successfully processed it.
2. This service must be used to release all vnode tokens obtained from other operations.

Example

In the following example, assume that *vntok* was assigned a value earlier in the exec:

```
"v_rel vntok"
```

v_remove

```
►►──v_remove──vntoken──filename──────────────────────────────────►◄
```

Function

v_remove invokes the v_remove callable service to remove a directory entry.

Parameters

vntoken
A variable name that contains the vnode token for the directory from which *filename* is to be removed.

filename
The name for the directory entry. The directory entry could be identified by a name for a file, the name of a hard link to a file, or the name of a symbolic link.

Usage Notes

1. If the name specified refers to a symbolic link, the symbolic link file named by *filename* is deleted.
2. If the v_remove service is successful and the link count becomes zero, the file is deleted. The contents of the file are discarded, and the space it occupied is freed for reuse. However, if another process (or more than one) has the file open or has a valid vnode token when the last link is removed, the file contents are not removed until the last process closes the file or releases the vnode token.
3. When the v_remove service is successful in removing a directory entry and decrementing the link count, whether or not the link count becomes zero, it returns control to the caller with RETVAL `0`. It updates the change and modification times for the parent directory, and the change time for the file itself (unless the file is deleted).
4. You cannot remove a directory using **v_remove**. To remove a directory, refer to “v_rmdir” on page 219.

Example

In the following example, assume that *dirtok* and *file* were assigned values earlier in the exec:

```
"v_remove dirtok (file)"
```

v_rename

```
►►──v_rename──vntoken──oldname──vntoken2──newname──────────────────►◄
```

Function

v_rename invokes the v_rename callable service to rename a file or directory to a new name.

Parameters

vntoken
A variable name that contains the vnode token for the directory that contains the filename *oldname*.

oldname
The existing name for the file or directory.

vntoken2
A variable name that contains the vnode token for the directory that is to contain the filename *newname*.

newname
The new name for the file or directory.

Usage Notes

1. The v_rename service changes the name of a file or directory from *oldname* to *newname*. When renaming completes successfully, the change and modification times for the parent directories of *oldname* and *newname* are updated.

2. The calling process needs write permission for the directory containing *oldname* and the directory containing *newname*. If *oldname* and *newname* are the names of directories, the caller does not need write permission for the directories themselves.

3. **Renaming files:** If *oldname* and *newname* are links referring to the same file, **v_rename** returns successfully and does not perform any other action.

   If *oldname* is the name of a file, *newname* must also name a file, not a directory. If *newname* is an existing file, it is unlinked. Then the file specified as *oldname* is given *newname*. The pathname *newname* always stays in existence; at the beginning of the operation, *newname* refers to its original file, and at the end, it refers to the file that used to be *oldname*.

4. **Renaming directories:** If *oldname* is the name of a directory, *newname* must also name a directory, not a file. If *newname* is an existing directory, it must be empty, containing no files or subdirectories. If it is empty, it is removed. *newname* cannot be a directory under *oldname*; that is, the old directory cannot be part of the pathname prefix of the new one.

Example

In the following example, assume that *olddir*, *oldfile*, *newdir*, and *newfile* were assigned values earlier in the exec:

```
"v_rename olddir (oldfile) newdir (newfile)"
```

v_rmdir

```
►►──v_rmdir──vntoken──dirname──────────────────────────────────────►◄
```

Function

v_rmdir invokes the v_rmdir callable service to remove an empty directory.

Parameters

vntoken
A variable name that contains the vnode token for the directory from which *dirname* is to be removed.

dirname
The name of the empty directory to be removed.

Usage Notes

1. The directory specified by *dirname* must be empty.
2. If the directory is successfully removed, the change and modification times for the parent directory are updated.
3. If any process has the directory open when it is removed, the directory itself is not removed until the last process closes the directory. New files cannot be created under a directory that is removed, even if the directory is still open.

Example

In the following example, assume that *dirtok* and *dirname* were assigned values earlier in the exec:

```
"v_rmdir dirtok (dirname)"
```

v_rpn

```
►►──v_rpn──pathname──vfstoken──vntoken──stem──stem2──────────────►◄
```

Function

v_rpn invokes the v_rpn callable service to accept a pathname of a file or directory and return a vnode token that represents this file or directory and the VFS token that represents the mounted file system that contains the file or directory.

Parameters

pathname
The absolute pathname to be resolved, specified as a string.

vfstoken
The name of a variable in which the VFS token for the resolved file is stored.

vntoken
The name of a variable in which the vnode token for the resolved file is stored.

stem
The name of a stem variable used to return the mount entry for the file system. To access mount table information, you can use a numeric value or the predefined variables beginning with MNTE_ used to derive the numeric value. See “getmntent” on page 83 for a description of the MNTE_ variables; see Appendix A for the numeric values.

stem2
Upon return, *stem2.0* contains the number of attribute variables returned. The same information is returned in *stem2* as if a v_getattr had been used on the file that was just resolved. You can use the predefined variables beginning with ST_ to access those respective values. For example, *stem2.st_size* accesses the file size. See “stat” on page 155 for a description of the ST_ variables.

Usage Notes

1. The mount point pathname is not available in the MNTE_ structure returned by the variable *stem2.mnte_path*.
2. The caller is responsible for freeing vnode tokens returned by the v_rpn service, by calling to the v_rel service when they are no longer needed.

Example

In the following example, assume that *path* was assigned a value earlier in the exec:

```
"v_rpn (path) vfstok filetok mnt. attr."
```

v_setattr

```
►►──v_setattr──vntoken──attribute_list──────────────────────────────►◄
```

Function

v_setattr invokes the v_setattr callable service to set the attributes associated with the file represented by the vnode token. You can change the mode, owner, access time, modification time, change time, reference time, audit flags, general attribute flags, and file size.

Parameters

vntoken
: A variable name that contains the vnode token for the file for which the attributes are to be set.

attribute_list
: A list of attributes to be set and their values. The attributes are expressed either as numeric values or as the predefined variables beginning with ST_, followed by arguments for that attribute. For the predefined variables beginning with ST_, see “chattr” on page 26; for the numeric values, see Appendix A.

Usage Notes

For usage notes, see “chattr” on page 26.

Example

In the following example, assume that *vntok* was assigned a value earlier in the exec. This example truncates a file to `0` length and sets the mode to 600:

```
"v_setattr vntok" st_size 0 st_mode 600
```

v_symlink

```
►►──v_symlink──vntoken──filename──pathname──┬─────────┬──────────────►◄
                                            └─extlink─┘
```

Function

v_symlink invokes the v_symlink callable service to create a symbolic link to a pathname or external name. The contents of the symbolic link file is *pathname*.

Parameters

vntoken
A variable name for the directory that contains the vnode token in which *filename* is being created.

filename
The name for the symbolic link.

pathname
The absolute or relative pathname of the file you are linking to (the contents of the symbolic link).

extlink
Specify 1 if this is a symbolic link to an external name rather than to a pathname in the file hierarchy. An external name is the name of an object outside of the file hierarchy.

Usage Notes

1. Like a hard link (described in "v_link" on page 203), a symbolic link allows a file to have more than one name. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link, however, provides no such assurance; in fact, the file identified by *pathname* need not exist when the symbolic link is created. In addition, a symbolic link can cross file system boundaries, and it can refer to objects outside of a hierarchical file system.

2. When a component of a pathname refers to a symbolic link (but not an external symbolic link) rather than to a directory, the pathname contained in the symbolic link is resolved. For v_rpn or other OS/390 OpenEdition callable services, a symbolic link in a pathname parameter is resolved as follows:

   - If the pathname in the symbolic link begins with / (slash), the symbolic link pathname is resolved relative to the process root directory.
   - If the pathname in the symbolic link does not begin with /, the symbolic link pathname is resolved relative to the directory that contains the symbolic link.
   - If the symbolic link is not the last component of the original pathname, remaining components of the original pathname are resolved from there.
   - When a symbolic link is the last component of a pathname, it may or may not be resolved. Resolution depends on the function using the pathname. For example, a rename request does not have a symbolic link resolved when it appears as the final component of either the new or old pathname. However, an open request does have a symbolic link resolved when it appears as the last component.

- When a slash is the last component of a pathname, and it is preceded by a symbolic link, the symbolic link is always resolved.
- Because it cannot be changed, the mode of a symbolic link is ignored during the lookup process. Any files and directories to which a symbolic link refers are checked for access permission.

3. The external name contained in an external symbolic link is not resolved. The *filename* cannot be used as a directory component of a pathname.

Example

In the following example, assume that *dirtok*, file, and linkname were assigned values earlier in the exec:

```
"v_symlink dirtok (file) (linkname)"
```

v_write

```
►►──v_write──vntoken──variable──length──position──stem──────────────►

►──┬──────────────┬─────────────────────────────────────────────────►◄
   └─access_check─┘
```

Function

v_rdwr invokes the v_rdwr callable service to accept a vnode token representing a file and to write data to the file. The number of bytes written and the file attributes are returned in RETVAL when the write completes.

Parameters

vntoken
A variable name that contains the vnode token for the file to be written.

variable
The name of the buffer from which data is to be written.

length
The number of characters to write.

position
The file offset where the write is to start from, specified in bytes.

stem
The name of a stem variable used to return the file attributes. Upon return, *stem.0* contains the number of attribute variables returned. The same information is returned in *stem* as if a v_getattr was used on the file. To access the file attributes, you can use a numeric value or the predefined variables beginning with ST_ used to derive the numeric value. See “stat” on page 155 for more information about the ST_ predefined variables; see Appendix A for the numeric values.

access_check
Specify `0` for no access check, or 1 for the system to check the user for read access to the file. The user is defined by the effective UID and GID. Setting the effective GID does not affect supplemental groups. Also, there is no support for altering the MVS user identity of the task using the ACEE.

Example

In the following example, assume that *filetok*, *buf*, and *pos* were assigned values earlier in the exec:

```
"v_write filetok buf" length(buf) pos
```

Chapter 6. Examples: REXX Coding with Virtual File System Syscall Commands

These are examples of REXX programs that use the virtual file system syscall commands.

List the Files in a Directory

Given a directory pathname, this example lists the files in the directory.

```
/* rexx */
parse arg dir                    /* take directory path as argument   */
if dir=' then
   do
   say 'directory argument required'
   return
   end
call syscalls 'ON'
address syscall
'v_reg 1 dirlist'                /* register as a file server         */
if retval=-1 then
   do
   say 'error registering as a server - error codes:' errno errnojr
   return
   end
'v_rpn (dir) vfs vn mnt. st.'    /* resolve the directory pathname    */
if retval=-1 then
   do
   say 'error resolving path' dir '- error codes:' errno errnojr
   return
   end
i=1                              /* next dir entry to read is 1       */
do forever                       /* loop reading directory            */
   'v_readdir vn d.' i           /* read starting at next entry       */
   if retval=-1 then
      do
      say 'error reading directory - error codes:' errno errnojr
      leave
      end
   if d.0=0 then leave           /* if nothing returned then done     */
   do j=1 to d.0                 /* process each entry returned       */
      say d.j
   end
   i=i+d.0                       /* set index to next entry           */
end
'v_rel vn'                       /* release the directory vnode       */
return
```

Remove a File or Empty Directory

Given a directory pathname and the filename to delete, this example removes the file or an empty directory.

```
/* rexx */
if __argv.0<>3 then              /* check for right number of args  */
   do
   say 'directory and filename required'
   return
   end
                                 /* _ _argv.1 is program name       */
dir=__argv.2                     /* 1st arg is directory pathname   */
file=__argv.3                    /* 2nd arg is file name            */
call syscalls 'ON'
address syscall
"v_reg 1 'remove file'"          /* register as a file server       */
if retval=-1 then
   do
   say 'error registering as a server - error codes:' errno errnojr
   return
   end
'v_rpn (dir) vfs vn mnt. st.'    /* resolve the directory pathname  */
if retval=-1 then
   do
   say 'error resolving path' dir '- error codes:' errno errnojr
   return
   end
'v_lookup vn (file) fst. fvn'    /* look up the file                */
if retval=-1 then
   do
   say "error locating file" file
   say "       in directory" dir
   say "       error codes:" errno errnojr
   end
 else
   do
   if fst.st_type=s_isdir then   /* if the file is a directory      */
      'v_rmdir vn (file)'        /* then delete it with v_rmdir     */
    else
      'v_remove vn (file)'       /* else delete it with v_remove    */
   if retval=-1 then
      say 'error deleting file - error codes:' errno errnojr
   'v_rel fvn'                   /* release the file vnode          */
   end
'v_rel vn'                       /* release the directory vnode     */
return
```

Appendix A. The OS/390 OpenEdition REXX Predefined Variables

Predefined variables make symbolic references easier and more consistent. Instead of using a numeric value, you can use the predefined variable that will derive that numeric value. The following list shows the data type and numeric value for each predefined variable. Each variable is discussed in the section about the syscall command with which it can be used. Most variable names correspond to the POSIX-defined names in the C runtime library include files (also known as header files).

Except for the error numbers and signal numbers, all variables contain an underscore. This is also true of most of the POSIX names in the C include files.

The data types are:

Bin	Binary: 2-byte hexadecimal
Char	Character
Dec	Decimal
Hex	Hexadecimal: 4-byte hexadecimal
Tok	Token
Oct	Octal

The predefined variables listed alphabetically are:

Variable	Data Type	Numeric Value
AUD_FEXEC	Dec	512
AUD_FREAD	Dec	33554432
AUD_FWRITE	Dec	131072
AUD_SEXEC	Dec	256
AUD_SREAD	Dec	16777216
AUD_SWRITE	Dec	65536
E2BIG	Hex	91
EACCES	Hex	6F
EAGAIN	Hex	70
EBADF	Hex	71
EBUSY	Hex	72
ECHILD	Hex	73
EDEADLK	Hex	74
EDOM	Hex	01
EEXIST	Hex	75
EFAULT	Hex	76
EFBIG	Hex	77
EILSEQ	Hex	93
EINTR	Hex	78
EINVAL	Hex	79
EIO	Hex	7A
EISDIR	Hex	7B
ELOOP	Hex	92
EMFILE	Hex	7C
EMLINK	Hex	7D
EMVSBADCHAR	Hex	A0
EMVSCATLG	Hex	99
EMVSCVAF	Hex	98

Variable	Data Type	Numeric Value
EMVSDYNALC	Hex	97
EMVSERR	Hex	9D
EMVSINITIAL	Hex	9C
EMVSNORTL	Hex	A7
EMVSNOTUP	Hex	96
EMVSPARM	Hex	9E
EMVSPATHOPTS	Hex	A6
EMVSPFSFILE	Hex	9F
EMVSPFSPERM	Hex	A2
EMVSSAF2ERR	Hex	A4
EMVSSAFEXTRERR	Hex	A3
EMVSTODNOTSET	Hex	A5
ENAMETOOLONG	Hex	7E
ENFILE	Hex	7F
ENODEV	Hex	80
ENOENT	Hex	81
ENOEXEC	Hex	82
ENOLCK	Hex	83
ENOMEM	Hex	84
ENOSPC	Hex	85
ENOSYS	Hex	86
ENOTDIR	Hex	87
ENOTEMPTY	Hex	88
ENOTTY	Hex	89
ENXIO	Hex	8A
EPERM	Hex	8B
EPIPE	Hex	8C
ERANGE	Hex	02
EROFS	Hex	8D
ESC_A	Hex	2F
ESC_B	Hex	16
ESC_F	Hex	0C
ESC_N	Hex	15
ESC_R	Hex	0D
ESC_T	Hex	05
ESC_V	Hex	0B
ESPIPE	Hex	8E
ESRCH	Hex	8F
EXDEV	Hex	90
F_OK	Dec	8
F_RDLCK	Dec	1
F_UNLCK	Dec	3
F_WRLCK	Dec	2
GR_GID	Dec	2
GR_MEM	Char	4
GR_MEMBERS	Dec	3
GR_NAME	Char	1
L_LEN	Dec	4
L_PID	Dec	5
L_START	Dec	3
L_TYPE	Dec	1
L_WHENCE	Dec	2
MNT_ASYNCHMOUNT	Dec	130

Variable	Data Type	Numeric Value
MNT_FILEACTIVE	Dec	0
MNT_FILEDEAD	Dec	1
MNT_FILEDRAIN	Dec	4
MNT_FILEFORCE	Dec	8
MNT_FILEIMMED	Dec	16
MNT_FILENORM	Dec	32
MNT_FILERESET	Dec	2
MNT_IMMEDTRIED	Dec	64
MNT_MODE_RDONLY	Dec	1
MNT_MODE_RDWR	Dec	0
MNT_MOUNTINPROGRESS	Dec	129
MNT_QUIESCED	Dec	128
MNTE_DD	Char	4
MNTE_DEV	Dec	3
MNTE_FSNAME	Char	6
MNTE_FSTYPE	Char	5
MNTE_MODE	Dec	2
MNTE_PARDEV	Dec	9
MNTE_PARM	Char	13
MNTE_PATH	Char	7
MNTE_QJOBNAME	Char	11
MNTE_QPID	Dec	12
MNTE_ROOTINO	Dec	10
MNTE_STATUS	Dec	8
MNTE_TYPE	Dec	1
MTM_DRAIN	Dec	2
MTM_FORCE	Dec	4
MTM_IMMED	Dec	8
MTM_NORMAL	Dec	16
MTM_NOSUID	Dec	262144
MTM_RDONLY	Dec	128
MTM_RDWR	Dec	64
MTM_REMOUNT	Dec	65536
MTM_RESET	Dec	1
MTM_SYNCHONLY	Dec	512
O_APPEND	Dec	8
O_CREAT	Dec	128
O_EXCL	Dec	64
O_NOCTTY	Dec	32
O_NONBLOCK	Dec	4
O_RDONLY	Dec	2
O_RDWR	Dec	3
O_SYNC	Dec	256
O_TRUNC	Dec	16
O_WRONLY	Dec	1
PC_LINK_MAX	Dec	2
PC_MAX_CANON	Dec	3
PC_MAX_INPUT	Dec	4
PC_NAME_MAX	Dec	5
PC_PATH_MAX	Dec	7
PC_PIPE_BUF	Dec	8
PC_POSIX_CHOWN_RESTRICTED	Dec	1
PC_POSIX_NO_TRUNC	Dec	6

Variable	Data Type	Numeric Value
PC_POSIX_VDISABLE	Dec	9
PS_CHILD	Char	W
PS_CMD	Dec	19
PS_CONTTY	Dec	17
PS_EGID	Dec	10
PS_EUID	Dec	7
PS_FGPID	Dec	6
PS_FORK	Char	X
PS_FREEZE	Char	E
PS_MAXVNODES	Dec	25
PS_MSGRCV	Char	A
PS_MSGSND	Char	B
PS_PATH	Char	18
PS_PAUSE	Char	G
PS_PGPID	Dec	5
PS_PID	Dec	2
PS_PPID	Dec	3
PS_QUIESCE	Char	Q
PS_RGID	Dec	11
PS_RUID	Dec	8
PS_RUN	Char	R
PS_SEMWT	Char	D
PS_SERVERFLAGS	Dec	27
PS_SERVERNAME	Dec	22
PS_SERVERTYPE	Dec	21
PS_SGID	Dec	12
PS_SID	Dec	4
PS_SIZE	Dec	13
PS_SLEEP	Char	S
PS_STARTTIME	Dec	14
PS_STAT	Dec	1
PS_STATE	Char	20
PS_SUID	Dec	9
PS_SYSTIME	Dec	16
PS_USERTIME	Dec	15
PS_VNODECOUNT	Dec	26
PS_WAITC	Char	C
PS_WAITF	Char	F
PS_WAITO	Char	K
PS_ZOMBIE	Char	Z
PS_ZOMBIE2	Char	L
PW_DIR	Char	4
PW_GID	Dec	3
PW_NAME	Char	1
PW_SHELL	Char	5
PW_UID	Dec	2
R_OK	Dec	4
RLIMIT_AS	Dec	5
RLIMIT_CORE	Dec	4
RLIMIT_CPU	Dec	0
RLIMIT_FSIZE	Dec	1
RLIM_INFINITY	Dec	2147483647
RLIMIT_NOFILE	Dec	6
S_FFBINARY	Dec	1
S_FFCR	Dec	3

Variable	Data Type	Numeric Value
S_FFCRLF	Dec	5
S_FFLF	Dec	4
S_FFLFCR	Dec	6
S_FFNA	Dec	0
S_FFNL	Dec	2
S_ISCHR	Dec	2
S_ISDIR	Dec	1
S_ISFIFO	Dec	4
S_ISREG	Dec	3
S_ISSYM	Dec	5
SA_NOCLDSTOP	Dec	32768
SA_NOCLDWAIT	Dec	512
SA_NORESETHAND	Dec	4096
SC_2_CHAR_TERM	Dec	12
SC_ARG_MAX	Dec	1
SC_CHILD_MAX	Dec	2
SC_CLK_TCK	Dec	3
SC_JOB_CONTROL	Dec	4
SC_NGROUPS_MAX	Dec	5
SC_OPEN_MAX	Dec	6
SC_SAVED_IDS	Dec	7
SC_THREAD_TASKS_MAX_NP	Dec	11
SC_THREADS_MAX_NP	Dec	13
SC_TZNAME_MAX	Dec	9
SC_VERSION	Dec	10
SEEK_CUR	Dec	1
SEEK_END	Dec	2
SEEK_SET	Dec	0
SIG_BLOCK	Dec	0
SIG_CAT	Dec	10
SIG_DFL	Dec	0
SIG_IGN	Dec	1
SIG_QRY	Dec	11
SIG_SETMASK	Dec	2
SIG_UNBLOCK	Dec	1
SIGABND	Dec	18
SIGABRT	Dec	3
SIGALRM	Dec	14
SIGBUS	Dec	10
SIGCHLD	Dec	20
SIGCONT	Dec	19
SIGDCE	Dec	38
SIGFPE	Dec	8
SIGHUP	Dec	1
SIGILL	Dec	4
SIGINT	Dec	2
SIGIO	Dec	23
SIGIOERR	Dec	27
SIGKILL	Dec	9
SIGPIPE	Dec	13
SIGPOLL	Dec	5
SIGPROF	Dec	32
SIGQUIT	Dec	24
SIGSEGV	Dec	11
SIGSTOP	Dec	7
SIGSYS	Dec	12
SIGTERM	Dec	15

Variable	Data Type	Numeric Value
SIGTRAP	Dec	26
SIGTSTP	Dec	25
SIGTTIN	Dec	21
SIGTTOU	Dec	22
SIGUSR1	Dec	16
SIGUSR2	Dec	17
SIGURG	Dec	6
SIGVTALRM	Dec	31
SIGXCPU	Dec	29
SIGXFSZ	Dec	30
ST_AAUDIT	Dec	16
ST_ATIME	Dec	9
ST_AUDITID	Char	20
ST_BLKSIZE	Dec	18
ST_BLOCKS	Dec	22
ST_CCSID	Char	21
ST_CRTIME	Dec	19
ST_CTIME	Dec	11
ST_DEV	Hex	4
ST_EXTLINK	Dec	24
ST_FID	Bin	27
ST_FILEFMT	Dec	28
ST_GENVALUE	Bin	25
ST_GID	Dec	7
ST_INO	Hex	3
ST_MAJOR	Dec	14
ST_MINOR	Dec	15
ST_MODE	Oct	2
ST_MTIME	Dec	10
ST_NLINK	Dec	5
ST_RTIME	Dec	26
ST_SETGID	Dec	13
ST_SETUID	Dec	12
ST_SIZE	Dec	8
ST_STICKY	Dec	23
ST_TYPE	Dec	1
ST_UAUDIT	Dec	17
ST_UID	Dec	6
STFS_AVAIL	Dec	4
STFS_BFREE	Dec	6
STFS_BLOCKSIZE	Dec	1
STFS_FAVAIL	Dec	9
STFS_FILES	Dec	7
STFS_FFREE	Dec	8
STFS_FRSIZE	Dec	5
STFS_FSID	Dec	10
STFS_INUSE	Dec	3
STFS_INVARSEC	Dec	15
STFS_NAMEMAX	Dec	13
STFS_NOSUID	Dec	12
STFS_RDONLY	Dec	11
STFS_TOTAL	Dec	2
TM_HOUR	Dec	1
TM_ISDST	Dec	9
TM_MDAY	Dec	5
TM_MIN	Dec	2

Variable	Data Type	Numeric Value
TM_MON	Dec	4
TM_SEC	Dec	3
TM_WDAY	Dec	8
TM_YDAY	Dec	7
TM_YEAR	Dec	6
TMS_CSTIME	Dec	4
TMS_CUTIME	Dec	3
TMS_STIME	Dec	2
TMS_UTIME	Dec	1
U_MACHINE	Char	5
U_NODENAME	Char	2
U_RELEASE	Char	3
U_SYSNAME	Char	1
U_VERSION	Char	4
VL_CLIENTPID	Dec	7
VL_CLIENTTID	Char	9
VL_DOSMODE	Char	13
VL_DOSACCESS	Char	14
VL_LOCK	Dec	3
VL_LOCKERTOK	Tok	8
VL_LOCKWAIT	Dec	4
VL_OBJCLASS	Char	10
VL_OBJID	Char	11
VL_OBJTOK	Tok	12
VL_PURGE	Dec	7
VL_QUERY	Dec	6
VL_REGLOCKER	Dec	1
VL_SERVERPID	Dec	6
VL_UNREGLOCKER	Dec	2
VL_UNLOCK	Dec	5
W_CONTINUED	Dec	3
W_EXITSTATUS	Dec	4
W_IFEXITED	Dec	3
W_IFSIGNALED	Dec	5
W_IFSTOPPED	Dec	7
W_NOHANG	Dec	1
W_OK	Dec	2
W_STAT3	Dec	1
W_STAT4	Dec	2
W_STOPSIG	Dec	8
W_TERMSIG	Dec	6
W_UNTRACED	Dec	2
X_OK	Dec	1

Appendix B. Setting Permissions for Files and Directories

Typically, octal permissions are specified with three or four numbers, in these positions:

```
1234
```

Each position indicates a different type of access:

- Position `1` is the bit that sets permission for set-user-ID on access, set-group-ID on access, or the *sticky bit*. Specifying this bit is optional.
- Position `2` is the bit that sets permissions for the owner of the file. Specifying this bit is required.
- Position `3` is the bit that sets permissions for the group that the owner belongs to. Specifying this bit is required.
- Position `4` is the bit that sets permissions for others. Specifying this bit is required.

Position 1

Specifying the bit in position 1 is optional. You can specify one of these values:

0	Off
1	Sticky bit on
2	Set-group-ID-on execution
3	Set-group-ID-on execution and set the sticky bit on
4	Set-user-ID on execution
5	Set-user-ID on execution and set the sticky bit on.
6	Set-user-ID and set-group-ID on execution
7	Set-user-ID and set-group-ID on execution and set the sticky bit on

Positions 2, 3, and 4

Specifying these bits is required. For each type of access—owner, group, and other—there is a corresponding octal number:

0	No access (`---`)
1	Execute-only access (`--x`)
2	Write-only access (`-w-`)
3	Write and execute access (`-wx`)
4	Read-only access (`r--`)
5	Read and execute access (`r-x`)
6	Read and write access (`rw-`)
7	Read, write, and execute access (`rwx`)

To specify permissions for a file or directory, you use at least a *3-digit* octal number, omitting the digit in the first position. When you specify just three digits, the first digit describes owner permissions, the second digit describes group permissions, and the third digit describes permissions for all others. When the first digit is not set, some typical 3-digit permissions are specified in octal as shown in Table 2 on page 236.

Table 2. Three-Digit Permissions Specified in Octal

Octal Number		Meaning
666	6 6 6 / | \\ rw- rw- rw-	owner (rw-) group (rw-) other (rw-)
700	7 0 0 / | \\ rwx --- ---	owner (rwx) group (---) other (---)
755	7 5 5 / | \\ rwx r-x r-x	owner (rwx) group (r-x) other (r-x)
777	7 7 7 / | \\ rwx rwx rwx	owner (rwx) group (rwx) other (rwx)

Example: Using BITOR and BITAND to Set Mode Bits

To set a file's mode bits, use the REXX functions BITOR() and BITAND() with the octal numbers.

For example, if you have obtained a file's permission bits and want to use **chmod** to turn on the write bits, you could code:

```
'chmod (file)' BITOR(st.st_mode, 222)
```

To turn the same bits off, you could code:

```
'chmod (file)' BITAND(st.st_mode, 555)
```

Appendix C. Return Codes

This appendix describes and lists return codes.

Hex Value	Return Code	Description
0001	EDOM	There is an error in the domain.
0002	ERANGE	The result is too large.
006F	EACCES	Permission is denied.
0070	EAGAIN	The resource is temporarily unavailable.
0071	EBADF	The file descriptor is incorrect.
0072	EBUSY	The resource is busy.
0073	ECHILD	No child process exists.
0074	EDEADLK	A resource deadlock is avoided.
0075	EEXIST	The file exists.
0076	EFAULT	The address is incorrect.
0077	EFBIG	The file is too large.
0078	EINTR	A function call is interrupted.
0079	EINVAL	The parameter is incorrect.
007A	EIO	An I/O error occurred.
007B	EISDIR	The file specified is a directory.
007C	EMFILE	Too many files are open for this process.
007D	EMLINK	Too many links occurred.
007E	ENAMETOOLONG	The filename is too long.
007F	ENFILE	Too many files are open in the system.
0080	ENODEV	No such device exists.
0081	ENOENT	No such file or directory exists.
0082	ENOEXEC	The exec call contained a format error.
0083	ENOLCK	No locks are available.
0084	ENOMEM	Not enough space is available.
0085	ENOSPC	No space is left on the device.
0086	ENOSYS	The function is not implemented.
0087	ENOTDIR	This is not a directory.
0088	ENOTEMPTY	The directory is not empty.
0089	ENOTTY	The I/O control operator is inappropriate.
008A	ENXIO	No such device or address exists.
008B	EPERM	The operation is not permitted.
008C	EPIPE	The pipe is broken.
008D	EROFS	The specified file system is read-only.
008E	ESPIPE	The seek is incorrect.
008F	ESRCH	No such process or thread exists.
0090	EXDEV	A link to a file on another file system was attempted.

Hex Value	Return Code	Description
0091	E2BIG	The parameter list is too long.
0092	ELOOP	A loop is encountered in symbolic links.
0093	EILSEQ	The byte sequence is not allowed.
0096	EMVSNOTUP	The OpenEdition kernel is not active.
0097	EMVSDYNALC	There was a dynamic allocation error.
0098	EMVSCVAF	There was a catalog volume access facility error.
0099	EMVSCATLG	There was a catalog obtain error.
009C	EMVSINITIAL	There was a process initialization error.
009D	EMVSERR	An MVS environmental or internal error has occurred.
009E	EMVSPARM	Bad parameters were passed to the service.
009F	EMVSPFSFILE	The HFS encountered a permanent file error.
00A0	EMVSBADCHAR	There was a bad character in an environment variable name.
00A2	EMVSPFSPERM	The HFS encountered a system error.
00A3	EMVSSAFEXTRERR	There was a SAF/RACF extract error.
00A4	EMVSSAF2ERR	There was a SAF/RACF error.
00A5	EMVSTODNOTSET	The system TOD clock is not set.
00A6	EMVSPATHOPTS	The access mode argument conflicts with the PATHOPTS parameter.
00A7	EMVSNORTL	Access to the OpenEdition version of the C RTL is denied.

Appendix D. Error Messages Issued from the OS/390 OpenEdition REXX Processor

All error messages are directed to standard output.

BPXW0000I Exec not found

Explanation: The REXX program could not be found.

System Action: The REXX program is not run.

User Response: Check the format of the REXX program and make sure you have permission to execute the program. Make sure you specified the name with letters in the correct case (upper or lower). If you specified a relative name, check that the program can be found with the **PATH** environment variable used to exec the REXX program.

When an external subroutine or function is called, you may see the IRX0043I (routine not found) message. Make sure the subroutine name is quoted if it contains lowercase or special characters.

BPXW0001I Storage allocation error

Explanation: The OS/390 OpenEdition REXX preprocessor was unable to allocate sufficient storage to process the REXX program.

System Action: The REXX program is not run.

User Response: Check whether the program is looping on a call to an external function or subroutine. Contact your system programmer to ensure that your region size is sufficient for your application.

BPXW0002I Unable to read exec

Explanation: The REXX program could not be read. The usual cause for this is that an I/O error occurred on the read operation.

System Action: The REXX program is not run.

User Response: Ensure that the entire file can be read.

BPXW0003I Improper text file

Explanation: The REXX program is not a compiled exec and contains a line that is not terminated by a <newline> character.

System Action: The REXX program is not run.

User Response: Check the format of the REXX program. Make sure each line is terminated by a <newline> character.

BPXW0004I Parameter string too long

Explanation: The parameter passed to a REXX program exceeds 4096 characters. This is most likely to occur when you run a REXX program under the shell, using shell wildcards to pass a long file list or passing the output of another command as the parameter.

System Action: The REXX program is not run.

User Response: Run the REXX program with fewer parameters.

Glossary

This glossary defines terms and abbreviations used in this publication. If you do not find the term you are looking for, refer to the index portion of this book or to *IBM Dictionary of Computing* (New York: McGraw-Hill, 1994), ISBN 0-07-031488-8 (North American hardcover, $39.50), ISBN 0-07-031489-6 (North American paperback, $24.95), ISBN 0-07-113383-6 (international).

This glossary includes terms and definitions from:

- *IBM Dictionary of Computing* (New York: McGraw-Hill, 1994).
- *Information Technology—Portable Operating System Interface (POSIX),* from the POSIX** series of standards for applications and user interfaces to open systems, copyrighted by the Institute of Electrical and Electronics Engineers (IEEE**). Copies of all POSIX drafts and standards may be purchased from IEEE at 1-800-678-IEEE.
  - Definitions identified by *[POSIX.0]* are from *Part 0: Standards Project, Draft Guide to the POSIX Open System Environment,* P1003.0 Draft 15 (June 1992), an unapproved draft subject to change.
  - Definitions identified by *[POSIX.1]* are from *Part 1: System Application Program Interface (API) [C Language],* approved September 28, 1990, as IEEE Std 1003.1-1990 by the IEEE Standards Board, and adopted in 1990 as an International Standard (ISO/IEC 9945-1: 1990) by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).
  - Definitions identified by *[POSIX.2]* are from *Part 2: Shell and Utilities,* P1003.2.
- *American National Standard Dictionary for Information Systems,* ANSI** X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol *[A]* after the definition.
- *Information Technology Vocabulary,* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions from published sections of these vocabularies are identified by the symbol *[I]* after the definition. Definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol *[T]* after the definition, indicating that final agreement has not yet been reached among the participating national bodies of SC1.
- *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the International Telecommunication Union, Geneva, 1978. These are identified by the symbol *[CCITT/ITU]* after the definition.
- Open Software Foundation (OSF**). These are identified by the symbol *[OSF]* after the definition. Copies of OSF documents may be obtained from Open Software Foundation, Inc., 11 Cambridge Center, Cambridge, MA 02142.

Sequence of Entries: For clarity and consistency of style, this glossary arranges the entries alphabetically on a letter-by-letter basis. In other words, only the letters of the alphabet are used to determine sequence; special characters and spaces between words are ignored.

Organization of Entries: Each entry consists of a single-word or multiple-word term or the abbreviation or acronym for a term, followed by a commentary. A commentary includes one or more items (definitions or references) and is organized as follows:

1. An item number, if the commentary contains two or more items.
2. A usage label, indicating the area of application of the term, for example, “In programming,” or “In TCP/IP.” Absence of a usage label implies that the term is generally applicable to the OpenEdition MVS interface, to IBM, or to data processing.
3. A descriptive phrase, stating the basic meaning of the term. The descriptive phrase is assumed to be preceded by “the term is defined as.” The part of speech being defined is indicated by the opening words of the descriptive phrase: “To...” indicates a verb, and “Pertaining to ...” indicates a modifier. Any other wording indicates a noun or noun phrase.
4. Annotative sentences, providing additional or explanatory information.
5. References, directing the reader to other entries or items in the dictionary.
6. A source label—for example, *[A], [I], [T], [CCITT/ITU], [OSF], [POSIX.0], [POSIX.1],* or *[POSIX.2]*—that follows the definition and identifies the originator of the definition. Definitions without source labels are IBM definitions.

References: The following cross-references are used in this glossary:

Contrast with. This refers to a term that has an opposed or substantively different meaning.

Synonym for. This indicates that the term has the same meaning as a preferred term, which is defined in its proper place in the glossary.
Synonymous with. This is a backward reference from a defined term to all other terms that have the same meaning.
See. This refers you to multiple-word terms that have the same last word.
See also. This refers the reader to related terms that have a related, but not synonymous, meaning.
Deprecated term for or **Deprecated abbreviation for.** This indicates that the term or abbreviation should not be used. It refers to a preferred term, which is defined in its proper place in the glossary.

Selection of Terms: A term is a word or group of words to be defined. In this glossary, the singular form of the noun and the infinitive form of the verb are the terms most often selected to be defined. If the term may be abbreviated, the abbreviation is given in parentheses immediately following the term. The abbreviation is also defined in its proper place in the glossary.

A

absolute value. The numeric value of a real number regardless of its algebraic sign (positive or negative). [OSF]

access control. In computer security, ensuring that the resources of a computer system can be accessed only by authorized users in authorized ways.

access mode. A form of access permitted to a file. [POSIX.1]

access permission. A group of designations that determine who can access a particular file and how the user can access the file.

address space. (1) The memory locations that can be referred to by a process. [POSIX.1] (2) The code, stack, and data that are accessible by a process. (3) The complete range of addresses that is available to a programmer. (4) The area of virtual storage available for a program. (5) See *forked address space*, *kernel address space*, *user address space*.

appropriate privileges. An implementation-defined means of associating privileges with a process with regard to the function calls and function call options defined in POSIX.1 that need special privileges. [POSIX.1] In the OpenEdition implementation, appropriate privilege is defined as superuser authority. A trusted or privileged attribute is an attribute associated with a started procedure address space and to any process associated with the address space.

argument. (1) An independent variable. [I][A] (2) Any value of an independent variable, for example, a search key; a number identifying the location of an item in a table. [I][A] (3) A parameter passed between a calling and a called program. (4) A parameter passed to a utility as the equivalent of a single string in the *argv* array created by one of the POSIX.1 **exec** functions. An argument is one of the options, option-arguments, or operands following the command name. [POSIX.2]

B

background. (1) In multiprogramming, the conditions under which low-priority programs are executed. (2) Contrast with *foreground.*

background process. (1) A process that is a member of a background process group. [POSIX.1] (2) An OpenEdition process that an interactive user starts running and that cannot interact with the user. An interactive user can move a background process to the foreground.

C

character special file. (1) A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. (2) A file that refers to a device. One specific type of character special file is a terminal device file. Other character special files have no structure defined by POSIX.1, and their use is unspecified by POSIX.1. [POSIX.1] The only character special file supported by the OpenEdition implementation is the pseudo-TTY.

character type. A data type that consists of alphanumeric characters. See also *data type*.

child process. A process created as a result of a fork. The child process receives a copy of the parent's storage and inherits open files. Execution in the child continues at the instruction following the fork. Contrast with *parent process*. See also *fork, process.*

controlling process. The session leader that established the connection to the controlling terminal. If the terminal subsequently ceases to be a controlling terminal for this session, the session leader ceases to be the controlling process. [POSIX.1]

controlling terminal. (1) An active terminal at which a user is authorized to enter commands that affect system operation. The controlling terminal for any process normally is the active terminal from which the process group for that process was started. A terminal can have no more than one controlling process group and a process group can have no more than one controlling terminal. The controlling process group receives certain interrupt signals from the controlling terminal. [OSF] (2) A terminal that is associated with a session. Each session may have at most one controlling terminal associated with it, and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal cause signals to be sent to all processes in the process group associated with the controlling terminal. [POSIX.1]

current directory. The directory that is active and can be displayed with the **pwd** command. Synonymous with current working directory. [OSF]

current working directory. (1) The directory a user is working with. (2) Synonym for *current directory.* (3) Synonym for *working directory.*

D

descriptor. An unsigned integer that a UNIX system uses to identify an object supported by the kernel. Descriptors can represent files, pipes, and other I/O streams. They are created, acted on, and deallocated by system calls specific to the object. [OSF]

detach. To stop a thread.

directory. (1) A type of file containing the names and controlling information for other files or other directories. (2) A construct for organizing computer files. As files are analogous to folders that hold information, a directory is analogous to a drawer that can hold a number of folders. Directories can also contain subdirectories, which can contain subdirectories of their own. (3) A file that contains directory entries. No two directory entries in the same directory can have the same name. [POSIX.1] (4) A file that points to files and to other directories. (5) An index used by a control program to locate blocks of data that are stored in separate areas of a data set in direct access storage.

directory entry. An object that associates a filename with a file. Several directory entries can associate names with the same file. Synonymous with link. [POSIX.1]

dub. To make an MVS address space known to OS/390 OpenEdition. Once dubbed, an address space is considered to be an OpenEdition “process.” Address spaces created by **fork()** are automatically dubbed when they are created; other address spaces become dubbed if they invoke an OpenEdition service. Dubbing also applies to MVS tasks. A dubbed task is considered an OpenEdition “thread.” Tasks created by **pthread_create()** are automatically dubbed threads; other tasks are dubbed if they invoke an OpenEdition service. Contrast with *undub.*

E

effective group ID. (1) The current group ID, but not necessarily the user's own ID. For example, a user logged in under a particular group ID may be able to change to another group ID. The ID to which the user changes becomes the effective group ID. [OSF] (2) An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject to change during the process lifetime. See also *group ID (GID).* [POSIX.1]

effective user ID. (1) The current user ID, but not necessarily the user's login ID. For example, a user logged in under a login ID may change to another user's ID. The ID to which the user changes becomes the effective user ID until the user switches back to the original login ID. [OSF] (2) An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject to change during the process lifetime. See also *user ID.* [POSIX.1]

end of file (EOF). A code that signals that the last record of a file has been read.

environment. (1) The settings for shell variables and paths set when the user logs in. These variables can be modified later by the user. [OSF] (2) A block of information passed (“exported”) to a command when the command is invoked. This block contains a number of environment variables. The environment provides information that the program may use in its operation, in a form that relieves you of the need to specify it with every command. (3) The set of all factors that may affect how a program behaves. (4) A named collection of logical and physical resources used to support the operation of a function. (5) See also *environment variable (ENV).*

environment variable (ENV). (1) A name associated with a string of characters, made available to the programs that you run. Some environment variables used by the OpenEdition shell are **PATH**, **TMPDIR**, **COLUMNS**, and **LINES**. For example, the **TMPDIR** environment variable holds the name of a directory where shell commands are free to create temporary working files. (2) A variable that describes the operating environment of the process and typically includes information about the home directory, command search path, the terminal in use, and the current time zone (the **HOME**, **PATH**, **TERM**, and **TZ** variables, respectively). (3) An environment variable the value of which is the name of a file that contains shell commands to customize a shell environment. When a user first runs the shell, the shell executes his or her profile. Then the shell runs the commands in the **ENV** file. (4) A variable included in the current software environment that is available to any called program that requests it.

Epoch. The time 0 hours, 0 minutes, 0 seconds, January 1, 1970, Coordinated Universal Time. See also *seconds since the Epoch.* [POSIX.1]

exec. To overlay the current process with another executable program. See also *fork*.

executable. See *executable file, executable module, executable program, executable statement*.

executable file. (1) A file suitable for execution. An executable file may be a program that has been compiled and link-edited, or it may be a shell script. (2) A file that contains programs or commands that perform operations on actions to be taken. (3) A regular file acceptable as a new process image file by the equivalent of the POSIX.1 **exec** family of functions, and thus usable as one form of a utility. The standard utilities described in POSIX.1 as compilers can produce executable files, but other unspecified methods of producing executable files may also be provided. The internal format of an executable file is unspecified, but a conforming application shall not assume an executable file is a text file. [POSIX.2] (4) See also *executable program*.

executable module. A module in a partitioned data set (PDS) that can be executed. An executable module that is copied into an HFS file and then given read and execute permissions becomes an executable file. On the other hand, an executable file is not necessarily an executable module.

executable program. (1) A program in a form suitable for execution by a computer. The program can be an application or a shell script. An executable program is equivalent to an MVS load module. (2) A program that has been link-edited and can therefore be run in a processor. (3) A program that can be executed as a self-contained procedure. It consists of a main program and, optionally, one or more subprograms. (4) See also *executable file, load module*.

executable statement. A statement that causes an action to be taken by the program; for example, to calculate, to test conditions, or to alter normal sequential execution. [OSF]

F

FIFO (first-in-first-out). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. [A] Contrast with *LIFO (last-in-first-out)*.

FIFO special file. (1) A type of file with the property that data written to such a file is read on a first-in-first-out basis. [POSIX.1] (2) A named permanent pipe that allows two unrelated processes to exchange information through a pipe connection. Synonymous with named pipe.

file. (1) A set of related records treated as a unit. (2) A sequence of records. If the file is located in internal storage, it is an internal file; if it is on an input/output device, it is an external file. [OSF] (3) A collection of related data that is stored and retrieved by an assigned name. [OSF] (4) Linear data that can be opened, written, read, and closed. A file can also contain information about the file, such as authorization information. The name used to obtain a file includes the directories in the path to the file. (5) Strings of characters with no additional structure. Structure is assumed only by the processing programs. Files can be located relative to the current directory or by an absolute pathname. (6) An object that can be written to, or read from, or both. A file has certain attributes, including access permissions and type. File types include regular file, character special file, block special file, FIFO special file, and directory. Other types of files may be defined by the implementation. [POSIX.1] In the OpenEdition implementation, the file system does not support block special files, but it does support symbolic link files. (7) A collection of information or data that is organized by some method (relative, indexed, or serial, for example) and stored on a device such as a disk. (8) Synonym for *data set.* (9) In word processing, synonym for *document.*

file creation mask. See *mask*.

file descriptor. (1) A small unsigned integer that a UNIX system uses to identify a file. A file descriptor is created by a process through issuing an **open** system call for the filename. A file descriptor ceases to exist when it is no longer held by any process. [OSF] (2) A per-process unique, nonnegative integer used to identify an open file for the purpose of file access. [POSIX.1] (3) An small positive number used to identify an open file in I/O operations. By convention, certain file descriptors are used for the same purpose by all programs. (4) See also *standard error (stderr), standard input (stdin), standard output (stdout).*

file lock. A means to limit or deny access to a file by other users. A file lock can be a read lock or a write lock.

file mode. An object containing the file permission bits and other characteristics of a file. [POSIX.1]

file mode creation mask. See *mask*.

filename. (1) A name assigned or declared for a file. (2) The name used by a program to identify a file. (3) A name consisting of 1 to [NAME_MAX] bytes used to name a file. The characters composing the name may be selected from the set of all character values excluding the slash character and the null character. The filenames dot and dot-dot have special meaning. Synonymous with pathname component. See also *dot, dot-dot.* [POSIX.1] (4) See also *label.*

file offset. The byte position in the file where the next I/O operation begins. Each open file description associated with a regular file, block special file, or directory has a file offset. A character special file that does not refer to a terminal device may have a file offset. There is no file offset specified for a pipe or FIFO. [POSIX.1]

file owner. The user who has the highest level of access authority to a file, as defined by the file.

file permission bits. Information about a file that is used, along with other information, to determine if a process has read, write, or execute/search permission to a file. The bits are divided into three parts: owner, group, and other. Each part is used with the corresponding file class of processes. These bits are contained in the file mode. [POSIX.1]

file serial number. A per-file system unique identifier for a file. File serial numbers are unique throughout a file system. [POSIX.1]

file system. (1) A collection of files and directories. (2) The collection of files and file management structures on a physical or logical mass storage device, such as a disk or disk partition. A single device can contain several file systems. (3) A mountable subtree of the directory hierarchy. [OSF] (4) A collection of files and certain of their attributes. A file system provides a name space for file serial numbers referring to those files. [POSIX.1] (5) See also *mountable file system.*

file system type. (1) A design for file management, and the rules for such a design. (2) The name of a program that processes file systems and files.

file type. One of the five possible types of files: ordinary file, directory, block device, character device, and first-in-first-out (FIFO or named pipe). See also *file.*

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. [A] Contrast with *last-in-first out (LIFO).*

flag. (1) A modifier that appears on a command line with the command name that defines the action of the command. [OSF] (2) An indicator or parameter that shows the setting of a switch. [OSF] (3) A variable indicating that a certain condition holds. [T] (4) A character that signals the occurrence of some condition, such as the end of a word. [A] (5) An internal indicator that describes a condition to the CPU. [OSF] Synonymous with condition code.

foreground. (1) An OpenEdition process that an interactive user starts running and that can interact with the user. An interactive user can move a foreground process to the background. (2) A mode of program execution in which the shell waits for the program specified on the command line to complete before responding to user input. (3) In multiprogramming, the environment in which high-priority programs are executed. (4) The interactive execution of programs and services. (5) The environment in which interactive programs are executed. Interactive processors reside in the foreground. (6) Contrast with *background.*

foreground process. (1) A process that is a member of a foreground process group. [POSIX.1] (2) A process that must run to completion before another command is issued to the shell. The foreground process is in the foreground process group.

foreground process group. A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal. Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that controlling terminal. [POSIX.1] The foreground process group receives the signals generated by the terminal.

foreground process group ID. The process group ID of the foreground process group. [POSIX.1]

foreground processing. The execution of a computer program that preempts the use of computer facilities. [I][A]

fork. To create and start a child process. Forking is similar to creating an address space and attaching. Forking creates a copy of the parent process, including open file descriptors.

forked address space. An address space created by a **fork** function.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. [OSF] (2) The pattern that determines how data is recorded. [OSF] (3) To arrange such things as characters, fields, and lines. (4) In programming languages, a language construct that specifies the representation, in character form, of data objects in a file. [I]

fully qualified name. A qualified name that includes all names in the hierarchical sequence above the structure member to which the name refers, as well as the name of the member itself. [OSF]

G

group. (1) A collection of users who can share access authorities for protected resources. [OSF] (2) A list of names that are known together by a single name. (3) A set of related records that have the same value for a particular field in all records. (4) A series of records logically joined together.

group ID (GID). (1) A unique number assigned to a group of related users. The GID can often be substituted in commands that take a group name as an argument. (2) A nonnegative integer, which can be contained in an object of type *gid_t,* that is used to identify a group of system users. Each system user is a member of at least one group. When the identity of a group is associated with a process, a group ID value is referred to as a real group ID, an effective group ID, one of the (optional) supplementary group IDs, or an (optional) saved set-group-ID. [POSIX.1] (3) Synonymous with group number.

group name. A name that uniquely identifies a group of users to the system.

H

hard link. (1) A mechanism that allows the **ln** command to assign more than one name to a file. Both the new name and the file being linked must be in the same file system. [OSF] (2) The relationship between two directory entries that represent the same file; the result of an execution of the **ln** utility or the POSIX.1 **link()** function. [POSIX.2]

home directory. (1) The current directory associated with the user at the time of login. [POSIX.2] (2) A directory associated with an individual user. (3) The user's current directory on login or after issuing the **cd** command with no argument.

I

inherit. To copy resources or attributes from a parent to a child.

input/output (I/O). (1) Pertaining to a device whose parts can perform an input process and an output process at the same time. [I] (2) Pertaining to a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process. (3) Pertaining to input, output, or both.

J

job control. (1) Facilities for monitoring and accessing background processes. [OSF] (2) A facility that allows users to selectively stop (suspend) the execution of processes and continue (resume) their execution at a later point. The user typically employs this facility via the interactive interface jointly supplied by the terminal I/O driver and a command interpreter. Conforming implementations may optionally support job control facilities; the presence of this option is indicated to the application at compile time or run time by the definition of the [_POSIX_JOB_CONTROL] symbol. [POSIX.1]

K

kernel. (1) The part of the OS/390 OpenEdition component containing programs for such tasks as I/O, management, and communication. (2) The part of the system that is an interface with the hardware and provides services for other system layers such as system calls, file system support, and device drivers. (3) The part of an operating system that performs basic functions such as allocating hardware resources. (4) A program that can run under different operating system environments. See also *shell*. (5) A part of a program that must be in central storage in order to load other parts of the program. (6) Synonym for *kernel address space*.

kernel address space. The address space containing the MVS support for OpenEdition services. This address space can also be called the kernel. See also *kernel*.

kernel mode. In a multiprocessor environment, the master in a master-slave relationship. The master processor operates in kernel mode, and the slave processor operates only in user mode. Kernel mode handles the interrupts and callable services. User mode informs the master when issuing a callable service. Contrast with *user mode*.

kill. An operating system command that stops a process. [OSF]

L

link. (1) In the file system, a connection between an inode and one or more file names associated with it. (2) Synonym for *directory entry.* (3) In data communication, a transmission medium and data link control component that together transmit data between adjacent nodes. [OSF] (4) In programming, the part of a program that passes control and parameters between separate portions of the computer program. (5) To interconnect items of data or portions of one or more computer programs: for example, the linking of object programs by a linkage editor or linking data items by pointers. [T] (6) In SNA, the combination of the link connection and the link stations joining network nodes—for example, a System/370 channel and its associated protocols in a serial-by-bit connection under the control of Synchronous Data Link Control (SDLC).

link count. The number of directory entries that refer to a particular file. [POSIX.1]

login name. (1) A string of characters that uniquely identifies a user to the system. (2) A user name that is associated with a login. [POSIX.1]

M

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters. [I][A]

master. In a multiprocessor environment, the designation for the processor that operates in kernel mode, running unparallelized code. The other processor is the slave and operates in user mode. Synonymous with master processor.

mode. (1) A method of operation. (2) A method of operation, frequently used in UNIX to refer to read, write, run, or search permissions of a file or directory. [OSF] (3) A collection of attributes that specifies a file's type and its access permissions. [POSIX.1]

mount. (1) To make a file system accessible. [OSF] (2) To logically mount a file system in another file system with the TSO/E command MOUNT. The mount point is in a directory. (3) The action taken by an NFS client to use file systems that an NFS server has made available. [OSF] (4) See also *file system, mount point*.

mountable file system. A file system stored in an HFS data set and, therefore, able to be logically mounted in another file system.

mount point. (1) The pathname of the directory on which the file system is mounted. (2) The local directory of an NFS client where the remote directory is mounted. [OSF] (3) Either the root directory or a directory for which the *st_rdev* field of the POSIX.1 *struct stat* differs from that of its parent directory. [POSIX.2]

MVS/ESA. Multiple Virtual Storage/Enterprise Systems Architecture.

N

node. (1) An endpoint of a link, or a junction common to two or more links in a network. Nodes can be processors, controllers, or workstations, and they can vary in routing and other functional capabilities. [OSF] (2) In a tree structure, a point at which subordinate items of data originate. [A] (3) In SNA, an endpoint of a link or a junction common to two or more links in a network. Nodes can be distributed to host processors, communication controllers, or terminals. Nodes can vary in routing and other functional capabilities. (4) In ACF/VTAM, a point in a network defined by a symbolic name.

O

open file. A file that is currently associated with a file descriptor. [POSIX.1]

option. (1) A specification in a statement that can influence the execution of the statement. (2) An argument to a command that is generally used to specify changes in the *utility*'s default behavior. [POSIX.2]

output file. (1) A file that a program opens so that it can write to that file. (2) A file that contains the results of processing.

P

parent directory. (1) The directory one level above the current directory. (2) When discussing a given directory, the directory that both contains a directory entry for the given directory and is represented by the pathname dot-dot in the given directory. [POSIX.1] (3) When discussing other types of files, a directory containing a directory entry for the file under discussion. [POSIX.1]

parent process. A process created to carry out a program. The parent process in turn creates child processes to execute requests. Contrast with *child process*. See also *parent process ID, process*.

parent process ID. An attribute of a new process after it is created by a currently active process. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an implementation-defined system process. [POSIX.1] In the OpenEdition implementation, the parent process ID of the children of an ended process is set to the process ID of the INIT process, or 1.

path. (1) In a network, any route between any two nodes. [T] (2) In a database, a sequence of segment occurrences from the root segment to an individual segment.

PATH. An environment variable that lists which directories on the disk need to be searched for a command. OpenEdition Shell and Utilities binaries should be placed in a directory on this search list. See also *current directory, directory*.

pathname. (1) A filename specifying all directories leading to the file. (2) See also *relative pathname*. (3) A filename specifying all directories leading to a file plus the filename itself. (4) A string that is used to identify a file. A pathname consists of, at most, [PATH_MAX] bytes, including the terminating null character. It has an optional beginning slash, followed by zero or more filenames separated by slashes. If the pathname refers to a directory, it may also have one or more trailing slashes. Multiple successive slashes are considered to be the same as one slash. A pathname that begins with two successive slashes may be interpreted in an implementation-defined manner, although more than two leading slashes shall be treated as a single slash. [POSIX.1] In the OpenEdition implementation, the C/370 functions **fopen()**, **freopen()**, **remove()**, and **rename()** interpret names with exactly two leading slashes, no leading blanks or other characters, and the third character not a slash to mean that the rest of the name refers to a traditional MVS data set. (5) See also *relative pathname*.

pathname component. Synonym for *filename.*

path prefix. A pathname, with an optional ending slash, that refers to a directory. [POSIX.1]

pending. Waiting, as in an operation that is pending. [OSF]

permission. A code that determine how the file can be used by any users who work on the system. [OSF]

PID. Process ID

pipe. (1) An interprocess communication mechanism that connects an output file descriptor to an input file descriptor. Usually the standard output of one process is connected to the standard input of another, forming a pipeline. (2) A sequence of one or more commands in FIFO order. The output of one command becomes the input to the next command. A pipe usually contains several filters. Pipes allow related or unrelated processes to read and write to each other as if they were files; they allow unidirectional communication from one process to another. OpenEdition Services treats pipes as though they were files. A named pipe has a directory name and is accessed by a pathname. An unnamed pipe must be used between a parent process and a child process. (3) An object accessed by one of the pair of file descriptors created by the **pipe()** function. Once it is created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO special file when accessed in this way. It has no name in the file hierarchy. [POSIX.1] (4) To direct data so that the output from one process becomes the input to another process. (5) An I/O stream that has a descriptor and can be used in unidirectional communications between related processes. [OSF] (6) See also *pipeline*.

pipeline. (1) A chain of two or more processes connected by pipes. Each process in the chain acts as a filter, reading data from the standard input, performing some transformation, and writing the results to the standard output. (2) A direct, one-way connection between two or more processes. (3) A serial arrangement of processors or a serial arrangement of registers within a processor. Each processor or register performs part of a task and passes results to the next processor. Several parts of different tasks can be performed at the same time. (4) To perform processes in a series. (5) For increased processing speed, to start execution of an instruction sequence before the previous instruction sequence is completed.

Portable Operating System Interface. See *POSIX*.

POSIX. Portable Operating System Interface for Computer Environments, an interface standard governed by the IEEE and based on UNIX. POSIX is not a product. Rather, it is an evolving family of standards describing a wide spectrum of operating system components ranging from C language and shell interfaces to system administration.

privilege. See *appropriate privileges.*

privileged user. A user logged into an account with root user authority.

process. (1) A function being performed or waiting to be performed. (2) An executing function, or one waiting to execute. (3) A function, created by a **fork()** request, with three logical sections:

- Text, which is the function's instructions.
- Data, which the instructions use but do not change.
- Stack, which is a push-down, pop-up save area of the dynamic data that the function operates upon.

The three types of processes are:

- User processes, which are associated with a user at a workstation
- Daemon processes, which do systemwide functions in user mode, such as printer spooling
- Kernel processes, which do systemwide functions in kernel mode, such as paging

A process can run in a user address space, a forked address space, or a kernel address space. In an MVS system, a process is handled like a task. See also *task*.

(4) An address space and one or more threads of control that execute within that address space, and their required system resources. [POSIX.0] (5) An address space and single thread of control that executes within that address space, and its required system resources. A process is created by another process issuing the **fork()** function. The process that issues **fork()** is known as the parent process, and the new process created by the **fork()** is known as the child process. [POSIX.1] (6) A sequence of actions required to produce a desired result. [OSF] (7) An entity receiving a portion of the processor's time for executing a program. [OSF] (8) An activity within the system that is started by a command, a shell program, or another process. Any running program is a process. (9) A unique, finite course of events defined by its purpose or by its effect, achieved under given conditions. (10) Any operation or combination of operations on data. (11) The current state of a program that is running—including a memory image, the program data, the variables used, the general register values, the status of opened files used, and the current directory. Programs running in a process must be either operating system programs or user programs. [OSF] (12) A running program, including the memory occupied, the open files, the environment, and other attributes specific to a running program.

process group. A collection of processes that permits the signaling of related processes. Each process in the system is a member of a process group that is identified by a process group ID. A newly created process joins the process group of its creator. [POSIX.1]

process group ID. The unique identifier representing a process group during its lifetime. A process group ID is a positive integer that can be contained in a *pid_t.* It shall not be reused by the system until the process group lifetime ends. [POSIX.1]

process group leader. A process whose process ID is the same as its process group ID. [POSIX.1]

process ID (PID). (1) A unique number assigned to a process that is running. [OSF] (2) The unique identifier representing a process. A process ID is a positive integer that can be contained in a *pid_t*. A process ID shall not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID shall not be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of 1. [POSIX.1]

profile. (1) A file containing customized settings for a system or user. [OSF] (2) Data that describes the significant characteristics of a user, a group of users, or one or more computer resources. (3) A set of instructions to initialize a user's shell session. The shell automatically reads and executes these commands if this file is in the user's home directory. (4) In computer security, a description of the characteristics of an entity to which access is controlled. (5) A set of one or more base standards, and, where applicable, the identification of chosen classes, subsets, options, and parameters of those base standards, necessary for accomplishing a particular function. [POSIX.0]

pseudoterminal (pty). A special file in the **/dev** directory that effectively functions as a keyboard and display device. Synonymous with pseudo-TTY.

pseudo-TTY. Synonym for *pseudoterminal*.

pty. Pseudoterminal.

Q

quiesce. To end a process by allowing operations to complete normally.

R

RACF. Resource Access Control Facility.

read lock. A lock that prevents any other process from setting a write lock on any part of the protected area. [OSF] Contrast with *write lock*. See also *lock*.

read-only file. In file system mounting, a file whose data can be read but not copied, printed, or modified. Contrast with *editable file version*.

read-only file system. A file system that has implementation-defined characteristics restricting modifications. [POSIX.1] In the OpenEdition implementation, it is a file system specified as read-only on the MOUNT command or parmlib statement that mounted the file system. No updates are made or allowed to a read-only file system. Writes are permitted to FIFO special files within a read-only file system, but FIFO data is kept only in memory and no updates are made to the disk.

real group ID. (1) For each user, any group ID defined in the password file. [OSF] (2) The attribute of a process that, at the time of process creation, identifies the group of the user who created the process. See also *group ID*. [POSIX.1]

regular file. A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system. [POSIX.1]

relative pathname. (1) The name of a directory or file expressed as a sequence of directories followed by a filename, beginning from the current directory. Relative pathnames do not begin with a / (slash) but are relative to the current directory. (2) A pathname that does not begin with a slash. The predecessor of the first filename in the pathname is taken to be the current working directory of the process. [POSIX.1] See also *absolute pathname, pathname*.

Resource Access Control Facility (RACF). An IBM-licensed program that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, and logging detected unauthorized access to protected resources.

root. (1) The starting point of the file system. (2) The first directory in the file system.

root directory. (1) The directory that contains all other directories in the system. (2) The lowest directory in the file system hierarchy. It is referred to as "/." (3) A directory, associated with a process, that is used in pathname resolution for pathnames that begin with a slash. [POSIX.1] (4) The top directory in the file system tree. UNIX and POSIX-conforming systems have a single root directory, with mounted devices.

S

saved set-group-ID. An attribute of a process that allows some flexibility in the assignment of the effective group ID attribute, when the saved set-user-ID option is implemented. [POSIX.1]

saved set-user-ID. An attribute of a process that allows some flexibility in the assignment of the effective user ID attribute, when the saved set-user-ID option is implemented. [POSIX.1]

seconds since the Epoch. A value to be interpreted as the number of seconds between a specified time and the Epoch. A Coordinated Universal Time name, specified in terms of seconds (*tm_sec*), minutes (*tm_min*), hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900 (*tm_year*) is related to a time represented as seconds since the Epoch, according to the following expression: If the year < 1970 or the value is negative, the relationship is undefined. If the year ≥ 1970 and the value is nonnegative, the value is related to a Coordinated Universal Time name according to the expression:

$$tm\_sec + tm\_min*60 + tm\_hour*3\ 600 + tm\_yday*86\ 400 + (tm\_year-70)*31\ 536\ 000 + ((tm\_year-69)/4)*86\ 400$$

[POSIX.1]

session. (1) The period of time during which a user of a terminal can communicate with an interactive system—usually, the elapsed time between logon and logoff. (2) The period of time during which programs or devices can communicate with each other. [OSF] (3) A collection of process groups established for job control purposes. Each process group is a member of a session. Each process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership. Implementations that support the **setpgid()** function can have multiple process groups in the same session. [POSIX.1] Every process group, and associated process, belongs to a session. Any new process also belongs to the session of the process that created it. (4) In network architecture, an association of facilities that establish, maintain, and release connections for communication between stations. [OSF] (5) In SNA, a logical connection established between two network addressable units (NAUs) that allows them to communicate. For routing purposes each session is identified by the local or network addresses of the session partners.

session leader. A process that has created a session. [POSIX.1]

set-group-ID mode bit. In setting file access permissions, the bit that sets the effective group ID of the process to the file's group on execution. Synonymous with S_ISGID bit.

set-user-ID mode bit. In setting file access permissions, the bit that sets the effective user ID of the process to the file's owner on execution. Synonymous with S_ISUID bit.

shell. (1) A program that interprets and processes interactive commands from a pseudoterminal or from lines in a shell script. (2) A program that interprets sequences of text input as commands. It may operate on an input stream, or it may interactively prompt and read commands from a terminal. [POSIX.2] Synonymous with command language interpreter. (3) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices and touch-sensitive screens and communicate them to the operating system. (4) The command interpreter that provides a user interface to the operating system and its commands. (5) The program that reads a user's commands and executes them. (6) The shell command language interpreter, a specific instance of a shell. [POSIX.2] (7) A layer, above the kernel, that provides a flexible interface between users and the rest of the system. (8) Software that allows a kernel program to run under different operating system environments.

signal. (1) A means of informing processes of asynchronous events. (2) A mechanism by which a process may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term *signal* is also used to refer to the event itself. [POSIX.1] (3) An indication that an asynchronous event completed. A signal is sent to a process. Signals are simulations of *interrupts*. (4) A simple method of communication between two processes. One process can inform the other process when an event occurs. [OSF] (5) A method of interprocess communication that simulates software interrupts. Contrast with *exception, interrupt*.

signal handler. A subroutine called when a signal occurs. [OSF]

signal mask. A mask that defines the set of signals currently blocked from delivery to a process.

standard error (stderr). (1) The place where many programs place error messages: the display screen unless another place is specified with redirection. [OSF] (2) An output stream usually intended to be used for diagnostic messages. [POSIX.2] (3) The conventional name for file descriptor 2. By convention, programs write diagnostics and error messages to this descriptor. Usually, the descriptor refers to the display screen, but it may be changed by redirection. This descriptor is separate from standard output so that error diagnostics are still visible when the output is redirected.

standard input (stdin). (1) The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command. (2) An input stream usually intended to be used for primary data input. [POSIX.2] (3) The conventional name for file descriptor 0. By convention, programs read input from this descriptor. Usually, the descriptor refers to the keyboard, but it may be changed by redirection.

standard output (stdout). (1) The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command. (2) An output stream usually intended to be used for primary data output. [POSIX.2] (3) The conventional name for file descriptor 1. By convention, programs write output to this descriptor. Usually, the descriptor refers to the display screen, but may be changed by redirection.

stderr. Standard error.

stdin. Standard input.

stdout. Standard output.

sticky bit. A file access permission bit that allows multiple users to share a single copy of an executable file. Only someone with root authority can set the sticky bit.

subdirectory. In the file system hierarchy, a directory contained within another directory. Directories may be nested to arbitrary depth.

suffix. A character string attached to the end of a filename that helps identify its file type. [OSF] For example, in **/dir/file.c**, the suffix is **c**.

superuser. A system user who operates without restrictions. A superuser has the special rights and privileges needed to perform administrative tasks.

superuser authority. The unrestricted ability to access and modify any part of the operating system, usually associated with the user who manages the system.

supplementary group ID. An attribute of a process used in determining file access permissions. A process has up to [NGROUPS_MAX] supplementary group IDs in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created. Whether a process's effective group ID is included in or omitted from its list of supplementary group IDs is unspecified. [POSIX.1] In the OpenEdition implementation, a supplementary group ID is an attribute of a process used in determining file access permissions.

symbolic link. A type of file system entry that contains the pathname of and acts as a pointer to another file or directory. [OSF]

T

thread. (1) A single flow of control within a process. [POSIX.0] (2) A single, sequential flow of control.

U

undub. The inverse of *dub*. Normally, a task (dubbed a thread) is undubbed when it ends. An address space (dubbed a process) is undubbed when the last dubbed thread ends. Contrast with *dub*.

UNIX. A highly portable operating system originally developed by Bell Laboratories that features multiprogramming in a multiuser environment. UNIX is implemented in the C language. UNIX was originally developed for use on minicomputers but has been adapted on mainframes and microcomputers. It is especially suitable for multiprocessor, graphics, and vector-processing systems. Many of the commands in the OpenEdition shell are based on similar commands available with UNIX System V.

unmount. To logically disassociate a mountable file system from another file system. The TSO/E command to perform this action is UNMOUNT or UMOUNT.

user address space. An address space that has at least one MVS task known to the kernel address space. This address space can contain a shell or an application program that uses OpenEdition services.

user ID. (1) A unique string of characters that identifies an operator to the system. This string of characters limits the functions and information the operator can use. (2) A nonnegative integer, which can be contained in an object of type *uid_t*, that is used to identify a system user. When the identity of the user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or an (optional) saved set-user-ID. [POSIX.1] (3) The identification associated with a user or job. The two types of user IDs are:

- **TSO/E user ID:** A string of characters that uniquely identifies a TSO/E user or a batch job owner to the security program for the system. The batch job owner is specified on the USER parameter on the JOB statement or inherited from the submitter of the job. This user ID identifies a RACF user profile.
- **OpenEdition user ID:** A fullword integer that the security administrator assigns to each MVS user ID. This integer, referred to as the UID, is the sole basis for authority checking against such POSIX-defined resources as hierarchical files.

A user ID is equivalent to an account on other open systems. (4) A symbol identifying a system user. (5) Synonymous with user identification

user mode. In a multiprocessor environment, the slave in a master-slave relationship. One processor operates in kernel mode, the other (slave) in user mode. Kernel mode handles the interrupts and callable service. User mode informs the master when making a callable service. In user mode, a process is carried out in the user's program rather than in the kernel. Contrast with *kernel mode*.

user name. (1) In RACF, one to twenty alphanumeric characters that represent a RACF-defined user. (2) A string that is used to identify a user. [POSIX.1] (3) Synonym for *user ID*.

V

VFS. Virtual file system.

vnode. Virtual inode. An object in a file system that represents a file. Unlike an inode, there is no one-to-one correspondence between a vnode and the file system; multiple vnodes can refer to a single file. Vnodes are used to communicate between the upper half of the file system and the file system representations.

W

working directory. (1) The active directory used to resolve pathnames that do not begin with a slash. In similar systems, a working directory may be called the *current directory* or the *current working directory*. (2) A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash. [POSIX.1] (3) Synonym for *current directory*. (4) See also *current directory*.

write access. In computer security, permission to write to an object. Synonymous with write permission.

write lock. A lock that prevents any other process from setting a read lock or a write lock on any part of the protected area. Contrast with *read lock*. See also *lock*.

write permission. Synonym for *write access*.

3270 passthrough mode. A mode that lets a program running from the OpenEdition shell send and receive a 3270 data stream or issue TSO/E commands.

Index

Communicating Your Comments to IBM

OS/390
Using REXX and OS/390 OpenEdition

Publication No. SC28-1905-02

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing an RCF from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - FAX: (International Access Code)+1+914+432-9405
- If you prefer to send comments electronically, use this network ID:
  - IBMLink: (United States customers only): KGNVMC(MHVRCFS)
  - IBM Mail Exchange: USIB6TC9 at IBMMAIL
  - Internet e-mail: mhvrcfs@vnet.ibm.com
  - World Wide Web: http://www.s390.ibm.com/os390

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

Reader's Comments — We'd Like to Hear from You

OS/390
Using REXX and OS/390 OpenEdition

Publication No. SC28-1905-02

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: ______________________________________

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ]	As an introduction	[ ]	As a text (student)
[ ]	As a reference manual	[ ]	As a text (instructor)
[ ]	For another purpose (explain)		

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Name	Address
Company or Organization	
Phone No.	

Reader's Comments — We'd Like to Hear from You
SC28-1905-02

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie NY 12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

SC28-1905-02

Cut or Fold
Along Line

File Number: S370/S390-39
Program Number: 5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC28-1905-02