Using REXX and z/OS UNIX System Services
Contents

Figures ........................................ vii

Tables ........................................... ix

About this document ............................ xi

Who should read Using REXX and z/OS UNIX System Services .............................. xi

Where to find more information ............ xi

Softcopy publications ........................... xi

IBM Systems Center publications ............. xi

Porting information for z/OS UNIX .......... xii

z/OS UNIX courses ............................... xii

z/OS UNIX home page ............................ xii

Discussion list .................................. xii

The z/OS Basic Skills Information Center . . xii

Finding more information about REXX .......... xiii

How to read syntax diagrams ............... xiii

Symbols ........................................... xiii

Syntax items ..................................... xiii

Syntax examples ................................ xiv

Summary of changes ............................ xvii

Changes made in z/OS Version 1 Release 13 . xvii

Changes made in z/OS Version 1 Release 12 . xvii

Changes made in z/OS Version 1 Release 11 . xviii

How to send your comments to IBM ....... xix

If you have a technical problem ............. xix

Chapter 1. Using TSO/E REXX for z/OS UNIX processing .................................. 1

Host command environments for z/OS UNIX processing .................................. 1

The SYSCALL environment ....................... 2

Running a REXX program from TSO/E or MVS batch ...................................... 2

Establishing the SYSCALL environment .... 2

Ending the SYSCALL environment .......... 3

Establishing and deleting the signal interface routine .................................. 3

The SH environment ................................ 3

Running a REXX program from the z/OS shells or from a program ................. 4

Using external functions and subroutines .................................................. 4

The TSO command environment ............... 5

Command input ................................... 5

Command output .................................. 5

Return codes .................................... 6

Examples ........................................ 6

Variable scope ................................... 7

Writing setuid and setgid REXX programs ............................................. 7

Input and output for z/OS UNIX processing ................................................ 7

Using standard input, output, and error (file descriptors 0, 1, and 2) .......... 7

Using SYSCALL commands ..................... 8

Using EXECIO .................................... 8

Exit status from a REXX program .......... 9

Tokens returned from the PARSE SOURCE instruction .................................. 9

Running from a z/OS shell or from a program ........................................... 9

Running from TSO/E or batch ................. 10

Using the REXX signal services ............. 10

Using immediate commands ................... 11

Moving a REXX program from TSO/E to a z/OS shell .................................. 11

Using argv and environment variables .... 12

Customizing the z/OS UNIX REXX environment ........................................ 13

Performance in the SYSCALL environment ............................................. 13

Authorization .................................... 14

Chapter 2. z/OS UNIX REXX programming services .................................. 15

Creating a z/OS UNIX REXX environment from an application ....................... 15

Running the REXX program .................... 16

Example: C/370 program ....................... 17

Chapter 3. The syscall commands .......... 19

Specifying a syscall command ................ 19

Specifying numerics ......................... 20

Specifying strings ............................. 20

Using predefined variables ................. 21

Return values .................................... 21

Returned from the SYSCALL environment ............................................. 21

Returned from the SH environment .......... 22

Syscall command descriptions ............... 22

access ............................................. 23

acldel ............................................ 24

acldelteentry .................................. 25

acffree ........................................... 26

acget ............................................. 27

aclgetentry ..................................... 28

aclinit .......................................... 29

aclset ............................................ 30

aclupdateentry .................................. 31

alarm .............................................. 33

catclose ........................................ 34

catget .......................................... 35

catopen .......................................... 36

cert ............................................. 37

chattr ............................................ 38

chaudit ........................................... 41

chdir ............................................. 42

chmod ............................................. 43

chown ............................................. 45

close ............................................ 46

closedir ......................................... 47

creat ............................................. 48

dup ............................................... 49

© Copyright IBM Corp. 1996, 2011
Chapter 4. Examples: Using syscall commands

Read the root directory into a stem and print it.
Open, write, and close a file.
Open a file, read, and close it.
Display the working directory and list a specified directory ........................................ 206
Parse arguments passed to a REXX program: the getopts function ........................................ 206
Count newlines, words, and bytes ........................................ 208
Obtain information about the mounted file system ........................................ 209
Mount a file system ........................................ 209
Unmount a file system ........................................ 211
Run a shell command and read its output into a stem ........................................ 213
Print the group member names ........................................ 213
Obtain information about a user ........................................ 214
Set up a signal to enforce a time limit for a program ........................................ 214
List the ACL entries for a file ........................................ 216

Chapter 5. z/OS UNIX REXX functions 217
REXX I/O functions ........................................ 217
Opening a stream implicitly ........................................ 217
Opening a stream explicitly ........................................ 217
bpxwunix() ........................................ 219
charin() ........................................ 221
charout() ........................................ 222
chars() ........................................ 223
chmod() ........................................ 224
convd2e() ........................................ 225
directory() ........................................ 226
environment() ........................................ 227
exists() ........................................ 228
getpass() ........................................ 229
linein() ........................................ 230
lineout() ........................................ 231
lines() ........................................ 232
outtrap() ........................................ 233
procinfo() ........................................ 234
rexxopt() ........................................ 238
sleep() ........................................ 239
stream() ........................................ 240
submit() ........................................ 244
syscalls() ........................................ 245

Chapter 6. BPXWDYN: a text interface to dynamic allocation and dynamic output ........................................ 247
Calling conventions ........................................ 247
REXX external function parameter list ........................................ 247
Conventional MVS variable-length parameter string ........................................ 247
Null-terminated parameter string ........................................ 248
Request types ........................................ 248
Keywords ........................................ 249
Return codes ........................................ 249
Key errors ........................................ 250
Error codes for dynamic allocation ........................................ 250
Error codes for dynamic output ........................................ 250
Message processing ........................................ 250
Requesting dynamic allocation ........................................ 251
Requesting dynamic concatenation ........................................ 254
Requesting dynamic unallocation ........................................ 255
Requesting allocation information ........................................ 256
Requesting dynamic output ........................................ 257
Freeing an output descriptor ........................................ 259
Examples: Calling BPXWDYN from a REXX program ........................................ 259
Example: calling BPXWDYN from C ........................................ 259

Chapter 7. Virtual file system (VFS) server syscall commands ........................................ 261
Security ........................................ 261
Tokens ........................................ 261
v_create ........................................ 262
v_fstatfs ........................................ 264
v_get ........................................ 265
v_getattr ........................................ 266
v_link ........................................ 267
v_lockctl ........................................ 268
v_lookup ........................................ 273
v_mkdir ........................................ 274
v_read ........................................ 275
v_readdir ........................................ 276
v_readlink ........................................ 277
v_reg ........................................ 278
v_rel ........................................ 279
v_remove ........................................ 280
v_rename ........................................ 281
v_rmdir ........................................ 282
v_rpn ........................................ 283
v_setattr ........................................ 284
v_symlink ........................................ 285
v_write ........................................ 287

Chapter 8. Examples: Using virtual file system syscall commands ........................................ 289
List the files in a directory ........................................ 289
Remove a file or empty directory ........................................ 289

Appendix A. REXX predefined variables ........................................ 291

Appendix B. Setting permissions for files and directories ........................................ 303
Position 1 ........................................ 303
Positions 2, 3, and 4 ........................................ 303
Example: using BITOR and BITAND to set mode bits ........................................ 304

Appendix C. Accessibility ........................................ 305
Using assistive technologies ........................................ 305
Keyboard navigation of the user interface ........................................ 305
z/OS information ........................................ 305
Dotted decimal syntax diagrams ........................................ 305

Notices ........................................ 309
Policy for unsupported hardware ........................................ 310
Trademarks ........................................ 310
Acknowledgments ........................................ 311

Index ........................................ 313
Figures

1. EXECIO syntax diagram .................... 9
2. Example of parameter list .................. 248
3. Example of a null-terminated parameter string .................... 248
4. Three-digit permissions specified in octal format .................... 304
## Tables

1. Syntax examples ........................................... xiv
2. REXX statements for defining signal sets ............ 10
3. Common keys used for dynamic allocation .......... 251
4. Additional keys used for dynamic allocation ....... 254
5. Keys used for dynamic concatenation ................. 254
6. Common keys used for dynamic unallocation ...... 255
7. Additional keys used for dynamic unallocation .... 255
8. Keys used to specify which allocation request information is to be returned for ............ 256
9. Additional keys used for dynamic information retrieval ............................................ 256
10. Keys used for dynamic output .......................... 257
11. Key used to free an output descriptor ............ 259
12. List of predefined variables ............................ 291
About this document

This document presents the information you need to write REXX programs that access z/OS UNIX System Services (z/OS UNIX). It describes the features and usage requirements for the z/OS UNIX REXX extensions, or syscall commands, which are interfaces between the z/OS 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 z/OS UNIX. This document also describes syscall commands that are not related to the standards.

Who should read Using REXX and z/OS UNIX System Services

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


Where to find more information

Where necessary, this document references information in other documents about the elements and features of the z/OS® system. For complete titles and order numbers for all z/OS documents, see [z/OS Information Roadmap](http://www.ibm.com/systems/z/os/zos/bkserv/).

**Softcopy publications**

The z/OS UNIX library is available on the [z/OS and Software Products DVD Collection](http://www.ibm.com/servers/eserver/zseries/zos/bkserv/), SK2T-4271.


**IBM Systems Center publications**

IBM® Systems Centers produce IBM Redbooks® publications that can be helpful in setting up and using z/OS UNIX. See the IBM Redbooks site at [http://www.ibm.com/redbooks/](http://www.ibm.com/redbooks/).

These documents 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 information on a wide range of topics. You must order them separately. A selected list of these documents is on the z/OS UNIX website at [http://www.ibm.com/systems/z/os/zos/features/unix/library/](http://www.ibm.com/systems/z/os/zos/features/unix/library/).
Porting information for z/OS UNIX

A Porting Guide is available at [http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1por.html](http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1por.html). It covers a range of useful topics, including sizing a port, setting up a porting environment, ASCII-EBCDIC issues, performance, and much more.

The porting page also features a variety of porting tips and lists porting resources that will help you in your port.

z/OS UNIX courses

For a current list of courses that you can take, go to [http://www.ibm.com/services/learning/](http://www.ibm.com/services/learning/).

z/OS UNIX home page


Some of the tools available from the website are ported tools, and some are unsupported tools designed for z/OS UNIX. The 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 lists any restrictions.

The simplest way to reach these tools is through the z/OS UNIX home page. From the home page, click on **Tools and Toys**.


---

Restrictions

Because the tools are not officially supported, APARs cannot be accepted.

Discussion list

Customers and IBM participants also discuss z/OS UNIX on the mvs-oe discussion list. This list is not operated or sponsored by IBM.

To subscribe to the mvs-oe discussion, send a note to:

listserv@vm.marist.edu

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

subscribe mvs-oe given_name family_name

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

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a Web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center
is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS system programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

- Provide basic education and information about z/OS without charge
- Shorten the time it takes for people to become productive on the mainframe
- Make it easier for new people to learn z/OS.

To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required):

http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp

---

**Finding more information about REXX**


**How to read syntax diagrams**

This section describes how to read syntax diagrams. It defines syntax diagram symbols, items that might be contained within the diagrams (keywords, variables, delimiters, operators, fragment references, operands) and provides syntax examples that contain these items.

Syntax diagrams pictorially display the order and parts (options and arguments) that comprise a command statement. They are read from left to right and from top to bottom, following the main path of the horizontal line.

**Symbols**

The following symbols may be displayed in syntax diagrams:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>⬅️ ➤️</td>
<td>Indicates the beginning of the syntax diagram.</td>
</tr>
<tr>
<td>➤️ ➤️</td>
<td>Indicates that the syntax diagram is continued to the next line.</td>
</tr>
<tr>
<td>➤️ ➤️</td>
<td>Indicates that the syntax is continued from the previous line.</td>
</tr>
<tr>
<td>➤️ ➤️</td>
<td>Indicates the end of the syntax diagram.</td>
</tr>
</tbody>
</table>

**Syntax items**

Syntax diagrams contain many different items. Syntax items include:

- Keywords - a command name or any other literal information.
- Variables - variables are italicized, appear in lowercase, and represent the name of values you can supply.
- Delimiters - delimiters indicate the start or end of keywords, variables, or operators. For example, a left parenthesis is a delimiter.
- Operators - operators include add (+), subtract (-), multiply (*), divide (/), equal (=), and other mathematical operations that may need to be performed.
- Fragment references - a part of a syntax diagram, separated from the diagram to show greater detail.
Separators - a separator separates keywords, variables or operators. For example, a comma (,) is a separator.

**Note:** If a syntax diagram shows a character that is not alphanumeric (for example, parentheses, periods, commas, equal signs, a blank space), enter the character as part of the syntax.

Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

<table>
<thead>
<tr>
<th>Item type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>Required items are displayed on the main path of the horizontal line.</td>
</tr>
<tr>
<td>Optional</td>
<td>Optional items are displayed below the main path of the horizontal line.</td>
</tr>
<tr>
<td>Default</td>
<td>Default items are displayed above the main path of the horizontal line.</td>
</tr>
</tbody>
</table>

**Syntax examples**

The following table provides syntax examples.

<table>
<thead>
<tr>
<th>Item</th>
<th>Syntax example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required item.</td>
<td>[KEYWORD required_item]</td>
</tr>
<tr>
<td>Required choice.</td>
<td>[KEYWORD required_choice1, required_choice2]</td>
</tr>
<tr>
<td>Optional item.</td>
<td>[KEYWORD optional_item]</td>
</tr>
<tr>
<td>Optional choice.</td>
<td>[KEYWORD optional_choice1, optional_choice2]</td>
</tr>
<tr>
<td>Default.</td>
<td>[KEYWORD default_choice1, optional_choice2, optional_choice3]</td>
</tr>
<tr>
<td>Variable.</td>
<td>[KEYWORD variable]</td>
</tr>
<tr>
<td>Item</td>
<td>Syntax example</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| Repeatable item. | An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.  
A character within the arrow means you must separate repeated items with that character.  
An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated. |
| Fragment. | The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram. |
Summary of changes

This document contains terminology, maintenance, and editorial changes to improve consistency and retrievability. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Changes made in z/OS Version 1 Release 13

This document contains information that was previously presented in z/OS Using REXX and z/OS UNIX System Services, SA22-7806-13, which supports z/OS Version 1 Release 12.

New information:

- The pathflag parameter was added to the getmntent command; see “getmntent” on page 99. The MNTE_UID variable was also added.

Changed information:

- These sections were changed to indicate that you need the appropriate mount authorities when mounting or unmounting file systems:
  - “mount” on page 127
  - “unmount” on page 193
  - “Unmount a file system” on page 211

Deleted information:

- The sigdce variable was deleted in various places because Distributed Computing Environment (DCE) is no longer supported.

Changes made in z/OS Version 1 Release 12

This document contains information that was previously presented in z/OS Using REXX and z/OS UNIX System Services, SA22-7806-12, which supports z/OS Version 1 Release 11.

New information:

- Two predefined variables, S_FFCRNL and S_FFRECORD, have been added.
  - “The TSO command environment” on page 5 has been updated to indicate that the tsocmd command is now shipped as part of the base release.

Changed information:

- Clarifications have been added to “The SH environment” on page 3 and “Using EXECIO” on page 8.
- Table 3 on page 251 has been updated with a clarification for the BLOCK key.
- Chapter 6, “BPXWDYN: a text interface to dynamic allocation and dynamic output,” on page 247 has been updated to indicate that arguments that accept REXX variable names will be accepted when BPXWDYN is called from any environment, but are only effective when called from a REXX environment.

The “Readers’ Comments – We’d Like to Hear from You” section at the back of the publication has been replaced with a new “How to send your comments to IBM” section in the front of the publication, located between the “About this document”
Changes made in z/OS Version 1 Release 11

This document contains information that was previously presented in z/OS Using REXX and z/OS UNIX System Services, SA22-7806-11, which supports z/OS Version 1 Release 10.

New information:
- A new variable, MTM_SAMEMODE, was added to “unmount” on page 193.
- A new common key, SUBSYS, was added to “Requesting dynamic allocation” on page 251.
- Two syscall commands, setreuid and setregid, were added to Chapter 3, “The syscall commands,” on page 19.

Changed information:
- The variable MNTE_MODE was changed in “getmntent” on page 99.

Deleted information:
- The REXX-related BPX messages are now in z/OS MVS System Messages, Vol 3 (ASB-BPX).
How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

1. Send an email to mhvrcfs@us.ibm.com
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   U.S.A.
4. Fax the comments to us as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:
• Your name and address
• Your email address
• Your telephone or fax number
• The publication title and order number:
  z/OS V1R13.0 Using REXX and z/OS UNIX System Services
  SA22-7806-14
• The topic and page number related to your comment
• The text of your comment.

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.

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

If you have a technical problem

Do not use the feedback methods listed above. Instead, do one of the following:
• Contact your IBM service representative
• Call IBM technical support
• Visit the IBM support portal at http://www.ibm.com/systems/z/support/
Chapter 1. Using TSO/E REXX for z/OS UNIX processing

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

**Restriction:** Blank spaces cannot precede the opening REXX comment symbol (/ * ).

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

You can run an interpreted or compiled REXX program with syscall commands from TSO/E, from MVS™ batch, from the z/OS shells, or from a program. You can run a REXX program with syscall commands only on a system with z/OS UNIX System Services installed. For a complete description of each of the syscall commands, see Chapter 3, “The syscall commands,” on page 19.

The set of z/OS UNIX REXX functions also extend the REXX language on z/OS in the z/OS UNIX environment. There are functions that provide:

- Standard REXX I/O
- Access to some common file services and environment variables

All of the z/OS UNIX functions, except bpxwunix() and syscalls(), must be run in a z/OS UNIX environment. For a complete description of each of the functions, see Chapter 5, “z/OS UNIX REXX functions,” on page 217.

For dynamic allocation and dynamic output, BPXWDYN is a text interface to a subset of the SVC 99 and SVC 109 services that is designed to be called from REXX. For a complete description of BPXWDYN, see Chapter 6, “BPXWDYN: a text interface to dynamic allocation and dynamic output,” on page 247.

### Host command environments for z/OS UNIX 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 z/OS UNIX extensions. These 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 a z/OS 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 z/OS shell commands, not as syscall commands.
A REXX program can run TSO/E commands, but you cannot use TSO commands to affect your REXX environment, or have REXX statements or other host command environments affect your TSO process. Commands that are addressed to TSO will be run in a TMP running in a separate address space and process from your REXX program. The TSO process is started when the first TSO command is run, and persists until your REXX program terminates or you run the TSO LOGOFF command.

When a REXX program is run from a z/OS 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 z/OS 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 z/OS shells (where the environment is automatically initialized).

Running a REXX program from TSO/E or MVS 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 a process; this is known as dubbing.
- Initializes the predefined 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 10 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 z/OS 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

Rule: 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.
The process was dubbed, but the SYSCALL environment was not established.

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:
```rexx
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 z/OS UNIX. The task is undubbed and the REXX program continues.

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 a z/OS shell or from 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 a z/OS shell or from a program using `exec()`; it is available to a REXX program only in those two situations. In the SH environment, a syscall command runs as a z/OS shell command that has been issued this way:

```
/bin/sh -c shell_command
```
All open file descriptors and environment variables are available to the shell command. Note that built-in shell commands run in the shell process, not your REXX process and cannot alter the REXX environment. For example, address sh 'cd /' will not change the current directory of your REXX process.

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

See Chapter 4, “Examples: Using syscall commands,” on page 205 for some sample REXX programs that show how REXX and z/OS shell commands can work together—for example, a REXX program that can read output from a z/OS shell command. The mount and unmount sample programs are shipped in the /samples directory as files mountx and unmountx.

Running a REXX program from the z/OS shells or from a program

You can run a REXX program from the z/OS shells, or you can call it from any program just as you would call an executable program. The REXX program runs as a separate process; it does not run in a TSO/E address space.

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

If you are working in a z/OS 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 that is run from a z/OS 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.

CEEXEC output from the REXX compiler is supported in the z/OS shell environments. To compile and put CEXEC output into the z/OS UNIX file system, 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 z/OS UNIX file system. The search path for an external routine is similar to that used for a REXX program that is invoked from a z/OS shell or from 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 can also be searched. The default z/OS environment searches for executable modules first. See “Customizing the z/OS UNIX REXX environment” on page 13.
The search order for modules and execs that are invoked as functions or subroutines is controlled by the FUNCSOFL flag in the REXX parameter module. For a description of that flag, see z/OS TSO/E REXX Reference.

The following rule 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:
  \[ \text{ans} = \text{'myfunc'}(p1,p2) \]

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

Executable external functions or subroutines that are written in a language other than interpreted REXX and located in the z/OS UNIX file system are not supported.

---

**The TSO command environment**

The TSO command environment (ADDRESS TSO) can be used from a z/OS UNIX REXX environment, and is initialized with:

```
address tso [command]
```

where `command` can be any TSO/E command, CLIST, or REXX exec that can run in a batch TSO TMP.

Commands addressed to TSO are run in a TSO TMP that is running in a separate address space and process from your REXX program. This provides you with the capability to run TSO commands. It does not provide you with the capability to use TSO commands to affect your REXX environment, or to have REXX statements or other host command environments affect your TSO process.

The TSO process is started when the first TSO/E command is run, and persists until your REXX program terminates or you run the TSO LOGOFF command. You can use the `ps` shell command to observe this process as the program `bpwxrtd`. Unexpected termination of the TSO process causes the next TSO command to fail with return code 16. A subsequent command starts a new TSO process.

The `BPXWRFT=YES` environment variable can be set to cause file descriptors to be inherited by the TSO command processor. When it is set, file descriptors 10-99 are inherited.

---

**Command input**

Most native TSO commands, including commands that prompt for missing arguments, use TGET for input. This results in a command error, and the command usually terminates.

For commands that are able to read input, the source of the input is first any data that is currently on your stack, and then any data in your REXX program’s standard input stream. Regardless of whether the command processes input, all data on the stack is queued to the TSO command. The stack is empty after any TSO command has been run.

The standard input stream can also be queued as input to the TSO command. For example, if you have a file redirected as input and you run a TSO command before
processing that file, some or all of the file can be queued to the TSO command. If input is the terminal, queued input can be queued to the TSO command. This characteristic can be used to interact with some TSO commands.

You can disable command input by using the `rexxopt()` function with NOTSOIN specified.

**Command output**

By default, all command output is directed to your REXX process’s standard output stream. You can use the `outtrap()` function to trap command output in variables.

**Return codes**

The special REXX variable RC usually contains the return code from the TSO command. If the command abends or is not found, or if another error is detected, special return codes are set, and a descriptive message can be written to the standard error stream:

- `-3` usually means that the TSO command was not found.
- Other negative numbers are usually abend codes. These should be accompanied by a message containing an abend reason code.
- `16` usually means that a processing error was encountered.

**Examples**

To run the TSO/E TIME command:

```
address tso 'time'
```

To trap command output and print it:

```
call outtrap out.
   address tso 'listc'
   do i=1 to out.0
      say out.i
   end
```

To run a REXX exec in TSO/E:

```
address tso
   "alloc fi(sysexec) da('schoen.rexx') shr"
   "myexec"
```

This is a functional replacement for the `tsocmd` command:

```
/* rexx */
   address tso arg(1)
   return rc
```

In previous releases, the `tsocmd` command was available for download from the Tools and Toys z/OS UNIX Web page at [http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1toy.html](http://www.ibm.com/systems/z/os/zos/features/unix/bpxa1toy.html). As of V1R12, `tsocmd` is shipped as part of the base release. The base release version of `tsocmd` takes advantage of this REXX support.
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:

```rexx
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.

Like any other setuid or setgid program, a REXX program should not allow the user of the program to 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 z/OS UNIX 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 that is run from a z/OS shell or from 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 147 for more
  information.
- **writefile** to write an entire text file. See "writefile" on page 203 for more
  information.
- **read** to read bytes from any kind of file. See "read" on page 144 for more
  information.
- **write** to write bytes to any kind of file. See "write" on page 201 for more
  information.

Using EXECIO

EXECIO can be used to read or write zero or more lines in a file. EXECIO differs
from readfile and writefile in that it operates on open files. The file must be
opened in a mode consistent with the requested read or write operation. Read
operations will read from the current file cursor position. Write operations will
write from the current file cursor position, or, if the file is opened for append,
write operations will append to the end of the file.

Example: To read the open file associated with file descriptor 7 into the stem called
data.:

```mvs
address mvs 'execio * diskr 7 {stem data. fini'
```

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, z/OS UNIX supports
all the TSO/E REXX operands except OPEN, DISKRU, and `linenum`.

Note: When using EXECIO to read a file, the maximum allowable length of a line
in the file is 1024 characters, including the newline character. For
information on reading blocks of data, see "read" on page 144.
For the *ddname* operand, you can use the following pseudo-ddnames for processing, when the REXX program is run from a shell or from a program:
- File descriptors 0 to 7
- STDIN, STDOUT, STDERR

For information on EXECIO, see [z/OS TSO/E REXX Reference](#).

**Exit status from a REXX program**

When a REXX program is run from a z/OS shell or from 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 from the PARSE SOURCE instruction depend on where the REXX program is run: from a z/OS shell, from a program, from TSO/E, or from batch.

**Running from a z/OS shell or from a program**

When a REXX program runs in a z/OS 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 with the CALL instruction
3. The path name of the REXX program
4. The string PATH
5. The path name 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 whether the REXX program was run from a z/OS shell, use token 8 or 9.
**Example**

If `myexec` is invoked from the z/OS shells 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 that are described in [z/OS 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 might 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 2.

<table>
<thead>
<tr>
<th>C function</th>
<th>Equivalent REXX statement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sigsetempty()</code></td>
<td><code>sigsetempty: return copies(0,64)</code></td>
</tr>
<tr>
<td></td>
<td>• Parameters: none</td>
</tr>
<tr>
<td></td>
<td>• Returns: signal set</td>
</tr>
<tr>
<td><code>sigfillset()</code></td>
<td><code>sigfillset: return copies(1,64)</code></td>
</tr>
<tr>
<td></td>
<td>• Parameters: none</td>
</tr>
<tr>
<td></td>
<td>• Returns: signal set</td>
</tr>
</tbody>
</table>
Using immediate commands

Immediate commands are TSO/E REXX commands, provided with the TSO/E implementation of the language. Immediate commands change characteristics that control the execution of an exec or program.

In response to an interrupt signal, usually <Ctrl-C>, the REXX interrupt handler suspends execution of the REXX program and prompts for an immediate command. The command is specified by number. In the z/OS UNIX REXX environment, the following commands are supported:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Continue execution</td>
</tr>
<tr>
<td>2</td>
<td>Issue a Halt Interruption</td>
</tr>
<tr>
<td>3</td>
<td>Start trace</td>
</tr>
<tr>
<td>4</td>
<td>End trace</td>
</tr>
<tr>
<td>5</td>
<td>Halt type</td>
</tr>
<tr>
<td>6</td>
<td>Resume type</td>
</tr>
</tbody>
</table>

You can use the rexxopt() function to disable this capability or attach this signal handler to other signals. As with any signal handler, the kernel may defer delivery of the signal depending on what the program is executing at the time.

Note: REXX programs that are run as setuid or setgid programs cannot be interrupted to issue an immediate command.

Moving a REXX program from TSO/E to a z/OS 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 a z/OS shell environment. Some of the differences between the two environments that you need to consider are:

- You can use the spawn syscall command to run z/OS 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 a z/OS shell environment. If you use syscalls('ON') in a z/OS shell environment, it clears the __argv and __environment stems. For this reason, it is not recommended that you use syscalls('ON') in a z/OS shell environment. Using syscalls('ON') in a z/OS shell
environment also sets up the REXX predefined variables and blocks all signals.
On entry to a REXX program in a z/OS shell environment, the REXX predefined
variables are already set.
v In TSO/E, the syscalls('OFF') function ends the process, but the REXX program
continues to run. In the z/OS shells, the syscalls('OFF') function undubs the task
and the REXX program continues.
v PARSE SOURCE returns different tokens in TSO/E and in a z/OS shell
environment. A REXX program uses the tokens to determine how it was run.
v In TSO/E, the variables __argv.0 and __environment.0 are set to zero (0).
See “The TSO command environment” on page 5 for information about running
TSO/E commands from a REXX program.

Using argv and environment variables
Environment variables are text strings in the form VNAME=value, where VNAME
is the name of the variable and value is its value. 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
returns 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.
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.
Argument 1: "pgm1"
Argument 2: "arguments to"

12

z/OS V1R13.0 Using REXX and z/OS UNIX System Services

They are:

They are:’


Argument 3: "pgm1"
Using arg(), there are 2 arguments:
Argument 1: "arguments"
Argument 2: "to pgm2"

---

Customizing the z/OS UNIX REXX environment

When a REXX program is run from the z/OS shells or called from a program
using exec(), the z/OS UNIX 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 z/OS UNIX without affecting REXX programs
running in the z/OS environment. For detailed information on how to change the
default values for initializing an environment, see z/OS 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:

<table>
<thead>
<tr>
<th>Module name</th>
<th>SYS1.SAMPLIB member name</th>
<th>REXX environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRXPARMS</td>
<td>IRXREXX1</td>
<td>MVS</td>
</tr>
<tr>
<td>IRXTSPRM</td>
<td>IRXREXX2</td>
<td>TSO</td>
</tr>
<tr>
<td>IRXISPRM</td>
<td>IRXREXX3</td>
<td>ISPF</td>
</tr>
</tbody>
</table>

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

See z/OS 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 ISPF Shell or the su shell command can be used to switch to superuser authority.

Security

This document assumes that your operating system contains Resource Access Control Facility (RACF). You could use an equivalent security product updated to handle z/OS UNIX security.
Chapter 2. z/OS UNIX REXX programming services

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

Creating a z/OS UNIX REXX environment from an application

To create a z/OS UNIX REXX environment, fetch and call the module BPXWRBLD from a key 8 problem state program. z/OS 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:

```
--------+    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 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(t)` 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 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 z/OS UNIX 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 C/370™ program creates a REXX environment and runs a REXX program:

```c
#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 *bpwrblk; /* pointer to BPXWRBLD routine */
    char *penvb; /* addr of REXX environment */
    int i,j; /* temps */
    long rinit; /* return code */
    int **environlp; /* ptr to env length pointers */
    int *environl; /* ptr to env lengths */
    char *xwork[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 UNIX System Services. */
    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(environl[j])+1; /* set len word */
    }
    environlp[i]=NULL; /* null entry at end */
    environl[i]=0;

    /* load routines */
    irxjcl=(EXTF *)fetch("IRXJCL ");
}
```

Chapter 2. z/OS UNIX REXX programming services 17
bpwxrlbld=(EXTF *)&fetch("BPXWRBLD ");

/* build the REXX environment */
rcinit=bpwxrlbld(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 z/OS UNIX 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

In most cases, syscall commands invoke the z/OS UNIX 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 z/OS UNIX callable services, where applicable.

For complete information about the processing of a particular syscall command, read about the callable service it invokes, as described in z/OS UNIX System Services 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 not case-sensitive: 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 path name is case sensitive, and it is specified as a string. The syscall commands can take a relative or absolute path name as a parameter. The search for a relative path name begins in your working directory:

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

Portable path names 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 path name.

mode

The mode is a three- or four-digit number that corresponds to the access 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 about 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 not case-sensitive.

variable
The name of a REXX variable. The name is not case-sensitive.

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 nine 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:

<table>
<thead>
<tr>
<th>String</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any series of characters not containing a space. This example shows a path name with no space in it.</td>
<td>&quot;creat /u/wjs/file 700&quot;</td>
</tr>
<tr>
<td>Any series of characters delimited by ' and not containing '. This example shows a path name with a space in it.</td>
<td>&quot;creat 'u/wjs/my file' 700&quot;</td>
</tr>
<tr>
<td>Any series of characters delimited by &quot; and not containing &quot;. This example shows a path name with a space in it.</td>
<td>&quot;creat &quot;u/wjs/my file&quot; 700&quot;</td>
</tr>
<tr>
<td>A variable name enclosed in parentheses. Strings that contain both the single and double quote characters must be stored in a variable, and you must use the variable name.</td>
<td>file='&quot;u/wjs/my file' &quot;creat (file) 700&quot;</td>
</tr>
</tbody>
</table>

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

The predefined variables that are available to a REXX program in a z/OS shell environment 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 document 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 might be returned by the REXX language processor. A negative value means that the command did not finish successfully.

- **RETV AL**
  - A numeric return value from the callable service. This indicates the success or failure of the service. For most successful calls to services, RETVAL is set to zero; for unsuccessful calls, this value is −1.

  However, there are some services (such as getgrgid and getgrnam) that return zero instead of −1 when the service fails. In addition, some services return a positive RETVAL to indicate success. For
details about a specific service, see z/OS UNIX System Services
Programming: Assembler Callable Services Reference

**ERRNO**
A hexadecimal error number from the callable service. This variable is valid only if the return code (RC) is not negative and RETVAL is −1.

**ERRNOJR**
A hexadecimal reason code from the callable service. This variable is valid only if the return code (RC) is not negative and RETVAL is −1.

**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

**Syscall command descriptions**
access

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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_OK</td>
<td>Test for file existence</td>
</tr>
<tr>
<td>R_OK</td>
<td>Test for permission to read</td>
</tr>
<tr>
<td>W_OK</td>
<td>Test for permission to write</td>
</tr>
<tr>
<td>X_OK</td>
<td>Test for permission to execute</td>
</tr>
</tbody>
</table>

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
```
Function

`acldelete` deletes an access control list (ACL) associated with `pathname`.

Parameters

**pathname**

The pathname of the file or directory the ACL is associated with.

**acltype**

Indicates the type of ACL. This parameter can have the following values:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_TYPE_ACCESS (1)</td>
<td>An access ACL</td>
</tr>
<tr>
<td>ACL_TYPE_FILEDEFAULT (2)</td>
<td>A file default ACL</td>
</tr>
<tr>
<td>ACL_TYPE_DIRDEFAULT (3)</td>
<td>A directory default ACL</td>
</tr>
</tbody>
</table>

Usage notes

1. For regular files, the `acltype` must indicate it is an access ACL that is to be deleted.
2. For a directory, the `acltype` must indicate one of the three types of ACLs (access, file default, or directory default).

Example

To delete the access ACL from the `/tmp` directory, this example assumes the user has set the appropriate stem variable before the call:

```
"acldelete /tmp/ acl." acl_type_access
```

For a complete example that uses several of the ACL services to list ACLs, see "List the ACL entries for a file" on page 216.

For more information about access control lists, see Using access control lists (ACLs) in z/OS UNIX System Services Planning.
Function

acldeleteentry deletes a specific entry in the access control list (ACL) represented by variable.

Parameters

variable

The name of a REXX variable that contains a token to access an ACL.

stem

The name of a stem variable that contains an ACL entry. STEM.0 contains a count of the number of variables set in the stem. The following variables may be used to access the stem variables. The number in parentheses is the actual value of the variable:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_ENTRY_TYPE (1)</td>
<td>Indicates the type of ACL entry:</td>
</tr>
<tr>
<td>ACL_ENTRY_USER (1)</td>
<td>ACL_ENTRY_USER (1) (User ACL)</td>
</tr>
<tr>
<td>ACL_ENTRY_GROUP (2)</td>
<td>ACL_ENTRY_GROUP (2) (Group ACL)</td>
</tr>
<tr>
<td>ACL_ID (2)</td>
<td>The numeric id, uid or gid of the entry</td>
</tr>
<tr>
<td>ACL_READ (3)</td>
<td>Indicates read access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_WRITE (4)</td>
<td>Indicates write access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_EXECUTE (5)</td>
<td>Indicates execute or search access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_DELETE (6)</td>
<td>Indicates that the ACL entry is deleted (1 = yes, 0 = no)</td>
</tr>
</tbody>
</table>

Usage notes

1. The entry to delete is identified by the entry type and ID, and is contained in stem.
2. If the entry does not exist, the service will return retval=−1 and errno=enoent

Example

To delete the ID in an ACL, this example assumes the user has set the appropriate stem variable before the call:

```
"acldeleteentry tokenvar acl." acl_id
```

For a complete example that uses several of the ACL services to list ACLs, see "List the ACL entries for a file" on page 216.

For more information about access control lists, see Using access control lists [ACLs] in z/OS UNIX System Services Planning.
Function

`aclfree` releases resources associated with the access control list (ACL) represented by `variable` and obtained using the `aclinit` syscall command.

Parameters

`variable`

The name of a REXX variable that contains a token to access an ACL.

Example

```
"aclfree tokenvar"
```

For a complete example that uses several of the ACL services to list ACLs, see “List the ACL entries for a file” on page 216.

For more information about access control lists, see Using access control lists (ACLs) in z/OS UNIX System Services Planning.
Function
 aclget reads an access control list (ACL) of the specified type associated with the file identified by pathname. The ACL is associated with the specified variable, and is accessed and altered using the acldeleteentry, aclgetentry, and aclupdateentry services.

Before using aclget, the variable must be initialized by using aclinit.

Parameters

variable
   The name of a REXX variable that contains a token to access an ACL.

pathname
   The pathname of the file or directory the ACL is associated with.

acltype
   Indicates the type of ACL. This parameter can have the following values:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_TYPE_ACCESS (1)</td>
<td>An access ACL</td>
</tr>
<tr>
<td>ACL_TYPE_FILEDEFAULT (2)</td>
<td>A file default ACL</td>
</tr>
<tr>
<td>ACL_TYPE_DIRDEFAULT (3)</td>
<td>A directory default ACL</td>
</tr>
</tbody>
</table>

Example

```
*aclget tokenvar /lpp/payroll* acl_type_access
```

For a complete example that uses several of the ACL services to list ACLs, see "List the ACL entries for a file” on page 216.

For more information about access control lists, see Using access control lists (ACLs) in /OS UNIX System Services Planning.
Function

`aclgetentry` reads an access control list (ACL) entry from the ACL represented by `variable`.

Parameters

`variable`

The name of a REXX variable that contains a token to access an ACL.

`stem`

The name of a stem variable that contains an ACL entry. STEM.0 contains a count of the number of variables set in the stem. The following variables may be used to access the stem variables. The number in parentheses is the actual value of the variable:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_ENTRY_TYPE (1)</td>
<td>Indicates the type of ACL entry:</td>
</tr>
<tr>
<td>ACL_ENTRY_USER (1)</td>
<td>(User ACL)</td>
</tr>
<tr>
<td>ACL_ENTRY_GROUP (2)</td>
<td>(Group ACL)</td>
</tr>
<tr>
<td>ACL_ID (2)</td>
<td>The numeric id, uid or gid of the entry</td>
</tr>
<tr>
<td>ACL_READ (3)</td>
<td>Indicates read access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_WRITE (4)</td>
<td>Indicates write access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_EXECUTE (5)</td>
<td>Indicates execute or search access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_DELETE (6)</td>
<td>Indicates that the ACL entry is deleted (1 = yes, 0 = no)</td>
</tr>
</tbody>
</table>

`index`

Specifies the relative ACL entry to access. The first entry is 1.

Usage notes

1. An entry is identified by `index`, if `index` is specified. Otherwise, the entry is identified by the type and ID specified in `stem`.
2. If the entry does not exist, the service will return `retval=−1` and `errno=enoent`.

Example

To read an ACL entry, this example assumes the user has set the appropriate stem variable before the call:

```
"aclgetentry tokenvar acl."
```

For a complete example that uses several of the ACL services to list ACLs, see "List the ACL entries for a file" on page 216.

For more information about access control lists, see [Using access control lists](/OS UNIX System Services Planning)
Function
aclinit obtains resources necessary to process access control lists (ACLs) and associates those resources with variable.

Parameters
variable
The name of a REXX variable that contains a token to access an ACL.

Usage notes
1. The variable associated with the obtained resources must be passed to other services that operate on an ACL.
2. Any one variable can only represent one ACL at a time.
3. aclfree must be used to release the resources obtained by aclinit.

Example
"aclinit tokenvar"

For a complete example that uses several of the ACL services to list ACLs, see “List the ACL entries for a file” on page 216.

For more information about access control lists, see Using access control lists (ACLs) in z/OS UNIX System Services Planning
aclset

Function
aclset replaces the access control list (ACL) associated with pathname with the ACL represented by variable.

Parameters

variable
The name of a REXX variable that contains a token to access an ACL.

pathname
The pathname of the file or directory the ACL is associated with.

acltype
Indicates the type of ACL. This parameter can have the following values:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_TYPE_ACCESS (1)</td>
<td>An access ACL</td>
</tr>
<tr>
<td>ACL_TYPE_FILEDEFAULT (2)</td>
<td>A file default ACL</td>
</tr>
<tr>
<td>ACL_TYPE_DIRDEFAULT (3)</td>
<td>A directory default ACL</td>
</tr>
</tbody>
</table>

Example
This example replaces the directory default ACL:
"aclset tokenvar /u/dept58" acl_type_dirdefault

For a complete example that uses several of the ACL services to list ACLs, see "List the ACL entries for a file” on page 216.

For more information about access control lists, see Using access control lists (ACLs) in z/OS UNIX System Services Planning.
aclupdateentry

Function

`aclupdateentry` updates an existing access control list (ACL) entry or creates a new entry if the entry does not already exist in the ACL contained in `variable`.

Parameters

`variable`

The name of a REXX variable that contains a token to access an ACL.

`stem`

The name of a stem variable that contains an ACL entry. STEM.0 contains a count of the number of variables set in the stem. The following variables may be used to access the stem variables. The number in parentheses is the actual value of the variable:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL_ENTRY_TYPE</td>
<td>Indicates the type of ACL entry:</td>
</tr>
<tr>
<td></td>
<td>- ACL_ENTRY_USER (1) (User ACL)</td>
</tr>
<tr>
<td></td>
<td>- ACL_ENTRY_GROUP (2) (Group ACL)</td>
</tr>
<tr>
<td>ACL_ID</td>
<td>The numeric id, uid or gid of the entry</td>
</tr>
<tr>
<td>ACL_READ</td>
<td>Indicates read access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_WRITE</td>
<td>Indicates write access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_EXECUTE</td>
<td>Indicates execute or search access (1 = yes, 0 = no)</td>
</tr>
<tr>
<td>ACL_DELETE</td>
<td>Indicates that the ACL entry is deleted (1 = yes, 0 = no)</td>
</tr>
</tbody>
</table>

`index`

Specifies the relative ACL entry to access. The first entry is 1.

Usage notes

1. An entry is identified by either the entry type and ID contained in `stem` or by the relative entry number if `index` is specified. If `index` is 0 or greater than the current number of ACL entries, a new entry is created.
2. `aclupdateentry` can update a deleted ACL entry, mark an entry as deleted, mark the entry as not deleted, or add a new entry that is also marked as deleted by appropriate setting of `stem.acl_delete`. Also, duplicate entries can be added when using `index`. This may result in an unexpected ACL and should be avoided.
3. The read, write, execute, and deleted attributes are set based on the corresponding stem variables. A value of 1 indicates the attribute is to be set to 1. Any other value, including not setting the variable, results in the attribute being set to 0.
4. If the requested entry cannot be located, a new entry is created and the relative index for that entry is returned in `retval`. If an existing entry is updated, `retval` will contain 0. If `retval=−1` and `errno=enoent`, then a new entry could not be created because it would exceed the maximum number of entries (1024).
Example
To update an ACL entry, this example assumes the user has set the appropriate stem variable before the call.
"aclupdateentry tokenvar acl."

For a complete example that uses several of the ACL services to list ACLs, see "List the ACL entries for a file" on page 216.

For more information about access control lists, see Using access control lists (ACLs) in z/OS UNIX System Services Planning.
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 10 for additional information on using signals.

Example
To generate a SIGALRM after 10 seconds:

"alarm 10"
catclose

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 it is unsuccessful, `catclose` returns -1 and sets ERRNO to indicate the error.

Example

See the example for “catgets” on page 35.
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

Function
catopen opens a message catalog that has been built by the gencat utility. (For more information about gencat, see the gencat command description in z/OS UNIX System Services 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 it is unsuccessful, catopen returns a −1 and sets ERRNO to indicate the error.

Example
See the example for “catgets” on page 35.
Function
cert calls BPX1SEC to register or deregister a certificate with the calling user.

Parameters
cert_variable
    The name of the variable that contains the certificate.
cert_length
    The length of the certificate
flag
    Valid values for the flag are:
    • 0 — register
    • 1 — deregister

Usage notes
1. The intent of the cert service is to provide a way for the caller to associate/disassociate a certificate with the calling user. No new security environment is created and no authentication of the user is conducted.
2. The caller needs access to the RACDCERT facility class (as defined in the initACEE documentation) to register/deregister a certificate. No other authority above that is required to use cert.
3. The certificate is a data area that includes a 4-byte length field, header information for some certificate types, the actual certificate, and trailer information for some certificate types. The length passed in on the cert syscall is the whole length of that data area. It is up to the caller to build the appropriate structure.

Example
'cert newcert' length(newcert) 1
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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_CCSID</td>
<td>Coded character set ID; first 4 characters are the file tag.</td>
</tr>
<tr>
<td>ST_MODE</td>
<td>1 argument: permission bits as 3 octal digits.</td>
</tr>
<tr>
<td>ST_UID</td>
<td>2 arguments: UID and GID numbers.</td>
</tr>
<tr>
<td>ST_SIZE</td>
<td>1 argument: new file size.</td>
</tr>
<tr>
<td>ST_ATIME</td>
<td>1 argument for access time: new time or −1 for TOD.</td>
</tr>
<tr>
<td>ST_MTIME</td>
<td>1 argument for modification time: new time or −1 for TOD.</td>
</tr>
<tr>
<td>ST_CTIME</td>
<td>1 argument for change time: new time or −1 for TOD.</td>
</tr>
<tr>
<td>ST_SETUID</td>
<td>No arguments.</td>
</tr>
<tr>
<td>ST_SETGID</td>
<td>No arguments.</td>
</tr>
<tr>
<td>ST_AAUDIT</td>
<td>1 argument: new auditor audit value.</td>
</tr>
<tr>
<td>ST_UAUDIT</td>
<td>1 argument: new user audit value.</td>
</tr>
<tr>
<td>ST_STICKY</td>
<td>No arguments.</td>
</tr>
<tr>
<td>ST_GENVALUE</td>
<td>2 arguments: names of two variables. The first variable contains the general attribute mask and the second contains the general attribute value.</td>
</tr>
<tr>
<td>ST_RTIME</td>
<td>1 argument for reference time: new time or −1 for TOD.</td>
</tr>
</tbody>
</table>
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_FILEFMT</td>
<td>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:</td>
</tr>
<tr>
<td>S_FFBINARY</td>
<td>Binary data</td>
</tr>
<tr>
<td>S_FFCR</td>
<td>Text data delimited by a carriage return character</td>
</tr>
<tr>
<td>S_FFCRLF</td>
<td>Text data delimited by carriage return and line feed characters</td>
</tr>
<tr>
<td>S_FFCRNL</td>
<td>A text file with lines delimited by carriage-return and newline characters.</td>
</tr>
<tr>
<td>S_FFLF</td>
<td>Text data delimited by a line feed character</td>
</tr>
<tr>
<td>S_FFLFCR</td>
<td>Text data delimited by a line feed and carriage return characters</td>
</tr>
<tr>
<td>S_FFNA</td>
<td>Text data with the file format not specified</td>
</tr>
<tr>
<td>S_FFNL</td>
<td>Text data delimited by a newline character</td>
</tr>
<tr>
<td>S_FFRECORD</td>
<td>File data consisting of records with prefixes. The record prefix contains the length of the record that follows.</td>
</tr>
</tbody>
</table>

### 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 also updated.

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
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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUD_FREAD</td>
<td>Audit failed read requests</td>
</tr>
<tr>
<td>AUD_SREAD</td>
<td>Audit successful read requests</td>
</tr>
<tr>
<td>AUD_FWRITE</td>
<td>Audit failed write requests</td>
</tr>
<tr>
<td>AUD_SWRITE</td>
<td>Audit successful write requests</td>
</tr>
<tr>
<td>AUD_FEXEC</td>
<td>Audit failed execute or search requests</td>
</tr>
<tr>
<td>AUD_SEXEC</td>
<td>Audit successful execute or search requests</td>
</tr>
</tbody>
</table>

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
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"
```
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 also updated.

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*
```
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

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
**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"
Function
create 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 create 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 190) 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:
"create /u/lou/test.exec 700"
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)*`
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
There is no `exec` syscall command. Instead of using `exec`, see "spawn" on page 174.
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

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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_CCSID</td>
<td>Coded character set ID; first 4 characters are the file tag.</td>
</tr>
<tr>
<td>ST_MODE</td>
<td>1 argument: permission bits as 3 octal digits.</td>
</tr>
<tr>
<td>ST_UID</td>
<td>2 arguments: UID and GID numbers.</td>
</tr>
<tr>
<td>ST_SIZE</td>
<td>1 argument: new file size.</td>
</tr>
<tr>
<td>ST_ATIME</td>
<td>1 argument for access time: new time or −1 for TOD.</td>
</tr>
<tr>
<td>ST_MTIME</td>
<td>1 argument for modification time: new time or −1 for TOD.</td>
</tr>
<tr>
<td>ST_CTIME</td>
<td>1 argument for change time: new time or −1 for TOD.</td>
</tr>
<tr>
<td>ST_SETUID</td>
<td>No arguments.</td>
</tr>
<tr>
<td>ST_SETGID</td>
<td>No arguments.</td>
</tr>
<tr>
<td>ST_AAUDIT</td>
<td>1 argument: new auditor audit value.</td>
</tr>
<tr>
<td>ST_UAUDIT</td>
<td>1 argument: new user audit value.</td>
</tr>
<tr>
<td>ST_STICKY</td>
<td>No arguments.</td>
</tr>
<tr>
<td>ST_GENVALUE</td>
<td>2 arguments: names of two variables. The first variable contains the general attribute mask and the second contains the general attribute value.</td>
</tr>
<tr>
<td>ST_RTIME</td>
<td>1 argument for reference time: new time or −1 for TOD.</td>
</tr>
</tbody>
</table>
fchattr

Variable Description
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_FFCRNL A text file with lines delimited by carriage-return and newline 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
S_FFRECORD File data consisting of records with prefixes. The record prefix contains the length of the record that follows.

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 \textit{file\_size} to the original end of the file is removed. If the file was previously shorter than \textit{file\_size}, bytes between the old and new lengths are read as zeros. The file offset is not changed.
   • If \textit{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 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 \textit{st\_atime} or \textit{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 \textit{st\_atime} or \textit{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 \textit{st\_ctime} or \textit{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 \textit{st\_ctime} or \textit{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 also updated.

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.

\textbf{Example}

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

\begin{verbatim}
"fchattr" fd st_size 0 st_mode 600
\end{verbatim}
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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUD_FREAD</td>
<td>Audit failed read requests</td>
</tr>
<tr>
<td>AUD_SREAD</td>
<td>Audit successful read requests</td>
</tr>
<tr>
<td>AUD_FWRITE</td>
<td>Audit failed write requests</td>
</tr>
<tr>
<td>AUD_SWRITE</td>
<td>Audit successful write requests</td>
</tr>
<tr>
<td>AUD_FEXEC</td>
<td>Audit failed execute or search requests</td>
</tr>
<tr>
<td>AUD_SEXEC</td>
<td>Audit successful execute or search requests</td>
</tr>
</tbody>
</table>

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"
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 also updated.
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
```
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**

**Function**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>fd</em></td>
<td>The file descriptor (a number) for a file. This is the first file descriptor to be closed.</td>
</tr>
<tr>
<td><em>fd2</em></td>
<td>The file descriptor (a number) for a file, which must be greater than or equal to <em>fd</em>. If a −1 is specified for <em>fd2</em>, all file descriptors greater than or equal to <em>fd</em> are to be closed.</td>
</tr>
</tbody>
</table>

**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"
Function

*f_control_cvt* controls automatic file conversion and specifies the program and file CCSIDs (character code set identifiers) for an opened I/O stream.

Parameters

**fd** The file descriptor (a number) for the file. It must be a regular file, FIFO, or character special file.

**command**

The *command* can be one of these variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVT_SETCVTOFF</td>
<td>Turns off any conversion that may be in effect. A hex value of 0 for both <em>program-ccsid</em> and <em>file-ccsid</em> is recommended.</td>
</tr>
<tr>
<td>CVT_SETCVTON</td>
<td>Turns automatic conversion on for the stream and, optionally, sets the program CCSID or file CCSID, or both. A hex value of 0 for <em>program-ccsid</em> indicates using ThliCcsid (the current program CCSID) at the time of each read or write. ThliCcsid is initially 1047, but can be reset directly by the program, or indirectly by setting the appropriate run-time option or environment variable. A hex value of 0 for <em>file-ccsid</em> indicates not changing ThliCcsid.</td>
</tr>
<tr>
<td>CVT_SETAUTOCVTON</td>
<td>Conditionally turns automatic conversion on for the stream and, optionally, sets the program CCSID or file CCSID, or both. Conversion will be in effect for this stream only if the system or the local run-time environment has been enabled for conversion. A hex value of 0 for <em>program-ccsid</em> indicates using ThliCcsid at the time of each read or write. ThliCcsid is initially 1047, but can be changed by this variable, or by setting the appropriate run-time option or environment variable. A hex value of 0 for <em>file-ccsid</em> indicates not changing ThliCcsid.</td>
</tr>
<tr>
<td>CVT_QUERYCVT</td>
<td>Returns an indicator that lets you know whether automatic conversion is in effect or not, and also returns the program and file CCSIDs that are being used (if conversion is in effect). The variables <em>command</em>, <em>program-ccsid</em>, and <em>file-ccsid</em> are set when you have a successful return. <em>command</em> is set to either <em>cvt_setcvtoff</em> or <em>cvt_setcvton</em>. <em>program-ccsid</em> and <em>file-ccsid</em> are set to the CCSID values that would be in use if conversion were to occur.</td>
</tr>
</tbody>
</table>

**program-ccsid**

The name of a 2-byte hex variable that describes the CCSID for the running program.

**Note:** For EBCDIC (1047) the CCSID is '0417'X; for ASCII (819) it is '0333'X.
**file-ccsid**

The name of a 2-byte hex variable that describes the CCSID for the file opened with the file descriptor *fd*.

**Example**

This example turns automatic file conversion on for a stream opened with *fd* as the file descriptor:

```plaintext
pccsid = '0000'x
fccsid = '0000'x
cmd   = cvt_setautocvt
"f_control_cvt (fd) cmd pccsid fccsid"
```

This example queries the current conversion state:

```plaintext
pccsid = '0000'x
fccsid = '0000'x
cmd   = cvt_querycvt
"f_control_cvt (fd) cmd pccsid fccsid"
```
fcntl

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

- "f_closfd" on page 60
- "fdupfd" on page 64
- "f_dupfd2" on page 65
- "f_getfd" on page 66
- "f_getfl" on page 67
- "f_getlk" on page 68
- "f_setfd" on page 75
- "f_setfl" on page 76
- "f_setlk" on page 77
- "f_setlkw" on page 80
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
```
Function
f_dupfd2 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
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: \((\text{retval} / 2) = 1\)

Example
To get the file descriptor flags for file descriptor 0:
"*f_getfd 0*"
Function

_fgetfl_ 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]):

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

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:

```c
wtr = D2C(O_WRITE + O_TRUNC, 4))
If BITAND(D2C(retval, 4), wtr) = wtr 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
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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_LEN</td>
<td>The length of the byte range that is to be set, cleared, or queried.</td>
</tr>
<tr>
<td>L_PID</td>
<td>The process ID of the process holding the blocking lock, if one was found.</td>
</tr>
<tr>
<td>L_START</td>
<td>The starting offset byte of the lock that is to be set, cleared, or queried.</td>
</tr>
<tr>
<td>L_TYPE</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>F_RDLCK</td>
</tr>
</tbody>
</table>

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.
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>F_WRLCK</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L_TYPE</th>
<th>F_UNLCK</th>
<th>Unlock. This is used to unlock all locks held on the given range by the requesting process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_WHENCE</td>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>SEEK_CUR</td>
<td>The current offset</td>
</tr>
<tr>
<td></td>
<td>SEEK_END</td>
<td>The end of the file</td>
</tr>
<tr>
<td></td>
<td>SEEK_SET</td>
<td>The specified offset</td>
</tr>
</tbody>
</table>

### Example

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

```c
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 */
```
There is no `fork` syscall command. Instead of using `fork`, see "spawn" on page 174.
**forkexecm**

**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.
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.
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'*
```
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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC_ACL</td>
<td>Test if access control lists (ACLs) are supported.</td>
</tr>
<tr>
<td>PC_ACL_MAX</td>
<td>Maximum number of entries allowed in an ACL.</td>
</tr>
<tr>
<td>PC_LINK_MAX</td>
<td>Maximum value of a file's link count.</td>
</tr>
<tr>
<td>PC_MAX_CANON</td>
<td>Maximum number of bytes in a terminal canonical input line.</td>
</tr>
<tr>
<td>PC_MAX_INPUT</td>
<td>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.</td>
</tr>
<tr>
<td>PC_NAME_MAX</td>
<td>Maximum number of bytes in a filename (not a string length; count excludes a terminating null).</td>
</tr>
<tr>
<td>PC_PATH_MAX</td>
<td>Maximum number of bytes in a pathname (not a string length; count excludes a terminating null).</td>
</tr>
<tr>
<td>PC_PIPE_BUF</td>
<td>Maximum number of bytes that can be written atomically when writing to a pipe.</td>
</tr>
<tr>
<td>PC_POSIX_CHOWN_RESTRICTED</td>
<td><strong>Change ownership</strong> 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.</td>
</tr>
<tr>
<td>PC_POSIX_NO_TRUNC</td>
<td>Pathname components longer than 255 bytes generate an error.</td>
</tr>
<tr>
<td>PC_POSIX_VDISABLE</td>
<td>Terminal attributes maintained by the system can be disabled using this character value.</td>
</tr>
</tbody>
</table>
Usage notes
1. If `name` refers to MAX_CANON, MAX_INPUT, or _POSIX_VDISABLE, 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)"
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"
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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_APPEND</td>
<td>Set offset to EOF on write.</td>
</tr>
<tr>
<td>O_NONBLOCK</td>
<td>Do not block an open, a read, or a write on the file (do not wait for terminal input).</td>
</tr>
<tr>
<td>O_SYNC</td>
<td>Force synchronous updates.</td>
</tr>
</tbody>
</table>

Example

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

"f_setfl 1" o_append
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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_LEN</td>
<td>The length of the byte range that is to be set, cleared, or queried.</td>
</tr>
<tr>
<td>L_PID</td>
<td>The process ID of the process holding the blocking lock, if one was found.</td>
</tr>
<tr>
<td>L_START</td>
<td>The starting offset byte of the lock that is to be set, cleared, or queried.</td>
</tr>
<tr>
<td>L_TYPE</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>F_RDLCK</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
Variable Description

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
F_UNLCK
Unlock. This is used to unlock all locks held on the given range by the requesting process.

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."
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.

Variable Description

- **L_LEN**
  - The length of the byte range that is 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 that is 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.
**Variable Description**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_WRLCK</td>
<td>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.</td>
</tr>
<tr>
<td>L_TYPE</td>
<td>F_UNLCK</td>
</tr>
<tr>
<td>L_WHENCE</td>
<td>Unlock. This is used to unlock all locks held on the given range by the requesting process.</td>
</tr>
<tr>
<td></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>SEEK_CUR</td>
</tr>
<tr>
<td></td>
<td>SEEK_END</td>
</tr>
<tr>
<td></td>
<td>SEEK_SET</td>
</tr>
</tbody>
</table>

**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 77 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."
```
Function

The function `f_settag` can be used to directly control basic file tagging for an opened file. The setting of the file tag may be immediate or deferred until the first write, depending upon the input parameters.

**fd**  The file descriptor (a number) for the file. It must be a regular file, FIFO, or character special file.


Example

The following example sets a file tag for an opened file to ASCII ISO8859-1, with the text conversion flag on as well:

```c
  tag = '03338000'x
  "f_settag (fd) tag"
```
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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_Audit</td>
<td>Auditor audit information</td>
</tr>
<tr>
<td>ST_ACCESSACL</td>
<td>1 if there is an access ACL (access control list)</td>
</tr>
<tr>
<td>ST_ATIME</td>
<td>Time of last access</td>
</tr>
<tr>
<td>ST_AUDITID</td>
<td>RACF File ID for auditing</td>
</tr>
<tr>
<td>ST_BLKSIZE</td>
<td>File block size</td>
</tr>
<tr>
<td>ST_BLOCKS</td>
<td>Blocks allocated</td>
</tr>
<tr>
<td>ST_CCSID</td>
<td>Coded character set ID; first 4 characters are the file tag</td>
</tr>
<tr>
<td>ST_CRTIME</td>
<td>File creation time</td>
</tr>
<tr>
<td>ST_CTIME</td>
<td>Time of last file status change</td>
</tr>
<tr>
<td>ST_DEV</td>
<td>Device ID of the file</td>
</tr>
<tr>
<td>ST_DMODELACL</td>
<td>1 if there is a directory model access control list (ACL)</td>
</tr>
<tr>
<td>ST_EXTLINK</td>
<td>External symbolic link flag, set to 0 or 1</td>
</tr>
<tr>
<td>ST_FID</td>
<td>File identifier</td>
</tr>
<tr>
<td>ST_FILEFMT</td>
<td>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:</td>
</tr>
</tbody>
</table>

- **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_FFCRNL** A text file with lines delimited by carriage-return and newline 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_FFNIL** Text data delimited by a newline character
- **S_FFRECORD** File data consisting of records with prefixes. The record prefix contains the length of the record that follows.
### fstat

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_FMODEACL</td>
<td>1 if there is a file model ACL</td>
</tr>
<tr>
<td>ST_GENVALUE</td>
<td>General attribute values</td>
</tr>
<tr>
<td>ST_GID</td>
<td>Group ID of the group of the file</td>
</tr>
<tr>
<td>ST_INO</td>
<td>File serial number</td>
</tr>
<tr>
<td>ST_MAJOR</td>
<td>Major number for a character special file</td>
</tr>
<tr>
<td>ST_MINOR</td>
<td>Minor number for a character special file</td>
</tr>
<tr>
<td>ST_MODE</td>
<td>File mode, permission bits only</td>
</tr>
<tr>
<td>ST_MTIME</td>
<td>Time of last data modification</td>
</tr>
<tr>
<td>ST_NLINK</td>
<td>Number of links</td>
</tr>
<tr>
<td>ST_RTIME</td>
<td>File backup time stamp (reference time)</td>
</tr>
<tr>
<td>ST_SETGID</td>
<td>Set Group ID on execution flag, set to 0 or 1</td>
</tr>
<tr>
<td>ST_SETUID</td>
<td>Set User ID on execution flag, set to 0 or 1</td>
</tr>
<tr>
<td>ST_SIZE</td>
<td>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).</td>
</tr>
<tr>
<td>ST_SECLABEL</td>
<td>Security Label</td>
</tr>
<tr>
<td>ST_STICKY</td>
<td>Sticky bit flag (keep loaded executable in storage), set to 0 or 1</td>
</tr>
<tr>
<td>ST_TYPE</td>
<td>Numeric value that represents the file type for this file. You can use a numeric value (see <a href="#">Appendix A</a>) or any of the predefined variables that begin with S_ to determine the file type:</td>
</tr>
<tr>
<td>S_ISCHR</td>
<td>Character special file</td>
</tr>
<tr>
<td>S_ISDIR</td>
<td>Directory</td>
</tr>
<tr>
<td>S_ISFIFO</td>
<td>FIFO special file</td>
</tr>
<tr>
<td>S_ISREG</td>
<td>Regular file</td>
</tr>
<tr>
<td>S_ISSYM</td>
<td>Symbolic link</td>
</tr>
<tr>
<td>ST_UAUDIT</td>
<td>Area for user audit information</td>
</tr>
<tr>
<td>ST_UID</td>
<td>User ID of the owner of the file</td>
</tr>
</tbody>
</table>

**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."
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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STFS_AVAIL</td>
<td>Space available to unprivileged users in block-size units.</td>
</tr>
<tr>
<td>STFS_BFREE</td>
<td>Total number of free blocks.</td>
</tr>
<tr>
<td>STFS_BLOCKSIZE</td>
<td>Block size.</td>
</tr>
<tr>
<td>STFS_FAVAIL</td>
<td>Number of free file nodes available to unprivileged users.</td>
</tr>
<tr>
<td>STFS_FFREE</td>
<td>Total number of free file nodes.</td>
</tr>
<tr>
<td>STFS_FILES</td>
<td>Total number of file nodes in the file system.</td>
</tr>
<tr>
<td>STFS_FRSIZE</td>
<td>Fundamental file system block size.</td>
</tr>
<tr>
<td>STFS_FSID</td>
<td>File system ID set by the logical file system.</td>
</tr>
<tr>
<td>STFS_INUSE</td>
<td>Allocated space in block-size units.</td>
</tr>
<tr>
<td>STFS_INVARSEC</td>
<td>Number of seconds the file system will remain unchanged.</td>
</tr>
<tr>
<td>STFS_NAMEMAX</td>
<td>Maximum length of file name.</td>
</tr>
<tr>
<td>STFS_NOSEC</td>
<td>Mount data set with no security bit.</td>
</tr>
<tr>
<td>STFS_NOSUID</td>
<td>SETUID and SETGID are not supported.</td>
</tr>
<tr>
<td>STFS_RDONLY</td>
<td>File system is read-only.</td>
</tr>
<tr>
<td>STFS_TOTAL</td>
<td>Total space in block-size units.</td>
</tr>
</tbody>
</table>

Example

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

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

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

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.
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.
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, "REXX predefined variables," on page 291) 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_GID</td>
<td>The group ID</td>
</tr>
<tr>
<td>GR_MEM</td>
<td>The stem index of the first member name returned</td>
</tr>
<tr>
<td>GR_MEMBERS</td>
<td>The number of members returned</td>
</tr>
<tr>
<td>GR_NAME</td>
<td>The name of the group</td>
</tr>
</tbody>
</table>

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
```
getgrgid

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, “REXX predefined variables,” on page 291) 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_GID</td>
<td>The group ID</td>
</tr>
<tr>
<td>GR_MEM</td>
<td>The stem index of the first member name returned</td>
</tr>
<tr>
<td>GR_MEMBERS</td>
<td>The number of members returned</td>
</tr>
<tr>
<td>GR_NAME</td>
<td>The name of the group</td>
</tr>
</tbody>
</table>

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:

```rexx
"getgrgid (gid) gr."
say 'Users connected to group number' gid ':'
do i=gr_mem to gr.0
   say gr.i
end
```
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, "REXX predefined variables," on page 291) 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR_GID</td>
<td>The group ID</td>
</tr>
<tr>
<td>GR_MEM</td>
<td>The stem index of the first member name returned</td>
</tr>
<tr>
<td>GR_MEMBERS</td>
<td>The number of members returned</td>
</tr>
<tr>
<td>GR_NAME</td>
<td>The name of the group</td>
</tr>
</tbody>
</table>

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 z/OS UNIX System Services.

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

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."
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

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**

---

**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 the z/OS UNIX 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"
```
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 formatted information in a stem variable.

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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNTE_AGGNAME</td>
<td>The name of the zFS aggregate data set.</td>
</tr>
<tr>
<td>MNTE_BYTESREADHW</td>
<td>The number of bytes read (high-word value).</td>
</tr>
<tr>
<td>MNTE_BYTESREADLW</td>
<td>The number of bytes read (low-word value).</td>
</tr>
<tr>
<td>MNTE_BYTESWRITTENHW</td>
<td>The number of bytes written (high-word value).</td>
</tr>
<tr>
<td>MNTE_BYTESWRITTENLW</td>
<td>The number of bytes written (low-word value).</td>
</tr>
<tr>
<td>MNTE_DD</td>
<td>The ddname specified on the mount.</td>
</tr>
<tr>
<td>MNTE_DEV</td>
<td>The device ID of the file system.</td>
</tr>
<tr>
<td>MNTE_DIRIBC</td>
<td>The number of directory I/O blocks.</td>
</tr>
<tr>
<td>MNTE.FILETAG</td>
<td>The file tag.</td>
</tr>
<tr>
<td>MNTE_FROMSYS</td>
<td>The file systems are to be moved from here.</td>
</tr>
<tr>
<td>MNTE_FSNAME</td>
<td>The file system name that was specified on the mount.</td>
</tr>
<tr>
<td>MNTE_FSTYPE</td>
<td>The file system type; for example, HFS.</td>
</tr>
</tbody>
</table>

This statement applies only to HFS and zFS. The file system name is the name of the MVS data set that contains the HFS or zFS file system.
**Variable** | **Description**
---|---
MNTE_MODE | The file system type mount method. MNTE_MODE contains a decimal number that corresponds to the value as defined in the field mntentfsmode in the BPXYMNTE mapping macro. The REXX predefined variables for this field, as well as other fields on this interface, are a subset of what might be returned by the getmntent service.

You can specify a numeric value (see Appendix A) or one of the following predefined variables used to derive the appropriate numeric value. For example, there is no predefined rexx variable corresponding to the bit position defined by mntentfssynchonly in the mntentfsmode.

- **MNT_MODE_RDWR** File system mounted read-write.
- **MNT_MODE_RDONLY** File system mounted read-only.
- **MNT_MODE_AUNMOUNT** The file system can be unmounted if the system's owner crashes.
- **MNT_MODE_CLIENT** The file system is a client.
- **MNT_MODE_EXPORT** The file system exported by DFS.
- **MNT_MODE_NOAUTOMOVE** Automove is not allowed.
- **MNT_MODE_NOSEC** No security checks are enforced.
- **MNT_MODE_NOSETID** SetUID is not permitted for files in this filesystem.
- **MNT_MODE_SECACL** ACLs are supported by the security product.

MNTE_PARDEV | The ST_DEV of the parent file system.
MNTE_PARM | The parameter specified with `mount()`.
MNTE_PATH | The mountpoint pathname.
MNTE_PFSSTATUSNORMAL | Normal status string returned by the physical file system.
MNTE_PFSSTATUSEXCP | Exception status string returned by the physical file system.
MNTE_ROSECLABEL | Default security label for objects in a read-only file system.
MNTE_QJOBNAME | The job name of the quiesce requestor.
MNTE_QPID | The PID of the quiesce requestor.
MNTE_QSYSNAME | The name of the quiesce system name.
MNTE_READCT | The number of reads from filesys.
MNTE_READIBC | The number of read 1/O blocks.
MNTE_RFLAGS | The request flags.
MNTE_ROOTINO | The inode of the mountpoint.
MNTE_ROSECLABEL | The read-only SECLABEL.
**getmntent**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNTE_STATUS</td>
<td>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:</td>
</tr>
<tr>
<td>MNT_ASYNCHMOUNT</td>
<td>Asynchronous mount in progress for this file system.</td>
</tr>
<tr>
<td>MNT_FILEACTIVE</td>
<td>File system is active.</td>
</tr>
<tr>
<td>MNT_FILEDEAD</td>
<td>File system is dead.</td>
</tr>
<tr>
<td>MNT_FILEDRAIN</td>
<td>File system is being unmounted with the drain option.</td>
</tr>
<tr>
<td>MNT_FILEFORCE</td>
<td>File system is being unmounted with the force option.</td>
</tr>
<tr>
<td>MNT_FILEIMMED</td>
<td>File system is being unmounted with the immediate option.</td>
</tr>
<tr>
<td>MNT_FILERESET</td>
<td>File system is being unmounted with the normal option.</td>
</tr>
<tr>
<td>MNT_FILENORM</td>
<td>File system is quiesced.</td>
</tr>
<tr>
<td>MNT_MOUNTINPROGRESS</td>
<td>Mount in progress for this file system</td>
</tr>
<tr>
<td>MNT_QUIESCED</td>
<td>File system is quiesced.</td>
</tr>
<tr>
<td>MNTE_STATUS2</td>
<td>The status of the file system.</td>
</tr>
<tr>
<td>MNTE_SUCCESS</td>
<td>Successful moves.</td>
</tr>
<tr>
<td>MNTE_SYSLIST</td>
<td>A list of system names.</td>
</tr>
<tr>
<td>MNTE_SYSNAME</td>
<td>The name of the owning system.</td>
</tr>
<tr>
<td>MNTE_TYPE</td>
<td>The file system type.</td>
</tr>
<tr>
<td>MNTE_UID</td>
<td>The effective UID of the nonprivileged user who mounted this file system. MNTE_UID is always 0 for the file system mounted by the privileged user.</td>
</tr>
<tr>
<td>MNTE_WRITEECT</td>
<td>The number of writes done.</td>
</tr>
<tr>
<td>MNTE_WRITEIBC</td>
<td>The number of write I/O blocks.</td>
</tr>
</tbody>
</table>

**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.

**pathflag**

An optional parameter, this specifies the returned path names.

- 0 Returns the current path name.
- 1 Returns up to 64 bytes of the path name at the time of the mount.

**Example**

Show all mounted file systems:
getmntent

    address syscall 'getmntent mounts.'
    do i=1 to mounts.0
        say mounts.mnte_fsname.i
    end
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.
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.
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.
Function

getpsent invokes the \texttt{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

\texttt{stem}

Upon return, \texttt{stem.0} contains the number of processes for which information is returned. \texttt{stem.1} through \texttt{stem.n} (where \texttt{n} is the number of entries returned) each contain process information for the \texttt{n}th process.

You can use the predefined variables that begin with PS_ or their equivalent numeric values (see Appendix A, “REXX predefined variables,” on page 291) to access the information. For example, \texttt{stem.1.ps\_pid} is the process ID for the first process returned.

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS_STATE</td>
<td>Process state This value can be expressed as one of the following predefined variables or as an alphabetic value (see Appendix A, “REXX predefined variables,” on page 291):</td>
</tr>
<tr>
<td>PS_CHILD</td>
<td>Waiting for a child process</td>
</tr>
<tr>
<td>PS_FORK</td>
<td>fork() a new process</td>
</tr>
<tr>
<td>PS_FREEZE</td>
<td>QUIESCEFREEZE</td>
</tr>
<tr>
<td>PS_MSGRCV</td>
<td>IPC MSGRCV WAIT</td>
</tr>
<tr>
<td>PS_MSGSND</td>
<td>IPC MSGSND WAIT</td>
</tr>
<tr>
<td>PS_PAUSE</td>
<td>MVSPAUSE</td>
</tr>
<tr>
<td>PS_QUIESCE</td>
<td>Quiesce termination wait</td>
</tr>
<tr>
<td>PS_RUN</td>
<td>Running, not in kernel wait</td>
</tr>
<tr>
<td>PS_SEMWT</td>
<td>IPC SEMOP WAIT</td>
</tr>
<tr>
<td>PS_SLEEP</td>
<td>sleep() issued</td>
</tr>
<tr>
<td>PS_WAITC</td>
<td>Communication kernel wait</td>
</tr>
<tr>
<td>PS_WAITF</td>
<td>File system kernel wait</td>
</tr>
<tr>
<td>PS_WAITO</td>
<td>Other kernel wait</td>
</tr>
<tr>
<td>PS_ZOMBIE</td>
<td>Process cancelled</td>
</tr>
<tr>
<td>PS_ZOMBIE2</td>
<td>Process terminated yet still the session or process group leader</td>
</tr>
<tr>
<td>PS_SUID</td>
<td>Saved set user ID</td>
</tr>
<tr>
<td>PS_SYSTIME</td>
<td>System CPU time, a value of the type clock_t, which needs to be divided by sysconf(_SC_CLK_TCK) to convert it to seconds. For z/OS UNIX, this value is expressed in hundredths of a second.</td>
</tr>
<tr>
<td>PS_USERTIME</td>
<td>User CPU time, a value of the type clock_t, which needs to be divided by sysconf(_SC_CLK_TCK) to convert it to seconds. For z/OS UNIX, this value is expressed in hundredths of a second.</td>
</tr>
<tr>
<td>PS_VNODECOUNT</td>
<td>Current number of vnode tokens</td>
</tr>
</tbody>
</table>

### 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.
4. PS_SYSTIME reports the system CPU time consumed for the address space that the process is running in. When only one process is running in the address space, this time represents the accumulated system CPU time for that process. However, when more than one process is running in an address space, the information that is returned is actually the accumulated system CPU time consumed by all of the work running in the address space.

### Example

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

```rexx
/* rexx */
address syscall
say right('PID',12) left('TTY',10) ' TIME' 'COMMAND'
ps.0=0
```

---

Chapter 3. The syscall commands 107
'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):'
  parse value reverse(ps.i.ps_contty),
     with tty '/'
  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

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, “REXX predefined variables,” on page 291) or the predefined variables beginning with PW_ to access the values:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PW_DIR</td>
<td>The initial working directory</td>
</tr>
<tr>
<td>PW_GID</td>
<td>The group ID</td>
</tr>
<tr>
<td>PW_NAME</td>
<td>The TSO/E user ID</td>
</tr>
<tr>
<td>PW_SHELL</td>
<td>The name of the initial user program</td>
</tr>
<tr>
<td>PW_UID</td>
<td>The user ID (UID) as defined to RACF</td>
</tr>
</tbody>
</table>

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
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, “REXX predefined variables,” on page 291) 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PW_DIR</td>
<td>The initial working directory</td>
</tr>
<tr>
<td>PW_GID</td>
<td>The group ID</td>
</tr>
<tr>
<td>PW_NAME</td>
<td>The TSO/E user ID</td>
</tr>
<tr>
<td>PW_SHELL</td>
<td>The name of the initial user program.</td>
</tr>
<tr>
<td>PW_UID</td>
<td>The user ID (UID) as defined to RACF.</td>
</tr>
</tbody>
</table>

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. 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."
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, “REXX predefined variables,” on page 291) or the predefined variables beginning with PW_ to access the values:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PW_DIR</td>
<td>The initial working directory</td>
</tr>
<tr>
<td>PW_GID</td>
<td>The group ID</td>
</tr>
<tr>
<td>PW_NAME</td>
<td>The TSO/E user ID</td>
</tr>
<tr>
<td>PW_SHELL</td>
<td>The name of the initial user program.</td>
</tr>
<tr>
<td>PW_UID</td>
<td>The user ID (UID) as defined to RACF.</td>
</tr>
</tbody>
</table>

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

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, “REXX predefined variables,” on page 291 for the numeric values.)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLIMIT_AS</td>
<td>Maximum address space size for a process</td>
</tr>
<tr>
<td>RLIMIT_CORE</td>
<td>Maximum size (in bytes) of a core dump created by a process</td>
</tr>
<tr>
<td>RLIMIT_CPU</td>
<td>Maximum amount of CPU time (in seconds) used by a process</td>
</tr>
<tr>
<td>RLIMITFSIZE</td>
<td>Maximum files size (in bytes) created by a process</td>
</tr>
<tr>
<td>RLIMTNOFILE</td>
<td>Maximum number of open file descriptors for a process</td>
</tr>
</tbody>
</table>

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
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.
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, “REXX predefined variables,” on page 291) 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`:

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

Example

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

```
gmtime" st.st_ctime "tm."
```
**ioctl**

```plaintext
 ioctl(fd, command, buffer, length)
```

**Function**

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

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

**Parameters**

- **fd** A file descriptor for an open file.
  
  **Restriction:** 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.
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

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:

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

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

**option**

The name of a variable that contains exactly four bytes mapped by the BPXYPPSD macro.

**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 10 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
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)
```
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"
```
Iseek

Function
Iseek 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEEK_CUR</td>
<td>Set the file offset to current offset plus the specified offset.</td>
</tr>
<tr>
<td>SEEK_END</td>
<td>Set the file offset to EOF plus the specified offset.</td>
</tr>
<tr>
<td>SEEK_SET</td>
<td>Set the file offset to the specified offset.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Position Specified</th>
<th>Whence</th>
<th>New File Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>SEEK_CUR</td>
<td>1080</td>
</tr>
<tr>
<td>1200</td>
<td>SEEK_SET</td>
<td>1200</td>
</tr>
<tr>
<td>−80</td>
<td>SEEK_END</td>
<td>1920</td>
</tr>
<tr>
<td>132</td>
<td>SEEK_END</td>
<td>2132</td>
</tr>
</tbody>
</table>

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):
Iseek

"Iseek" fd 0 seek_set
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 83 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

Format

```
mkdir pathname mode
```

Purpose

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

Parameters

`pathname`

A path name 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

1. The file permission bits specified in `mode` are modified by the file creation mask of the calling process (see “umask” on page 190), 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.

Examples

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

```
"mkdir (pathname)" mode
```
**mkfifo**

**Format**

```
  mkfifo pathname mode
```

**Purpose**

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

**Parameters**

- **pathname**
  A path name 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**

1. The file permission bits specified in *mode* are modified by the process's file creation mask (see “umask” on page 190), 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.

**Examples**

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

Format

```plaintext
-mknod pathname mode major minor
```

Purpose

**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 path name 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 Creating special files in z/OS UNIX System Services 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 Creating special files in z/OS UNIX System Services Planning.

Usage

1. The file permission bits specified in **mode** are modified by the process's file creation mask (see "umask" on page 190), 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.

Examples

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

```
I$earchByExample"mknod /dev/null 777 4 0"
```
mount

Format

```
mount path-name name type flags
```

```
mount stem
```

Purpose

*mount* invokes the mount callable service to mount a file system, making the files in it available for use.

**Requirement:** The caller must have mount authorities to mount a file system. See the section on [mount authority](https://z/OS UNIX System Services Planning) in *z/OS UNIX System Services Planning*.

When used in a sysplex, *mount* can also be used to change some of the mounted file system attributes, including the system that owns that mount.

Parameters

- **pathname**
  The path name 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_NOSECURITY**
    Mount with no security.
  - **MTM_NOSUID**
    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.

**stem**

The name of a stem variable which contains the mount variables. To set the mount variables, you can use a numeric value (see Appendix A) or the predefined variables beginning with MNTE_ used to derive the numeric value. Unused stem variables should be set to the null string.

The following variables are used for mount requests:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNTE_FILETAG</td>
<td>4-byte file tag, the contents of which are mapped by the ST_FILETAG structure in the BPXSTAT mapping macro. BPXSTAT is described in BPXSTAT — Map the Response Structure for stat in z/OS UNIX System Services Programming.</td>
</tr>
<tr>
<td>MNTE_FSNAME</td>
<td>The name of the HFS data set containing the file system.</td>
</tr>
<tr>
<td>MNTE_FSTYPE</td>
<td>The file system type; for example, HFS.</td>
</tr>
<tr>
<td>MNTE_MODE</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td><strong>MNT_MODE_RDWR</strong> File system mounted read-write.</td>
</tr>
<tr>
<td></td>
<td><strong>MNT_MODE_RDONLY</strong> File system mounted read-only.</td>
</tr>
<tr>
<td></td>
<td><strong>MNT_MODE_AUNMOUNT</strong> The file system can be unmounted if the system's owner crashes.</td>
</tr>
<tr>
<td></td>
<td><strong>MNT_MODE_NOAUTOMOVE</strong> Automove is not allowed.</td>
</tr>
<tr>
<td></td>
<td><strong>MNT_MODE_NOSEC</strong> No security checks are enforced.</td>
</tr>
<tr>
<td></td>
<td><strong>MNT_MODE_NOSETID</strong> SetUID is not permitted for files in this filesystem.</td>
</tr>
<tr>
<td>MNTE_PARM</td>
<td>The parameter specified with <code>mount()</code>.</td>
</tr>
<tr>
<td>MNTE_PATH</td>
<td>The mountpoint path name.</td>
</tr>
<tr>
<td>MNTE_SYSLIST</td>
<td>A list of system names. For more information about MNTE_SYSLIST, see BPXYMNT — Map response and element structure of w_getmntent in z/OS UNIX System Services Programming: Assembler Callable Services Reference.</td>
</tr>
<tr>
<td>MNTE_SYSNAME</td>
<td>The name of the file system to be mounted on.</td>
</tr>
</tbody>
</table>

The following variables are used for changing mount attributes:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNTE_FSNAME</td>
<td>The name of the HFS data set containing the file system.</td>
</tr>
</tbody>
</table>
Variable Description

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**
  - File system mounted read-write.
- **MNT_MODE_RDONLY**
  - File system mounted read-only.
- **MNT_MODE_AUNMOUNT**
  - The file system can be unmounted if the system's owner crashes.
- **MNT_MODE_NOAUTOMOVE**
  - Automove is not allowed.

MNTE_RFLAGS Request flags (set to 3 to change the automove attribute, 1 to change mount owner).

MNTE_SYSNAME The name of the system to be mounted on.

MNTE_SYSLIST A list of system names. For more information about MNTE_SYSLIST, see BPXYMNT — Map response and element structure of w_getmntent in z/OS UNIX System Services Programming: Assembler Callable Services Reference.

Usage

1. The mount service effectively creates a virtual file system. After a file system is mounted, references to the path name that is mounted refer to the root directory on the mounted file system.

2. A file system can be mounted at only one point.

3. 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.

4. You can use a wildcard (*) as the last item (or only item) of a system list (MNTE_SYSLIST). A wildcard is allowed only with an INCLUDE list, not with an EXCLUDE list.

Examples

1. 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

2. To mount an HFS read/write with this system as the owner:

   ```
   m." /* set all of the stem variables to the null string */
   m.mnte_mode=mnt_mode_rdwr
   m.mnte_fsname='OMVS.HFS.U.WJS'
   m.mnte_fstype='HFS'
   m.mnte_path='/u/wjs'
   address syscall 'mount m.'
   ```

3. To mount an HFS file system read/write with system SYS2 as the owner and with NOAUTOMOVE:

   ```
   m.="
   m.mnte_mode=mnt_mode_rdwr+mnt_mode_noautomove
   m.mnte_fsname='OMVS.HFS.U.WJS'
   m.mnte_fstype='HFS'
   m.mnte_path='/u/wjs'
   m.mnte_sysname='SYS2'
   address syscall 'mount m.'
   ```

4. **mnt.****

   ```
   mnt.mnte_fsname = 'FS1.HFS'
mnt.mnte_path = '/fs1'
   ```
mnt.mnte_mode = mnt_mode_rdwr
mnt.mnte_fstype = 'HFS'

/* build an INCLUDE system list */
syslist = 'SY1 SY2 SY3'
num_systems = d2c(words(syslist),2)
type_syslist = '0000'x /* or '0001'x for exclude list */
mnt.mnte_syslist = num_systems || type_syslist

do i = 1 to words(syslist)
mnt.mnte_syslist = mnt.mnte_syslist||left(word(syslist,i),8)
end

address syscall 'mount mnt.'
say 'mount RRR = ' retval errno errnoj
**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:

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

**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 190), 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
```
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 142) 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"
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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PC_ACL</code></td>
<td>Test if access control lists (ACLs) are supported.</td>
</tr>
<tr>
<td><code>PC_ACL_MAX</code></td>
<td>Maximum number of entries allowed in an ACL.</td>
</tr>
<tr>
<td><code>PC_LINK_MAX</code></td>
<td>Maximum value of a file's link count.</td>
</tr>
<tr>
<td><code>PC_MAX_CANON</code></td>
<td>Maximum number of bytes in a terminal canonical input line.</td>
</tr>
<tr>
<td><code>PC_MAX_INPUT</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>PC_NAME_MAX</code></td>
<td>Maximum number of bytes in a filename (not a string length; count excludes a terminating null).</td>
</tr>
<tr>
<td><code>PC_PATH_MAX</code></td>
<td>Maximum number of bytes in a pathname (not a string length; count excludes a terminating null).</td>
</tr>
<tr>
<td><code>PC_PIPE_BUF</code></td>
<td>Maximum number of bytes that can be written atomically when writing to a pipe.</td>
</tr>
<tr>
<td><code>PC_POSIX_CHOWN_RESTRICTED</code></td>
<td>Change ownership 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.</td>
</tr>
<tr>
<td><code>PC_POSIX_NO_TRUNC</code></td>
<td>Pathname components longer than 255 bytes generate an error.</td>
</tr>
</tbody>
</table>
Variable Description
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
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 10 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 170 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

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. For more information about the use of pfsctl, see [z/OS DFSMS Using Data Sets](https://www.ibm.com). With pfsctl, your status utility program can easily fetch the information it needs from the physical file system.
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 76 for the flag O_NONBLOCK
- "f_setfd" on page 75 for the flag FD_CLOEXEC

Example
To create a pipe:
"pipe pfd."
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:

<table>
<thead>
<tr>
<th>Number</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>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.</td>
</tr>
<tr>
<td>0 or -1</td>
<td>This file cannot support 3270 passthrough mode.</td>
</tr>
<tr>
<td>1</td>
<td>This file can support 3270 passthrough mode.</td>
</tr>
<tr>
<td>3</td>
<td>This file is currently in 3270 passthrough mode.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>Reset 3270 passthrough support for this file.</td>
</tr>
</tbody>
</table>

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
/* 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
```

Chapter 3. The syscall commands  139
"write 1 buf" /* send command to OMVS */
"read 1 ibuf 1000" /* discard the response */
'pt3270 1 3' /* reset passthrough mode */
return 0
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"
Function
rddir invokes the readdir callable service to read multiple name entries from a directory and format the information in the readdir callable service format. To format this type of information in a stem, see “readdir” on page 146. 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 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.
The caller of the readdir service should check for both conditions.

**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"
```
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
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 a return code of 0, and ERRNO of EAGAIN. 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 SIGTTOU 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 SIGTTOU signal is not blocked or ignored, and
- The process group of the process is not orphaned.

If these conditions are met, SIGTTOU is sent. If SIGTTOU has a handler, the handler gets control and the read ends with the return code set to EINTRO. If SIGTTOU 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"
```
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 83 for a list of the ST_ variables.

Usage notes
The readdir command reads a directory in the readdir callable service format. You can use opendir and closedir together with the readdir 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

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 contains the number of lines read. stem.1 through stem.n (where n is the number of lines) each contain a line read.

Usage notes
1. The maximum allowable length of a line in the file is 1024 characters. RC=4 is returned when a line is greater than 1024 characters long or is not delimited with a newline character.
2. The newline characters that delimit the lines in a text file are stripped before the lines are saved in the stem.
3. When the RC indicates success, then Retval will be equal to or greater than zero and indicates the corresponding file description number.

Example
In the following example, assume that pathname was assigned a value earlier in the exec:
"readfile (pathname) file."
**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"
```
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

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 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 152.

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)"
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 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"
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"
**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
```
**Function**

The `seteuid` callable service is used 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)"
```
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)"
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 92.
setgroups

Function
setgroups invokes the setgroups callable service to set the supplemental group list for the process.

Parameters
stem
   The name of a stem variable used to set the group list. Upon return, stem.0 contains the number of variables containing a group id. stem.1 to stem.n (where n is the number of variables) each contain one group id number.

Usage notes
A RETVAL of −1 indicates failure.

Example
In the following example, assume that the stem gr. was setup earlier in the exec:
"setgroups gr."
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
Function
setpwent invokes the setpwent callable service to effectively rewind the user database to allow repeated searches. For more information, see “getpwent” on page 109.
Function

setregid invokes the setregid callable service to set the real or effective GIDs for the calling process. If a specified value is set to -1, the corresponding real or effective GID of the calling process is left unchanged.

Parameters

rgid
The numeric GID value that becomes the real GID for the calling process.

egid
The numeric GID value that becomes the effective GID for the calling process.

Example

In the following example, assume that rgid and egid were assigned values earlier in the exec:

```
setregid rgid egid
```
Function

setreuid invokes the setreuid callable service to set the real or effective user IDs (UIDs) of the calling process. If a specified value is set to -1, the corresponding real or effective UID of the calling process is left unchanged.

Parameters

ruid
The numeric UID value that becomes the real UID for the calling process.

euid
The numeric UID value that becomes the effective UID for the calling process.

Example

In the following example, assume that ruid and euid were assigned values earlier in the exec:

setreuid ruid euid
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.)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Allowable Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLIMIT_AS</td>
<td>Maximum address space size for a process.</td>
<td>10 485 760—2 147 483 647</td>
</tr>
<tr>
<td>RLIMIT_CORE</td>
<td>Maximum size (in bytes) of a core dump created by a process.</td>
<td>0—2 147 483 647</td>
</tr>
<tr>
<td>RLIMIT_CPU</td>
<td>Maximum amount of CPU time (in seconds) used by a process.</td>
<td>7—2 147 483 647</td>
</tr>
<tr>
<td>RLIMITFSIZE</td>
<td>Maximum files size (in bytes) created by a process.</td>
<td>0—2 147 483 647</td>
</tr>
<tr>
<td>RLIMTNOCORE</td>
<td>Maximum number of open file descriptors for a process.</td>
<td>5—131 072</td>
</tr>
</tbody>
</table>

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.
setrlimit

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 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 [z/OS 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. If the limit that is specified for RLIMIT_NOFILE is higher than that currently supported by the system, the limit will be reduced to the system maximum when it is used.

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 RLIMITFSIZE, 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 the current limit to 50:
```
r.2=100
r.1=50
"setrlimit" rlimit_noFile r.
```
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
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

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

| 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. |

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>signal</td>
</tr>
<tr>
<td>The signal, as specified by a numeric value or a predefined variable beginning with SIG.</td>
</tr>
</tbody>
</table>

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

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

**new_handler**

Specifies the new setting for handling the signal. The following predefined variables can be used for *new_handler*:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG_CAT</td>
<td>Set signal handling to catch the signal.</td>
</tr>
<tr>
<td>SIG_DFL</td>
<td>Set signal handling to the default action.</td>
</tr>
<tr>
<td>SIG_IGN</td>
<td>Set signal handling to ignore the signal.</td>
</tr>
<tr>
<td>SIG_QRY</td>
<td>Query the handling for that signal.</td>
</tr>
</tbody>
</table>

**new_flag**

Used to modify the behavior of the specified signal. Specify one of these:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use the default behavior.</td>
</tr>
<tr>
<td>SA_NOCLDWAIT</td>
<td>Do not create zombie processes when child processes exit.</td>
</tr>
<tr>
<td>SA_RESETHAND</td>
<td>Reset the signal action to the default, SIG_DFL, when delivered.</td>
</tr>
<tr>
<td>SA_NOCLDSTOP</td>
<td>Do not generate SIGCHLD when the child processes stop.</td>
</tr>
</tbody>
</table>

**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.
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 "prevhndl prevflag"
```
sigpending

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

Function

sigprocmask invokes the sigprocmask callable service to examine or change the calling thread’s signal mask.

Parameters

class 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG_BLOCK</td>
<td>Add the signals in new_mask to those to be blocked for this thread.</td>
</tr>
<tr>
<td>SIG_SETMASK</td>
<td>Replace the thread’s signal mask with new_mask.</td>
</tr>
<tr>
<td>SIG_UNBLOCK</td>
<td>Delete the signals in new_mask from those blocked for this thread.</td>
</tr>
</tbody>
</table>

class 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 10.

class 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 \textit{newsigset} was assigned a value earlier in the exec:

```
sigprocmask sig_setmask newsigset "oldsigset"
```
sigsuspend

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 10.

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
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 10.

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)
```
**Function**

**spawn** invokes the spawn callable service to create a new process, called a *child process*, to run an 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.
spawn

- 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 FCTLCLOSEXEC or FCTLCLOFORK. The files marked FCTLCLOSEXEC 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 FCTLCLOSEXEC 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 setuid 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 setgid 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
evacallable 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.

7. When the executable file to be run is a REXX exec, the first argument should be
   the pathname of the REXX exec. Subsequent arguments for the exec can follow
   this.

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:

```bash
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.'
```
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 174 for more information.
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 123.

To use a file descriptor to obtain this information, see “fstat” on page 83.

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.)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_AUDIT</td>
<td>Auditor audit information</td>
</tr>
<tr>
<td>ST_ACCESSACL</td>
<td>1 if there is an access ACL (access control list)</td>
</tr>
<tr>
<td>ST_ATIME</td>
<td>Time of last access</td>
</tr>
<tr>
<td>ST_AUDITID</td>
<td>RACF File ID for auditing</td>
</tr>
<tr>
<td>ST_BLKSIZE</td>
<td>File block size</td>
</tr>
<tr>
<td>ST_BLOCKS</td>
<td>Blocks allocated</td>
</tr>
<tr>
<td>ST_CCSID</td>
<td>Coded character set ID; first 4 characters are the file tag</td>
</tr>
<tr>
<td>ST_CRTIME</td>
<td>File creation time</td>
</tr>
<tr>
<td>ST_CTIME</td>
<td>Time of last file status change</td>
</tr>
<tr>
<td>ST_DEV</td>
<td>Device ID of the file</td>
</tr>
<tr>
<td>ST_DMODEACL</td>
<td>1 if there is a directory model access control list (ACL)</td>
</tr>
<tr>
<td>ST_EXTLINK</td>
<td>External symbolic link flag, set to 0 or 1</td>
</tr>
<tr>
<td>ST_FID</td>
<td>File identifier</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ST_FILEFMT</td>
<td>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:</td>
</tr>
<tr>
<td>S_FFBINARY</td>
<td>Binary data</td>
</tr>
<tr>
<td>S_FFCR</td>
<td>Text data delimited by a carriage return character</td>
</tr>
<tr>
<td>S_FFCRNL</td>
<td>Text data delimited by carriage return and line feed characters</td>
</tr>
<tr>
<td>S_FFCRLF</td>
<td>A text file with lines delimited by carriage-return and newline characters.</td>
</tr>
<tr>
<td>S_FFLF</td>
<td>Text data delimited by a line feed character</td>
</tr>
<tr>
<td>S_FFLFCR</td>
<td>Text data delimited by a line feed and carriage return characters</td>
</tr>
<tr>
<td>S_FFNA</td>
<td>Text data with the file format not specified</td>
</tr>
<tr>
<td>S_FFNL</td>
<td>Text data delimited by a newline character</td>
</tr>
<tr>
<td>S_FFRECORD</td>
<td>File data consisting of records with prefixes. The record prefix contains the length of the record that follows.</td>
</tr>
<tr>
<td>ST_FMODEACL</td>
<td>1 if there is a file model ACL.</td>
</tr>
<tr>
<td>ST_GENVALUE</td>
<td>General attribute values</td>
</tr>
<tr>
<td>ST_GID</td>
<td>Group ID of the group of the file</td>
</tr>
<tr>
<td>ST_INO</td>
<td>File serial number</td>
</tr>
<tr>
<td>ST_MAJOR</td>
<td>Major number for a character special file</td>
</tr>
<tr>
<td>ST_MINOR</td>
<td>Minor number for a character special file</td>
</tr>
<tr>
<td>ST_MODE</td>
<td>File mode, permission bits only</td>
</tr>
<tr>
<td>ST_MTIME</td>
<td>Time of last data modification</td>
</tr>
<tr>
<td>ST_NLINK</td>
<td>Number of links</td>
</tr>
<tr>
<td>ST_RTIME</td>
<td>File backup time stamp (reference time)</td>
</tr>
<tr>
<td>ST_SETGID</td>
<td>Set Group ID on execution flag, set to 0 or 1</td>
</tr>
<tr>
<td>ST_SETUID</td>
<td>Set User ID on execution flag, set to 0 or 1</td>
</tr>
<tr>
<td>ST_SIZE</td>
<td>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).</td>
</tr>
<tr>
<td>ST_SECLABEL</td>
<td>Security Label</td>
</tr>
<tr>
<td>ST_STICKY</td>
<td>Sticky bit flag (keep loaded executable in storage), set to 0 or 1</td>
</tr>
<tr>
<td>ST_TYPE</td>
<td>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:</td>
</tr>
<tr>
<td>S_ISCHR</td>
<td>Character special file</td>
</tr>
<tr>
<td>S_ISDIR</td>
<td>Directory</td>
</tr>
<tr>
<td>S_ISFIFO</td>
<td>FIFO special file</td>
</tr>
<tr>
<td>S_ISREG</td>
<td>Regular file</td>
</tr>
<tr>
<td>S_ISSYM</td>
<td>Symbolic link</td>
</tr>
<tr>
<td>ST_UAUDIT</td>
<td>Area for user audit information</td>
</tr>
<tr>
<td>ST_UID</td>
<td>User ID of the owner of the file</td>
</tr>
</tbody>
</table>
**stat**

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."
```
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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STFS_AVAIL</td>
<td>Space available to unprivileged users in block-size units.</td>
</tr>
<tr>
<td>STFS_BFREE</td>
<td>Total number of free blocks.</td>
</tr>
<tr>
<td>STFS_BLOCKSIZE</td>
<td>Block size.</td>
</tr>
<tr>
<td>STFS_FAVAIL</td>
<td>Number of free file nodes available to unprivileged users.</td>
</tr>
<tr>
<td>STFS_FFREE</td>
<td>Total number of free file nodes.</td>
</tr>
<tr>
<td>STFS_FILES</td>
<td>Total number of file nodes in the file system.</td>
</tr>
<tr>
<td>STFS_FRSIZE</td>
<td>Fundamental file system block size.</td>
</tr>
<tr>
<td>STFS_FSID</td>
<td>File system ID set by the logical file system.</td>
</tr>
<tr>
<td>STFS_INUSE</td>
<td>Allocated space in block-size units.</td>
</tr>
<tr>
<td>STFS_INTRUS</td>
<td>Number of seconds the file system will remain unchanged.</td>
</tr>
<tr>
<td>STFS_NAMEMAX</td>
<td>Maximum length of file name.</td>
</tr>
<tr>
<td>STFS_NOSEC</td>
<td>Mount data set with no security bit.</td>
</tr>
<tr>
<td>STFS_NOSUID</td>
<td>SETUID and SETGID are not supported.</td>
</tr>
<tr>
<td>STFS_RDONLY</td>
<td>File system is read-only.</td>
</tr>
<tr>
<td>STFS_TOTAL</td>
<td>Total space in block-size units.</td>
</tr>
</tbody>
</table>

Example

In the following example, assume that `fname` was assigned a value earlier in the exec:

```bash
"statfs (fname) st."
```
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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STFS_AVAIL</td>
<td>Space available to unprivileged users in block-size units.</td>
</tr>
<tr>
<td>STFS_BFREE</td>
<td>Total number of free blocks.</td>
</tr>
<tr>
<td>STFS_BLOCKSIZE</td>
<td>Block size.</td>
</tr>
<tr>
<td>STFS_FAVAIL</td>
<td>Number of free file nodes available to unprivileged users.</td>
</tr>
<tr>
<td>STFS_FFREE</td>
<td>Total number of free file nodes.</td>
</tr>
<tr>
<td>STFS_FILES</td>
<td>Total number of file nodes in the file system.</td>
</tr>
<tr>
<td>STFS_FRSIZE</td>
<td>Fundamental file system block size.</td>
</tr>
<tr>
<td>STFS_FSID</td>
<td>File system ID set by the logical file system.</td>
</tr>
<tr>
<td>STFS_INUSE</td>
<td>Allocated space in block-size units.</td>
</tr>
<tr>
<td>STFS_INVARSEC</td>
<td>Number of seconds the file system will remain unchanged.</td>
</tr>
<tr>
<td>STFS_NAMEMAX</td>
<td>Maximum length of file name.</td>
</tr>
<tr>
<td>STFS_NOSEC</td>
<td>Mount data set with no security bit.</td>
</tr>
<tr>
<td>STFS_NOSUID</td>
<td>SETUID and SETGID are not supported.</td>
</tr>
<tr>
<td>STFS_RDONLY</td>
<td>File system is read-only.</td>
</tr>
<tr>
<td>STFS_TOTAL</td>
<td>Total space in block-size units.</td>
</tr>
</tbody>
</table>

Example

In the following example, assume that `fsname` was assigned a value earlier in the `exec`:

```
"statvfs (fsname) st."
```
strerror

Function
strerror retrieves diagnostic text for error codes and reason codes.

Parameters
error_code
Hex value for an error code as returned in ERRNO for other SYSCALL host commands. Specify 0 if text for this code is not being requested.

reason_code
Hex value for the reason code as returned in ERRNOJR for other SYSCALL host commands. Specify 0 if this code is not being requested.

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 SE_ or their equivalent numeric values to access the values that they represent. See Appendix A for the numeric values. For example, stem.se_reason accesses the reason code text. If text is unavailable, a null string is returned.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE_ERRNO</td>
<td>Text for the error number.</td>
</tr>
<tr>
<td>SE_REASON</td>
<td>Text for the reason code.</td>
</tr>
<tr>
<td>SE_ACTION</td>
<td>Text for any action to be taken to correct this error. This variable will be available only when a reason code is requested.</td>
</tr>
<tr>
<td>SE_MODID</td>
<td>Name of the module that detected the error. This variable will be available only when a reason code is requested.</td>
</tr>
</tbody>
</table>

Example
To get error text for the last syscall error:
"strerror" errno errnojr "err."
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 120), 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"
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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_ARG_MAX</td>
<td>The maximum length of all arguments and environment strings to exec()</td>
</tr>
<tr>
<td>SC_CHILD_MAX</td>
<td>The maximum number of simultaneous processes per real user ID</td>
</tr>
<tr>
<td>SC_CLK_TCK</td>
<td>The number of intervals per second used in defining the type clock_t, which is used to measure process execution times</td>
</tr>
<tr>
<td>SC_JOB_CONTROL</td>
<td>Support for job control</td>
</tr>
<tr>
<td>SC_NGROUPS_MAX</td>
<td>Maximum number of simultaneous supplementary group IDs per process</td>
</tr>
<tr>
<td>SC_OPEN_MAX</td>
<td>Maximum number of simultaneous open files per process</td>
</tr>
<tr>
<td>SC_SAVED_IDS</td>
<td>Support for saved set-user-IDs and set-group-IDs</td>
</tr>
<tr>
<td>SC_THREAD_TASKS_MAX_NP</td>
<td>Constant for querying the maximum number of threaded tasks per calling process</td>
</tr>
<tr>
<td>SC_THREADS_MAX_NP</td>
<td>Constant for querying the maximum number of threads per calling process</td>
</tr>
<tr>
<td>SC_TZNAME_MAX</td>
<td>The number of bytes supported for the name of a time zone</td>
</tr>
<tr>
<td>SC_VERSION</td>
<td>The integer value 199009L</td>
</tr>
<tr>
<td>SC_2_CHAR_TERM</td>
<td>Constant for querying whether the system supports at least one raw mode terminal</td>
</tr>
</tbody>
</table>

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
maxopenfiles = retval /*value of desired configurable system variable is returned in retval*/
```
Function

```
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.
```
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 z/OS UNIX, 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, "REXX predefined variables," on page 291.) For example, you could specify stem.4 or stem.tms_cstime to obtain system CPU values:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS_CSTIME</td>
<td>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.</td>
</tr>
<tr>
<td>TMS_CUTIME</td>
<td>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.</td>
</tr>
<tr>
<td>TMS_STIME</td>
<td>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.</td>
</tr>
<tr>
<td>TMS_UTIME</td>
<td>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.</td>
</tr>
</tbody>
</table>

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."
Function

The `trunc` callable service is used to change the size of the file identified by the `pathname` parameter.

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"
```
ttynametttyname

Function
ttynametttyname invokes the ttynametttyname 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:
"ttynametttyname 0 path"
umask

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 invokes the uname callable service to obtain information about the 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_MACHINE</td>
<td>The name of the hardware type on which the system is running</td>
</tr>
<tr>
<td>U_NODENAME</td>
<td>The name of this node within the communication network</td>
</tr>
<tr>
<td>U_RELEASE</td>
<td>The current release level of this implementation</td>
</tr>
<tr>
<td>U_SYSNAME</td>
<td>The name of this implementation of the operating system (z/OS)</td>
</tr>
<tr>
<td>U_VERSION</td>
<td>The current version level of this release</td>
</tr>
</tbody>
</table>

Example

"uname uts."
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 152.

Example
 In the following example, assume that file was assigned a value earlier in the exec:
"unlink (file)"
Function

unmount invokes the umount callable service to unmount a file system; that is, it removes a file system from the file hierarchy.

Requirement: The caller must have unmount authorities to unmount a file system. See the section on mount authority in z/OS UNIX System Services Planning.

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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTM_DRAIN</td>
<td>An unmount drain request. All uses of the file system are normally ended before the file system is unmounted.</td>
</tr>
<tr>
<td>MTM_FORCE</td>
<td>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.</td>
</tr>
<tr>
<td>MTM_IMMED</td>
<td>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.</td>
</tr>
<tr>
<td>MTM_NORMAL</td>
<td>A normal unmount request. If no one is using the named file system, the unmount request is done. Otherwise, the request is rejected.</td>
</tr>
<tr>
<td>MTM_RESET</td>
<td>A reset unmount request. This stops a previous unmount drain request.</td>
</tr>
<tr>
<td>MTM_REMOUNT</td>
<td>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.</td>
</tr>
<tr>
<td>MTM_SAMEMODE</td>
<td>Remounts the file system without changing the mount mode. This can be used to regain the use of a file system that has I/O errors.</td>
</tr>
</tbody>
</table>

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

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

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

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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W_CONTINUED</td>
<td>Process continued from stop</td>
</tr>
<tr>
<td>W_EXITSTATUS</td>
<td>The exit status of the child process</td>
</tr>
<tr>
<td>W_IFEXITED</td>
<td>The child process ended normally</td>
</tr>
<tr>
<td>W_IFSIGNALED</td>
<td>The child process ended because of a signal that was not caught</td>
</tr>
<tr>
<td>W_IFSTOPPED</td>
<td>Wait if the child process is stopped</td>
</tr>
<tr>
<td>W_STAT3</td>
<td>Byte 3 of the BPXYWAST macro. See BPXYWAST — Map the Wait Status Word in z/OS UNIX System Services Programming: Assembler Callable Services Reference.</td>
</tr>
<tr>
<td>W_STAT4</td>
<td>Byte 4 of the BPXYWAST macro. See BPXYWAST — Map the Wait Status Word in z/OS UNIX System Services Programming: Assembler Callable Services Reference.</td>
</tr>
<tr>
<td>W_STOPSIG</td>
<td>The signal number that caused the child process to stop</td>
</tr>
<tr>
<td>W_TERMSIG</td>
<td>The signal number that caused the child process to end</td>
</tr>
</tbody>
</table>

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.
wait

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 214 for an example of signal coding that interprets the stem this returns:
"wait wstat."
waitpid

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 197).

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.)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Wait for the child process to end (default processing)</td>
</tr>
<tr>
<td>W_NOHANG</td>
<td>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.</td>
</tr>
<tr>
<td>W_UNTRACED</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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 214](#) for an example of signal coding that interprets the stem this returns:

"waitpid -1 wst. 0"
**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

Function
writefile invokes the open, write, and close callable services to write or append 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 that contains the information to be written to the file. stem.0 is set to the number of lines to be 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 201.

append
An optional flag to indicate that the information is to be appended to the file (the file must already exist), as follows:
0 Do not append. (This is the default value.)
1 Append the information to the file.

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 46
- "open" on page 131
- "write" on page 201

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: Using syscall commands

The examples in this chapter are provided to assist you with coding REXX programs that use z/OS UNIX syscall commands.

The first three examples can run in TSO/E, batch, or from the z/OS shells. 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
/* 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:

```rexx
'writefile /u/schoen/root.dir 777 root.'
```

Open, write, and close a file

```rexx
/* rexx */
call syscalls 'ON'
address syscall
path='/u/schoen/my.file'
'open' path,
    0_rdwr+0_creat+0_trunc,
    660
if retval=-1 then
   do
      say 'file not opened, error codes' errnoerrnojr
      return
   end
fd=retval
rec='hello world' || esc_n
'write' fd 'rec' length(rec)
if retval=-1 then
   say 'record not written, error codes' errnoerrnojr
'close' fd
```

Open a file, read, and close it

```rexx
/* rexx */
call syscalls 'ON'
address syscall
path='/u/schoen/my.file'
'open' (path),
    0_rdonly,
    000
if retval=-1 then
   do
      say 'file not opened, error codes' errnoerrnojr
      return
   end
fd=retval
'read' fd 'bytes 80'
```
Display the working directory and list a specified directory

This REXX program runs in the z/OS shells; 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
/\ rexx */
parse arg dir
address syscall 'getcwd cwd'
say 'current directory is' cwd
if dir=' ' then
do
 say 'enter directory name to list'
purge dir
end
'ls -l' dir
return
```

Parse arguments passed to a REXX program: the getopt 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 a shell. This function parses the `__argv` for options in the format used by most POSIX commands.

```rexx
/***********************************************
* Function: getopt                             *
* 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<>1 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
                     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
   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
/* 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 */
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 /* open failed */
      say 'unable to open' fi
   iterate
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 */
   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 */
```

208  z/OS V1R13.0 Using REXX and z/OS UNIX System Services
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. This sample REXX program, as coded, must be run from a shell.

```rexx
/* 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.i)>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 path name and the file system name can be specified in any order. `stat` is used to determine which name is the path name.

For the `getopts` function called in this program, see "Parse arguments passed to a REXX program: the `getopts` function" on page 206.
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 z/OS UNIX 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. */
Unmount a file system

This REXX program uses `unmount` to unmount a file system, an action that requires mount authority as described in the section on mount authority in z/OS UNIX System Services Planning. 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 path name 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.
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. */
/*************************************************************************/
"unmount (fsn)" mtm_normal /* unmount name as specified */
if retval =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:
      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 */
  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
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` 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
/* rexx */
address syscall 'pipe p.' /* make a pipe */
address syscall '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`, `getgrnam`, 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
/* rexx */
arg group .
address syscall
if datatype(group,'W')=1 then
  'getgrgid (group) gr.' /* use getgrgid if GID specified */
else
  'getgrnam (group) gr.' /* use getgrnam if group name specified */
if retval=0 then /* check for error */
do
  say 'No information available for group' group
return 1
end
```
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
/* 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 */
  j=j+1
  '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
/* REXX */
address syscall
/* initialize file descriptor map (see note 1) */
fd.0=-1
fd.2=-1
```

214  z/OS V1R13.0 Using REXX and z/OS UNIX System Services
'creat /tmp/dirlst 600'
fd.1=retval
/* initialize parameter stem (see note 2) */
parm.1='/bin/ls'
parm.2='-l'
parm.3=''
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 path name 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.
List the ACL entries for a file

This example will show the access ACL entries for a path name given as a parameter:

```rexx
/* REXX */
parse arg path
call syscalls 'ON'
address syscall
'aclinit acl' /* init variable ACL to hold acl */
'aclget acl (path) acl_type_access /* get the file access acl */
do i=1 by 1 /* get each acl entry */
  'aclgetentry acl acl.' i
do i=1 by 1 /* get each acl entry */
  if rc<0 | retval=-1 then leave /* error, assume no more */
  parse value '- - -' with pr pw px
  if acl.acl_read=1 then pr='R' /* set rwx permissions */
  if acl.acl_write=1 then pw='W'
  if acl.acl_execute=1 then px='X'
  aclid=acl.acl_id /* get uid or gid */
  /* determine acl type */
  if acl.acl_entry_type=acl_entry_user then type='UID='
  else
    if acl.acl_entry_type=acl_entry_group then type='GID='
  else
    type='???'
  say pr || pw || px type || aclid /* print line */
end
'aclfree acl' /* must free acl buffer */
```
Chapter 5. z/OS UNIX REXX functions

The z/OS UNIX REXX functions extend the REXX language on z/OS when it is used in the z/OS UNIX REXX environment. With the exception of bpxwunix() and syscalls(), these functions must be run in a z/OS UNIX environment. The z/OS UNIX REXX functions include functions for standard REXX I/O, and for accessing common file services and environment variables.

All of the functions are fully enabled for large files (>2GB).

All numbers that are used as input on the functions must be integers. The default precision for REXX is 9 digits. If arithmetic is used on large numbers, be sure to change your precision appropriately, using the NUMERIC DIGITS statement.

REXX I/O functions

The REXX input functions are charin() and linein(). The chars() and lines() functions determine if data remains in an input stream.

The REXX output functions are charout() and lineout().

The REXX stream() function controls the processing of I/O streams (file streams and process streams) and obtains their status. A number of commands are used within the stream() function to control stream processing.

Opening a stream implicitly

File streams can be opened implicitly or explicitly. You open a stream implicitly by using a pathname as the stream name for one of the six input or output functions. If the function is charout() or lineout(), the file is opened for output. If the file does not exist, it is created. For newly created files, the permission bits 0666 are applied to your process umask. The position for the first write is set to the end of file, unless the function call explicitly specifies a write location. If one of the four input functions is used, the file is opened for input.

When it is opened implicitly, a stream can be opened for both input and output. The input and output locations are independent of each other. If the stream is opened for both input and output, two file descriptors are used. The stream close command closes the stream and both file descriptors.

Opening a stream explicitly

You open a stream explicitly by using the stream() function.

To explicitly open a file stream, use the open command. The advantage of opening file streams explicitly is that the program can determine that the stream open failed. The program can also have several separate streams for the same file.

The stream open command fails if:
• The file does not exist.
• The pathname cannot be accessed for either input or output.

If the file cannot be opened, a message is written to stderr. The I/O function returns as if the stream is empty, at end of file, and the file cannot be extended.
When a stream is explicitly opened, the `stream()` function returns a string that is the name of the stream. This is the only name that can be used to identify that stream. Multiple opens for the same pathname open multiple streams, each with its own name.

**Process streams**

Process streams can only be opened explicitly, with the `popen` command. With process streams, you can run a shell command and provide its input or receive its output. To write the input for the command, use the `popen write` command. To receive the command’s output, use the `popen read` command. The `stream()` function returns a string that is the name of the process stream. Use this name on the input functions for `popen read` or the output functions for `popen write`.

A process stream spawns `/bin/sh -c your_command`. The process inherits `stderr` and either `stdin` or `stdout`. If the process is opened for read, `stdout` for the process is a pipe; otherwise, `stdin` is a pipe. The shell completion code is returned as the result for the stream `pclose` command. `pclose` closes your end of the pipe and waits for the process to terminate.
Function
Runs a shell command and optionally:
- Provides its stdin
- Traps its stdout
- Traps its stderr
- Exports a set of environment variables

Parameters

**cmd**
The shell command that you want to run. The shell is run as /bin/sh -c followed by the string you specify as the command.

**stdin**
An optional argument, stdin is the name of a compound variable (stem) that contains input for the command.

stdin.0 must contain the number of lines that are to be redirected to the command. stdin.1, stdin.2, ... contain the lines.

stdin can also be specified as:
- The string STACK, if the input is on the stack.
- DD:ddname, if the input is to be read from an allocated DD.

If this argument is not specified, your current stdin file is passed to the shell for stdin.

**stdout**
An optional argument, stdout is the name of a compound variable (stem) that, on return, contains the normal output from the command. stdout.0 is the number of lines output by the command. stdout.1, stdout.2, ... contain the output lines.

stdout can also be specified as:
- The string STACK, if the output is to be returned on the stack
- DD:ddname, if the output is to be written to an allocated DD

If stdout is not specified, your current stdout file is passed to the shell for stdout.

**stderr**
An optional argument, stderr is the name of a compound variable (stem) that, on return, contains the error output from the command. stderr.0 is the number of lines output by the command. stderr.1, stderr.2, ... contain the output lines.

stderr can also be specified as:
bpwxunix()

- The string STACK, if the output is to be returned on the stack
- DD:ddname, if the output is to be written to an allocated DD

If this argument is not specified, your current stderr file is passed to the shell for stderr.

env

An optional argument, env is the name of a compound variable (stem) that contains environment variables for the command. env.0 must contain the number of environment variables to be passed to the command. env.1, env.2, ... contain the variables in the form variable_name=variable_value. If env is not specified, your current environment is passed to the shell for stdin.

Usage notes

1. The bpwxunix() function runs the shell by passing a single command similar to sh -c cmd. It does not run a login shell.
2. bpwxunix() can be used outside of the z/OS UNIX REXX environment (for example, in TSO/E). In this case, stdin, stdout, stderr, and environment variables are not inherited from the current process environment.
3. If the stdout or stderr stems are specified, they are filled as appropriate:
   - A return value in the range 0-255 is the exit status of the command.
   - A negative return value indicates failure, and is usually a signal number.
   - A number less than −1000 indicates a stop code.
4. The DD names that are used for input and output are processed by the standard REXX input and output services. They have the same restrictions as REXX in terms of the types of allocations they can handle.
5. STDOUT lines cannot exceed 2048 characters.

Example

To trap output from the ls command and display it:

call bpwxunix 'ls -l',out.
   do i=1 to out.0
      say out.i
   end

To send output from the above example to word count and print byte count:

call bpwxunix 'wc',out.,bc.
   parse var bc.1... count
   say 'byte count is' count

To trap output on stack and feed it to word count:

if bpwxunix('ls -l',stack)=0 then
   call bpwxunix 'wc',stack
Function

Returns a string of up to length characters read from the stream specified by name. The location for the next read is the current location increased by the number of characters returned. This function does no editing of the data.

Parameters

name

The name for the stream can be a pathname or a string that was returned from the stream open or popen commands. If name is omitted, the standard input stream is used.

start

For a persistent stream, specifies the byte number in the file where the read begins. start should not be specified for other types of streams.

length

The number of bytes returned by the function. If length bytes are not available in the stream, the function returns the number of bytes that are available and marks an error condition on the stream. For non-persistent streams, this function either blocks until length bytes are available, or returns with fewer bytes, depending on the file type and open flags. If length is 0, no characters are read, a null string is returned, and the read position is set based on the value of start.

Example

To read the next 256 characters:

say charin(file,,256)

To set the read location to the sixth 80-byte record:

call charin file,5*80+1,0
Function

Returns the number of characters remaining after an attempt to write string to the stream specified by name. The location for the next write is the current location increased by the number of characters written.

Parameters

name
The name for the stream can be a pathname or a string that was returned from the stream open or popen commands. If name is omitted, the standard output stream is used.

string
Data to be written to the stream specified by name.

start
For a persistent stream, specifies the byte number in the file where the write begins. start should not be specified for other types of streams.

Usage notes

1. If string is omitted, no data is written, 0 is returned, and the write location is set to the value start.
2. If start is also omitted, the write position is set to the end of file.

Example

To write the string to stdout:
call charout,’hello world’esc_n

to set the write position to end:
call charout file
Function

Returns the number of characters remaining in the input stream specified by `name`. For persistent streams, this is the number of characters between the current read location and the end of the stream. If the stream was created by the `stream popen` command, `chars()`, while the process is active or bytes remain in the stream, returns either 1 or the number of bytes in the stream. After the process has terminated and the stream is empty, `chars()` returns a value of 0.

Parameters

`name`

The name for the stream, which can be a pathname or a string that was returned from the `stream open` or `popen` commands. If `name` is omitted, the standard input stream is used.

Example

To get the number of bytes in the `stdin` stream:
```
remainder=chars()
```
**Function**
Changes the mode for the specified pathname. It returns 0 if the mode for the specified pathname is changed; otherwise, it returns the system call error number.

**Parameters**
- **pathname**
  An absolute or relative pathname for a file.
- **operation**
  Specifies whether mode bits are to be set, added, or deleted:
  - `=` Set the mode bits. If an operation is not specified, `=` is used.
  - `+` Add the mode bits to what is currently set for the file.
  - `-` Remove the mode bits from what is currently set for the file.
- **mode**
  A string of octal digits for the new file mode.

**Example**
To set permissions for owner and group to read-write:
```rexx
call chmod file,660
```

To add read permission for other:
```rexx
call chmod file,+4
```
Function

Converts *timestamp* to POSIX epoch time, and returns the time in seconds past the POSIX epoch (1/1/1970).

Parameters

*timestamp*

A 14-character string in the form mmddyyyyHHMMSS.

Example

To set POSIX time for 4/21/99 7:15:00:

```
say convd2e('04211999071500')
```
Function

Returns the full pathname to the current directory, first changing it to newdirectory if the argument is supplied and you have access to that directory.

Parameters

newdirectory

An absolute or relative pathname for a directory.

Example

To change the current directory to /u/ehk:

call directory '/u/ehk'
Function
Queries and alters environment variables. The stem __environment. is not altered through this service. That stem contains the environment variables on entry to the REXX program, and is available for your use. Alterations of the environment are used on subsequent calls to the stream popen command and ADDRESS SH.

Parameters

variable name
The name of the environment variable to operate on. If this is the only argument specified, the value of the variable is returned and the variable is not affected.

new value
A string to replace the value of variable name. The previous value of the variable is returned.

operation
An optional argument that specifies the operation to be performed. Only the first character is significant. The values can be:

exists Tests the existence of the variable. The function returns 1 if the variable is defined, and 0 if it is not defined.

delete Deletes the variable from the environment, if it exists. The function returns 0 if the variable is successfully deleted, and 1 if the variable is not defined.

Example
To get the value of the PATH environment variable:
path=environment('PATH')

To reset PATH to the current directory:
call environment 'PATH','.'

To delete the PATH environment variable:
call environment 'PATH','d'
Function
Returns the full pathname for the specified file. If the file does not exist, the function returns a null string.

Parameters

filename
A string that names a file

Example
To print the full pathname for the file myfile:
say exists('myfile')
Function
Prints prompt on the controlling TTY and reads and returns one line of input with terminal echo suppressed.

Parameters
prompt
A string that prints on the controlling TTY

Example
To prompt for a password and read it:
psw=getpass('enter password')
Function

Returns one line or no lines from the stream specified in name, and sets the location for the next read to the beginning of the next line. The data is assumed to be text. The newline character is the line delimiter and is not returned. A null string is returned if no line is returned; this appears exactly the same as a null line in the file. Use chars() or lines() to determine if you are at end of file. Use stream() to determine if there is an error condition on the stream.

Parameters

name

The name for the stream can be a pathname or a string that was returned from the stream open or popen commands. If name is omitted, the standard input stream is used.

start

For a persistent stream only, this argument can have the value 1, to begin reading at the beginning of the stream. No other value for start is supported. Do not specify start for other types of streams.

count

Specify 0 or 1 for the number of lines to be returned by the function. If one line is not available in the stream, the function returns a null string and marks an error condition on the stream. For non-persistent streams, this function either blocks until a line is available, or returns as though it is end of file, depending on whether any data remains in the file, on the file open flags, and on the type of file. If count is 0, no lines are read, a null string is returned, and, if start is 1, the read position is set to the beginning of the stream.

Example

To read the next line:

```r
line=linein(file)
```

To read the first line in the file:

```r
line=linein(file,1)
```
Function
Returns 1 line or 0 lines that are remaining to be written after an attempt to write string to the stream specified by name. A newline character is written following string. If an error occurs on the write, some data may be written to the stream, and the function returns the value 1.

Parameters
name
The name for the stream can be a pathname or a string that was returned from the stream open or popen commands. If name is omitted, the standard output stream is used.

string
Data that is to be written to the stream specified by name. If string is omitted, no data is written, 0 is returned, and the write position is set based on the value of start. If both string and start are omitted, the write position is set to the beginning of the file.

start
For a persistent stream, this argument can have the value 1, to begin writing at the beginning of the stream. No other value for start is supported. Do not specify start for other types of streams.

Example
to write the line to stdout:
call lineout ,'hello world'
Function
Returns 1 if data remains in the stream; otherwise it returns 0. Programs should check for a value of 0 or nonzero.

Parameters
name
The name for the stream can be a pathname or a string that was returned from the stream open or popen commands. If name is omitted, the standard input stream is used.

Example
To set more to nonzero, if stdin has data:
more=lines()
Function
Enables or disables the trapping of output from commands run using ADDRESS TSO, and returns the name of the variable in which trapped output is stored. If trapping is off, the word OFF is returned. Note that **outtrap** does not trap output for ADDRESS SH or any other command environment besides TSO. To trap shell command output, see "bpxwunix()" on page 219.

Parameters

**name**
The name of a stem, a variable prefix used to contain command output, or the string **OFF** to turn off trapping.

**max**
The maximum number of lines to trap. If **max** is not specified, or if it is specified as * or a blank, the number of lines is set to 999 999 999.

**catopt**
Specify one of these:

**concat**
Each command output trapping begins following the previous command output.

**noconcat**
Output from each command is trapped starting with the variable concatenated with 1. Unused variables do not have their values altered.

For additional information about using the TSO command environment, see "The TSO command environment" on page 5.

Example
To trap TSO command output, use the stem **OUT**, rewriting the stem on each command:

```
call outtrap 'out.',,'NOCONCAT'
```
Function
Retrieves information about one or more processes.

Parameters

**pid**
The process ID number of the process for which information is to be returned.
If `pid` is not specified, basic information is retrieved for all processes you have authorization to view.

**request**
Specify one of these:

- **process**
  Retrieve information about the specified process. This is the default.
- **file**
  Retrieve file information for the specified process.
- **thread**
  Retrieve information for all threads in the specified process.

Results
Information is returned in simple and compound variables. Each variable name has a prefix that defaults to `bpixw_`. This prefix can be changed using the `rexxopt()` function. A stem can be used as the prefix, in which case the tails are the simple suffixes set by this function.

If `pid` is not specified, the following suffixes are set:

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID.n</td>
<td>Each variable contains one process ID number. <code>PID.0</code> is the number of PIDs returned. <code>PID.1, PID.2, ...</code> are the PID numbers.</td>
</tr>
<tr>
<td>THREADS.n</td>
<td>The number of threads for the process in the corresponding <code>PID.n</code></td>
</tr>
<tr>
<td>ASID.n</td>
<td>The address space ID for the process in the corresponding <code>PID.n</code></td>
</tr>
<tr>
<td>JOBNAME.n</td>
<td>The jobname for the process in the corresponding <code>PID.n</code></td>
</tr>
<tr>
<td>LOGNAME.n</td>
<td>The login name (user ID) for the process in the corresponding <code>PID.n</code></td>
</tr>
</tbody>
</table>

If process information is requested for a process, the following suffixes are set:
### procinfo()

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Description</th>
</tr>
</thead>
</table>
| STATE    | Contains 0 or more of the following strings:  
• MULPROCESS  
• SWAP  
• TRACE  
• STOPPED  
• INCOMPLETE  
• ZOMBIE |
| PID      | Process ID number |
| PPID     | Parent process ID number |
| PGRID    | Process group ID number |
| SID      | Session ID number |
| FPGID    | Foreground process group number |
| EUID     | Effective user ID |
| RUID     | Real user ID |
| SUID     | Saved set user ID |
| Egid     | Effective group ID |
| Rgid     | Real group ID |
| SGID     | Saved set group ID |
| SIZE     | Region size |
| SLOWPATH | Number of slow path syscalls |
| USERTIME | Time spent in user code |
| SYSTIME  | Time spent in system code |
| STARTTIME| Time when the process started (dub time) |
| OETHREADS| Number of z/OS UNIX threads |
| PTCREATE | Number of threads created using pthread_create |
| THREADS  | Number of threads in the process |
| ASID     | Address space ID of the process |
| JOBNAME  | Jobname for the process |
| LOGNAME  | Login name (user ID) for the process |
| CONTTY   | Pathname for the controlling TTY |
| CMDPATH  | Pathname for the command that started the process |
| CMDLINE  | Command line that started the process |

If file information is requested for a process, the following suffixes are set:

| TYPECD   | Type code of the file:  
• rd: root directory  
• cd: current directory  
• fd: file descriptor  
• vd: vnode descriptor |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Type number of the file</td>
</tr>
<tr>
<td>OPENF</td>
<td>Open flags</td>
</tr>
<tr>
<td>INODE</td>
<td>Inode number</td>
</tr>
</tbody>
</table>
### procinfo()

<table>
<thead>
<tr>
<th>DEVNO</th>
<th>Device number</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODES</td>
<td>Count of the total number of file nodes returned</td>
</tr>
</tbody>
</table>

If thread information is requested for a process, the following suffixes are set:

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THREAD_ID</td>
<td>Thread ID</td>
</tr>
<tr>
<td>SYSCALL</td>
<td>Current syscall, if in kernel</td>
</tr>
<tr>
<td>TCB</td>
<td>TCB address</td>
</tr>
<tr>
<td>RUNTIME</td>
<td>Time running in milliseconds</td>
</tr>
<tr>
<td>WAITTIME</td>
<td>Time waiting in milliseconds</td>
</tr>
<tr>
<td>SEMNUM</td>
<td>Semaphore number if on a semop (ptrunwait.x=’D’)</td>
</tr>
<tr>
<td>SEMVALUE</td>
<td>Semaphore value if on a semop (ptrunwait.x=’D’)</td>
</tr>
<tr>
<td>LATCHPID</td>
<td>Latch that the process waited for</td>
</tr>
<tr>
<td>SIGPENDMASK</td>
<td>Signal pending mask</td>
</tr>
<tr>
<td>LOGNAME</td>
<td>Login name</td>
</tr>
<tr>
<td>LASTSYSCALL_n</td>
<td>Last 5 syscalls</td>
</tr>
<tr>
<td>PTCREATED</td>
<td>Contains J if this was pthread-created</td>
</tr>
<tr>
<td>PTRUNWAIT</td>
<td>Contains one of the following letters, to indicate the current run or wait state of the thread:</td>
</tr>
<tr>
<td></td>
<td>A  msgrecv wait</td>
</tr>
<tr>
<td></td>
<td>B  msgsend wait</td>
</tr>
<tr>
<td></td>
<td>C  communication wait</td>
</tr>
<tr>
<td></td>
<td>D  Semaphore wait</td>
</tr>
<tr>
<td></td>
<td>F  File System Wait</td>
</tr>
<tr>
<td></td>
<td>G  MVS in Pause</td>
</tr>
<tr>
<td></td>
<td>K  Other kernel wait</td>
</tr>
<tr>
<td></td>
<td>P  PTwaiting</td>
</tr>
<tr>
<td></td>
<td>R  Running or non-kernel wait</td>
</tr>
<tr>
<td></td>
<td>S  Sleep</td>
</tr>
<tr>
<td></td>
<td>W  Waiting for child</td>
</tr>
<tr>
<td></td>
<td>X  Fork new process</td>
</tr>
<tr>
<td></td>
<td>Y  MVS wait</td>
</tr>
</tbody>
</table>
### procinfo()

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTTYPE</td>
<td>Contains one of the following letters to indicate thread type:</td>
</tr>
<tr>
<td>N</td>
<td>Medium-wait thread</td>
</tr>
<tr>
<td>O</td>
<td>Asynchronous thread</td>
</tr>
<tr>
<td>U</td>
<td>Initial process thread</td>
</tr>
<tr>
<td>Z</td>
<td>Process terminated and parent has not completed wait</td>
</tr>
<tr>
<td>PTDETACH</td>
<td>Contains V if the thread is detached</td>
</tr>
<tr>
<td>PTTRACE</td>
<td>Contains E if the thread is in quiesce freeze</td>
</tr>
<tr>
<td>PTTAG</td>
<td>Contains the ptagdata string if it exists</td>
</tr>
<tr>
<td>THREADS</td>
<td>Contains the total number of threads</td>
</tr>
</tbody>
</table>

### Usage notes

1. This function uses the __getthent callable service. For additional information, see __getthent (BPX1GTH, BPX4GTH) — Get thread data in z/OS UNIX System Services Programming: Assembler Callable Services Reference.

2. The procinfo() function returns as its value the prefix used to create variable names. If there is an error returned by the__getthent service, the function returns a null string, and sets the variables ERRNO and ERRNOJR to the hex values of the error codes returned by the__getthent service.

### Example

To show the command line for each process:
```rexx
(call procinfo
do i=1 to bpxw_pid.0
   if procinfo(bpxw_pid.1)='' then /* ignore processes that ended */
      say bpxw_cmdline
   end
end)
```

To show the current directory for each process:
```rexx
(call procinfo
do i=1 to bpxw_pid.0
   if procinfo(bpxw_pid.i,'file')='' then /* ignore processes that ended */
      do j=1 to bpxw_nodes
         if bpxw_typecd.j='cd' then
            do
               address syscall 'getmntent m.' bpxw_devno.j
               strm=stream('find ' m.mnte_path.1 ' -xdev -inum', bpxw_inode.j,,
                    'c', 'popen read')
               say right(bpxw_pid.i,12) linein(strm)
               call stream strm,'c','pclose'
            end
         leave
      end
   end
end)
```
Function

Sets, resets, or queries z/OS UNIX REXX options.

Parameters

- **option**
  Specify one of these:
  - **immed**
    Associates the immediate command interrupt handler to a signal number.
  - **noimmed**
    Restores the default action for a signal. arg1 is the signal number.
  - **version**
    Returns a string showing last compile time for the function package.
  - **varpref**
    Sets the variable name prefix used by some of the REXX functions. arg1 is the new prefix. If arg1 is not specified, the current value is not changed. If it is specified but null, it is defaulted to bpxw_. This returns a string that is the current setting.
  - **notsoin**
    Disables any input to the TSO command processor that was started using ADDRESS TSO, including the stack. This returns the string TSO input disabled.
  - **tsoin**
    Sets back to its default setting the input mode to the TSO command processor that was started using ADDRESS TSO. This returns the string TSO input enabled.
  - **arg1**
    A signal number.

Example

To make the interrupt signal prompt for a command:

```
call rexxo 'immed',sigint
```
Function
Places the process in a signals-enabled wait, and returns after the wait expires. If a signal interrupts the wait, the function returns the number of seconds remaining for the wait, otherwise it returns 0.

Parameters
seconds
A positive whole number for the number of seconds to sleep

Example
To wait 5 seconds:
call sleep 5
Function

Returns the state of the stream or the result of the command.

Parameters

name

The stream name for the operation. This argument is case-sensitive.

operation

The operation that is to be performed. Only the first character is significant.

The operations can be:

S Returns the current state of the stream. This is the default operation.
The following values can be returned:

ready The stream is available for normal input or output operations.

error The stream encountered an error on an input or output operation. After this call, the error condition is reset, and normal input or output operations can be attempted once again.

unknown The specified stream is not open.

D This operation is almost the same as S, with one exception. When ERROR is returned, it is followed by text describing the error. This text usually contains the error number and reason code if the failure is due to a system call failure.

C Execute a command for the specified stream. The command is specified as the third argument.

command

The stream command that is to be executed. This argument is valid only when the operation is C. The following commands are supported:

clearfile Truncates a file to zero bytes for a stream that is opened for write. clearfile should only be used on a persistent stream.

This example empties the file:
call stream name,'c','clearfile'

close Closes a stream. On success, the function returns the string ready. If the stream is not known, it returns the string UNKNOWN.

This example closes the stream:
call stream file,'c','close'

infileno Returns the file descriptor number of the input side of a
stream() stream. The file descriptor can be used on lower-level system calls, using, for example, ADDRESS SYSCALL. This example gets the file descriptor for the read side of the stream:

```rjsx
df=stream(name,'c','infileno')
```

nosignal Disables Halt Interruption for stream errors. See [242](#). This example disables the halt signal for the standard input/output stream:

```rjsx
call stream ,c',nosignal'
```

open `<open-type>` Opens a stream. `<open-type>` specifies either read or write. If `<open-type>` is not specified, read is assumed. The function returns a string that is the name to be used for the stream on subsequent I/O functions. This string is the only name by which this stream will be known. Note that streams do not have to be explicitly opened. A pathname that is used as a stream name on the other I/O functions will cause the stream to be opened, and the name of the stream will be that pathname. Optional arguments may be specified with write, optionally followed by octal permission bits. Write always opens the file with O_CREAT, creating the file if it does not exist. The additional arguments are:

- replace Opens the file with O_TRUNC, setting the file size to 0.
- append Opens the file with O_APPEND, causing all writes to be appended to the file.

This example opens a stream for the file `mydata.txt`:

```rjsx
file=stream('mydata.txt','c','open write')
```

This example opens a stream for the file `mydata.txt`, but replaces the file if it exists:

```rjsx
file=stream('mydata.txt','c','open write replace')
```

outfileno Returns the file descriptor number of the output side of a stream. The file descriptor can be used on lower-level system calls, using, for example, ADDRESS SYSCALL. This example gets the file descriptor for the write side of the stream:

```rjsx
fd=stream(name,'c','outfileno')
```

pclose Closes a process stream. On success, the function returns the completion code for the process run via the `popen` command.

pid Returns the process ID number for the shell process opened with `popen`. This example gets the PID for a stream opened with `popen`:

```rjsx
pid=stream(name,'c','pid')
```

popen `<open-type>` Opens a pipe to a shell command that is specified by the stream name. `<open-type>` must specify read or write. If read is specified, the input functions can be used to read the standard output from the command. If write is specified, the output functions can be used to pipe data to the standard input of the shell command. In either case, the shell command inherits the standard error file for the calling process.
stream()

The command that is run is always /bin/sh -c followed by the specified shell command. This means that the completion code is the one returned by the shell. It usually returns the command’s completion code. You can obtain the completion code by using the pclose command.

The function returns a string that is the name to be used for the stream on subsequent I/O functions. This string is the only name by which this stream will be known.

This example opens a pipe stream to the output from the shell command ls | wc:

```plaintext
file=stream('ls | wc','c','popen read')
```

query <attribute>
Queries a stream attribute and returns the result:

- **exists** Returns the full pathname of the stream name. This is equivalent to the exists() function, but is more portable. This command does not cause a stream to be opened.

- **size** Returns the size of the stream. This is equivalent to the size stream command, but is more portable.

This example prints the pathname of the stream:

```plaintext
say stream('myfile','c','query exists')
```

readpos [location]
Returns the position in the file where the next read will begin. If location is specified, the position is also set to the byte specified by location. location is specified as a number optionally preceded by one of the following characters:

- = An absolute byte location. This is the default.
- < An offset from the end of the stream.
- + An offset forward from the current location.
- - An offset backward from the current location.

This example gets the read location in the file, and then sets the read location to the sixth 80-byte record:

```plaintext
pos=stream(name,'c','readpos') /* get read location */
call stream name,'c','readpos' 5*80+1 /* set read location*/
```

signal
Enables Halt Interruption for stream errors. Note that the NOTREADY REXX signal is not supported. This example enables the halt signal for the standard input/output stream:

```plaintext
call stream ,c','signal'
```

size
Returns the size of the file associated with the stream. This example prints the size of the file:

```plaintext
say stream(name,'c','size')
```

writepos <location>
Returns the position in the file where the next write will begin. If location is specified, the position is also set to the byte specified by location. location is specified as a number optionally preceded by one of the following characters:
stream()

= An absolute byte location. This is the default.
< An offset from the end of the stream.
+ An offset forward from the current location.
− An offset backward from the current location.

This example sets the position to the end of the file:
call stream name,'c','writepos <0'

This example sets the position to the start of the file:
call stream name,'c','writepos 1'
Function

Submits a job to the primary subsystem (JES), returning the job ID of the submitted job.

Parameters

stem.

The stem compound variable contains the number of lines in stem.0, and each variable from stem.1, stem.2, ... contains a line for the job that is being submitted.

Example

This example reads the file into the stem, sets the number of lines, and submits the job:

do i=1 by 1 while lines(fn)>0
   fn.i=linein(fn)
end
fn.0=i-1
say submit('fn.')
syscalls()

Function

Establishes the SYSCALL environment or ends it; or establishes or deletes the signal interface routine (SIR).

Parameters

**ON**

Establishes the SYSCALL environment. It sets up the REXX predefined variables and blocks all signals.

**OFF**

Ends the connection between the current task and z/OS UNIX. It is usually not necessary to make a `syscalls (OFF)` call.

**SIGON**

Establishes the signal interface routine.

**SIGOFF**

Deletes the signal interface routine, and resets the signal process mask to block all signals that can be blocked.

Usage notes

1. This function can be used outside of the z/OS UNIX REXX environment (for example, in TSO/E). When `syscalls` is used in such an environment, `stdin`, `stdout`, `stderr`, and environment variables are not inherited from the current process environment.
2. For more usage information, see "The SYSCALL environment" on page 2.

Example

See "The SYSCALL environment" on page 2.
Chapter 6. BPXWDYN: a text interface to dynamic allocation and dynamic output

BPXWDYN is a text interface to a subset of the SVC 99 (dynamic allocation) and SVC 109 (dynamic output) services. BPXWDYN supports data set allocation, unallocation, concatenation, the retrieval of certain allocation information, and the addition and deletion of output descriptors. BPXWDYN is designed to be called from REXX, but it may be called from several other programming languages, including Assembler, C, and PL/I.

This interface makes dynamic allocation and dynamic output services easily accessible to programs running outside of a TSO environment; however, it also functions in a TSO environment.

The syntax for allocation is quite similar to that of TSO for the TSO ALLOCATE and FREE commands. It should be possible to provide parameters to BPXWDYN that would be acceptable as a TSO ALLOCATE or FREE command. However, there are keys supported by TSO ALLOCATE that are not currently supported by BPXWDYN. There are also some keys that can be used with BPXWDYN which are not compatible with TSO.

The syntax for accessing dynamic output facilities is similar to that of the TSO OUTDES command, but the name of the output descriptor is identified differently. You associate an output descriptor with a SYSOUT allocation by using the OUTDES key on the SYSOUT allocation request, or by creating a default output descriptor.

Calling conventions

BPXWDYN must be called in 31-bit mode in an environment that permits dynamic allocation and dynamic output requests. To call BPXWDYN from REXX or any other program, three parameter list forms can be used:

• REXX external function parameter list
• Conventional MVS variable-length parameter string
• Null-terminated parameter string

REXX external function parameter list

The external function parameter list allows a REXX program to call the BPXWDYN program as a function or subroutine. The BPXWDYN program must be called with a single string parameter. For example:

```plaintext
if BPXWDYN("alloc dd(sysin) da(my.dataset) shr")<>0 then
call allocfailed
```

Conventional MVS variable-length parameter string

This is the same parameter list as the one generated by ADDRESS LINKMVS with one parameter, and by JCL with EXEC PGM=, PARM=. Any program can easily use this parameter list form. Note that this is a single-item variable-length parameter list. The high bit is on in the parameter address word, and length is a halfword.
PL/I usage might include the following statements:

```
DCL PLIRETV BUILTIN;
DCL BPXWDYN EXTERNAL ENTRY OPTIONS(ASM INTER RETCODE);
DCL ALLOC_STR CHAR(100) VAR
  INIT('ALLOC FI(SYSIN) DA(MY.DATASET) SHR');
FETCH BPXWDYN;
CALL BPXWDYN(ALLOC_STR);
```

### Null-terminated parameter string

This parameter list is used most easily from C, passing a string to BPXWDYN defined with a `#pragma` for OS linkage, or `extern "OS"` for C++. Note that the high bit is on in the parameter address word.

```
typedef int EXTF();
#pragma linkage(EXTF,OS)
EXTF *bpxwdyn;
int rc;
bpxwdyn=(EXTF *)fetch("BPXWDYN");
rc=bpxwdyn("alloc fi(sysin) da(my.dataset) shr");
```

### Request types

The request is specified in a string parameter. The request type should be the first keyword in the parameter string. If a valid request type is not the first keyword, or no request type is specified, the request type defaults to ALLOC.

The following request types are supported:

- **ALLOC** Dynamic allocation of a data set
- **CONCAT** Dynamic concatenation of a ddname list
FREE  Dynamic unallocation of a ddname, or freeing of an output descriptor
INFO  Retrieval of allocation information
OUTDES Creation of an output descriptor

Keywords

The ordering of keywords on the request is arbitrary. You cannot specify the same keys multiple times for one request.

BPXWDYN does no consistency checking on keys. If some keys are not valid when combined together on a single request, dynamic allocation or dynamic output fails the request.

Some keywords accept arguments:

• Keyword arguments must be specified within parentheses. Spaces are not permitted between the key and the opening parenthesis, or anywhere within the additional argument string through the closing parenthesis, unless the argument string is quoted. Spaces are permitted between key specifications. Where multiple arguments are permitted, the arguments must be separated by commas.

• An argument may be enclosed within single (' ) or double (" ) quotes. Two adjacent quotes cannot be used to represent a single quote within a quoted string.

• Arguments that are not quoted are treated as uppercase. Arguments that are quoted are only treated as mixed case if it makes sense for the keyword.

• Arguments that accept REXX variable names will be accepted when BPXWDYN is called from any environment, but are only effective when called from a REXX environment.

Return codes

When BPXWDYN is called as a REXX function or subroutine, the return code can be accessed in RESULT or as the value of the function. When BPXWDYN is called as a program, the return code is available in R15.

BPXWDYN returns the following codes:

0  Success.
20  Invalid parameter list. See “Calling conventions” on page 247 for parameter list formats.
-21 to -9999  Key error. See “Key errors” for more information.
-100nn  Message processing error. IEFDB476 returned code nn.
>0, or -1610612737 to -2147483648  Dynamic allocation or dynamic output error codes. See “Error codes for dynamic allocation” on page 250 and “Error codes for dynamic output” on page 250 for more information.

Key errors

The low-order two digits are the number of the key that failed the parse, offset by 20. The first key has the value 21, resulting in a return code of -21.
The high-order two digits are an internal diagnostic code that indicates where in
the code the parse failed. If the high digits are 0, the key itself was probably not
recognized. Other values usually indicate a problem with the argument for the
failing keyword. Likely causes for failure are:

- Blanks in arguments that are not quoted
- Misspelling of keywords

**Error codes for dynamic allocation**

The return code contains the dynamic allocation error reason code in the high two
bytes, and the information reason code in the low two bytes.

You can use the high four hexadecimal digits to look up the error code in the
dynamic allocation error reason codes table found in the [z/OS MVS Programming:
Authorized Assembler Services Guide](https://www.ibm.com)

**Note:** For the returned decimal codes (>0, or -1610612737 to -2147483648), convert
them to hexadecimal codes before determining the value for the high and
low two bytes.

**Error codes for dynamic output**

Dynamic output errors do not produce messages. The return code contains:

- The dynamic output return code (S99ERROR) in the high two bytes
- The information code (S99INFO) in the low two bytes

You can use the high four hexadecimal digits to look up the error code in the
dynamic output return codes table found in the [z/OS MVS Programming:
Authorized Assembler Services Guide](https://www.ibm.com)

You can use the low four digits to look up the information code.

There is no indication of the key that is in error.

Some of the most common reason codes are:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>300-30C, 312, 380</td>
<td>The arguments may be specified incorrectly.</td>
</tr>
<tr>
<td>401</td>
<td>The output descriptor already exists.</td>
</tr>
<tr>
<td>402</td>
<td>The output descriptor does not exist.</td>
</tr>
<tr>
<td>403</td>
<td>Output descriptors created by JCL cannot be deleted by dynamic output.</td>
</tr>
</tbody>
</table>

**Message processing**

Dynamic allocation provides message text for failed allocation requests. It is
usually easier to use this message text rather than decode allocation return codes.

BPXWDYN can return these messages to a REXX program. By default, the
messages are returned in the S99MSG. stem. S99MSG.0 contains the number of
messages and S99MSG.1 through S99MSG.n contain the message text.

To change the name of the stem, use the MSG key. A stem is not required. Digits
are simply appended to the variable specified by the MSG key.

You can also use the MSG key to request allocation to issue the messages to your
job log (write to programmer using WTO).
Under z/OS UNIX, you can have messages written to opened files by providing the file descriptor number as the argument to the MSG key. For example, MSG(2) writes messages to the STDERR file.

**Requesting dynamic allocation**

To request dynamic allocation, specify `alloc` at the beginning of the parameter string.

BPXWDYN supports single data set allocation. Many of the common allocation keys can be used. For detailed information about those common allocation keys, see [z/OS TSO/E Command Reference](#). Some additional keys are supported to access additional functions.

BPXWDYN supports the following keys:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA(data set name[(member name)])</td>
<td>Specifies the data set name to allocate. The name must be fully qualified and may include a member name. Quotes can be used for TSO compatibility.</td>
</tr>
<tr>
<td>DUMMY</td>
<td>Allocates a dummy data set.</td>
</tr>
<tr>
<td>FI(name)</td>
<td>Specifies the ddname to allocate.</td>
</tr>
<tr>
<td>FILEDATA(TEXT</td>
<td>BINARY)</td>
</tr>
<tr>
<td>OLD</td>
<td>SHR</td>
</tr>
<tr>
<td>VOL(volser [,volser...])</td>
<td>Specifies the serial numbers for eligible direct access volumes where the data set is to reside.</td>
</tr>
<tr>
<td>DATACLAS(data class)</td>
<td>With SMS, specifies the data class for the data set.</td>
</tr>
<tr>
<td>MGMTCLAS(management class)</td>
<td>With SMS, specifies the management class for the data set.</td>
</tr>
<tr>
<td>STORCLAS(storage class)</td>
<td>With SMS, specifies the storage class for the data set.</td>
</tr>
<tr>
<td>SPACE(primary[,secondary])</td>
<td>Specifies primary and optionally secondary space allocations.</td>
</tr>
<tr>
<td>BLOCK(length)</td>
<td>Specifies block allocation with an average block or record size of <code>length</code>.</td>
</tr>
<tr>
<td>TRACKS</td>
<td>Specifies the unit of space in tracks.</td>
</tr>
<tr>
<td>CYL</td>
<td>Specifies the unit of space in cylinders.</td>
</tr>
<tr>
<td>BLKSIZE(block size)</td>
<td>Specifies the block size.</td>
</tr>
<tr>
<td>DIR(directory blocks)</td>
<td>Specifies the number of directory blocks.</td>
</tr>
<tr>
<td>DEST((destination</td>
<td>destination[.user]))</td>
</tr>
<tr>
<td>REUSE</td>
<td>Causes the named DD to be freed before the function is performed.</td>
</tr>
</tbody>
</table>
### Table 3. Common keys used for dynamic allocation (continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOLD</td>
<td>Specifies that the output data set is to be held until released by the user or operator.</td>
</tr>
<tr>
<td>UNIT(unit name)</td>
<td>Specifies unit name, device type, or unit address.</td>
</tr>
<tr>
<td>MAXVOL(num vols)</td>
<td>Number of volumes for a multi-volume data set.</td>
</tr>
<tr>
<td>KEEP</td>
<td>DELETE</td>
</tr>
<tr>
<td>BUFNO(number)</td>
<td>Sets the number of buffers. This number should be in the range 1–255. Numbers outside that range give unpredictable results.</td>
</tr>
<tr>
<td>LRECL(record length)</td>
<td>Specifies the logical record length.</td>
</tr>
<tr>
<td>RECFM(format[format...])</td>
<td>Specifies the record format. The valid values are A, B, D, F, M, S, T, U, and V. Several of these can be used in combination.</td>
</tr>
<tr>
<td>DSORG(PS</td>
<td>PO</td>
</tr>
<tr>
<td>COPIES(number of copies)</td>
<td>Specifies the number of copies to print.</td>
</tr>
<tr>
<td>FORMS(forms name)</td>
<td>Specifies the print form.</td>
</tr>
<tr>
<td>LIKE(model data set name)</td>
<td>Copies attributes for the allocation from the model data set.</td>
</tr>
<tr>
<td>OUTDES(output descriptor name)</td>
<td>Specifies the output descriptor name.</td>
</tr>
<tr>
<td>SPIN(UNALLOC)</td>
<td>Spins off a sysout data set at unallocation.</td>
</tr>
<tr>
<td>DSNTYPE(LIBRARY</td>
<td>PDS</td>
</tr>
<tr>
<td>WRITER(external writer name)</td>
<td>Names the external writer.</td>
</tr>
<tr>
<td>PATH(pathname)</td>
<td>Specifies the path name of an HFS file to allocate.</td>
</tr>
<tr>
<td>PATHDISP(KEEP</td>
<td>DELETE[,KEEP</td>
</tr>
<tr>
<td>PATHMODE(path mode list)</td>
<td>Set mode bits for a new allocation. This list is a list of keywords separated with commas. The supported keywords are: SIRUSR, SIWUSR, SIXUSR, SIRWXU, SIRGRP, SIWGRP, SIXGRP, SIRWXG, SIROTH, SIWOTH, SIXOTH, SIRWXO, SISUID, SISGID, SISVTX</td>
</tr>
</tbody>
</table>
Table 3. Common keys used for dynamic allocation (continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATHOPTS(path options list)</td>
<td>Sets options for path name allocation. The options list is a list of keywords separated with commas. The supported keywords are: ORDWR OEXCL OSYNC OTRUNC OCREAT OWRONLY ORDONLY OAPPEND ONOCTTY ONONBLOCK</td>
</tr>
<tr>
<td>RECORGLS</td>
<td>Creates a VSAM linear data set.</td>
</tr>
<tr>
<td>SEQUENCE(sequence number)</td>
<td>Specifies the relative position (number) of a data set on a tape volume.</td>
</tr>
<tr>
<td>LABEL(type)</td>
<td>Specifies the type of tape label processing to be done, as follows:</td>
</tr>
<tr>
<td></td>
<td>NL The volume has no label.</td>
</tr>
<tr>
<td></td>
<td>SL The volume has an IBM standard label.</td>
</tr>
<tr>
<td></td>
<td>NSL The volume has a non-standard label.</td>
</tr>
<tr>
<td></td>
<td>SUL The volume has both an IBM standard label and a user label.</td>
</tr>
<tr>
<td></td>
<td>BLP Bypass label processing for the volume.</td>
</tr>
<tr>
<td></td>
<td>LTM The system is to check for and bypass a leading tape mark on a DOS unlabeled tape.</td>
</tr>
<tr>
<td></td>
<td>AL The volume has an American National Standard label.</td>
</tr>
<tr>
<td></td>
<td>AUL The volume has both an American National Standard label and a user label.</td>
</tr>
<tr>
<td>RETPD(number of days)</td>
<td>Specifies the data set retention period, in days.</td>
</tr>
<tr>
<td>TRTCH(technique)</td>
<td>Specifies the tape recording technique, as follows:</td>
</tr>
<tr>
<td></td>
<td>NONCOMP Non-compaction mode</td>
</tr>
<tr>
<td></td>
<td>COMP Compaction mode</td>
</tr>
<tr>
<td></td>
<td>C Data conversion</td>
</tr>
<tr>
<td></td>
<td>E Even parity</td>
</tr>
<tr>
<td></td>
<td>ET Even parity and BCD/EBCDIC translation</td>
</tr>
<tr>
<td></td>
<td>T BCD/EBCDIC translation</td>
</tr>
<tr>
<td>SUBSYS(subsystem name[,subsys parm...])</td>
<td>Specifies that the data set is a subsystem data set, and also specifies the name of the subsystem and any parameters necessary for the subsystem to process the data set. Case and special characters are preserved for quoted parameters.</td>
</tr>
</tbody>
</table>

The following additional keys are unique to BPXWDYN.
### Table 4. Additional keys used for dynamic allocation

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG (WTP</td>
<td>default.S99MSG.</td>
</tr>
<tr>
<td>NORECALL</td>
<td>Fail the allocation request if the data set is migrated.</td>
</tr>
<tr>
<td>PATHPERM(octal path mode)</td>
<td>Set mode bits for a new allocation. This key is effectively the same as PATHMODE but accepts a simple octal number for the mode bit settings.</td>
</tr>
<tr>
<td>RTDDN(variable)</td>
<td>Return allocated ddname into the REXX variable <code>variable</code>.</td>
</tr>
<tr>
<td>RTDSN(variable)</td>
<td>Return allocated data set name into the REXX variable <code>variable</code>.</td>
</tr>
<tr>
<td>RTVOL(variable)</td>
<td>Return allocated volume name into the REXX variable <code>variable</code>.</td>
</tr>
<tr>
<td>SHORTRC</td>
<td>If the dynamic allocation fails, the dynamic allocation error code (S99ERROR) is returned in R15 and the information code is not returned. This is useful if the application can only process R15 as a halfword, such as with PL/I.</td>
</tr>
<tr>
<td>SYNTAX</td>
<td>When this is used, BPXWDYN parses the request and does not issue the dynamic allocation or dynamic output request.</td>
</tr>
<tr>
<td>RELEASE</td>
<td>Space allocated to an output data set but not used is released when the data set is closed.</td>
</tr>
<tr>
<td>UCOUNT(number)</td>
<td>Specifies the number of devices to be allocated.</td>
</tr>
<tr>
<td>GDGNT</td>
<td>The S99GDGNT flag is set in the S99FLAG1 field. For information about this flag, see <a href="https://www.ibm.com">z/OS MVS Programming: Authorized Assembler Services Guide</a>.</td>
</tr>
<tr>
<td>MOUNT</td>
<td>Resets the S99NOMNT flag, allowing volumes to be mounted.</td>
</tr>
</tbody>
</table>

### Requesting dynamic concatenation

To request dynamic concatenation, specify **concat** at the beginning of the parameter string.

This function concatenates multiple DDs to a single DD. The DDs are concatenated in the order specified in the DDLIST key.

### Table 5. Keys used for dynamic concatenation

| DDLIST(DDname1,DDname2[,DDname3...]) | Specifies a list of ddnames to concatenate to DDname1. Use this key with the CONCAT key. |
Table 5. Keys used for dynamic concatenation (continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG (WTP</td>
<td>default.S99MSG.</td>
</tr>
<tr>
<td>SHORTRC</td>
<td>If the dynamic allocation fails, the dynamic allocation error code (S99ERROR) is returned in R15 and the information code is not returned. This is useful if the application can only process R15 as a halfword, such as with PL/I.</td>
</tr>
</tbody>
</table>

**Requesting dynamic unallocation**

To request dynamic unallocation, specify `free` at the beginning of the parameter string.

BPXWDYN supports single data set allocation. Many of the common unallocation keys can be used. For detailed information about those common keys, see [z/OS TSO/E Command Reference](https://www.ibm.com). Some additional keys are supported to access additional functions.

Table 6. Common keys used for dynamic unallocation

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA(data set name([member name]))</td>
<td>DSN(data set name([member name]))</td>
</tr>
<tr>
<td>FI(name)</td>
<td>DD(name)</td>
</tr>
<tr>
<td>KEEP</td>
<td>DELETE</td>
</tr>
<tr>
<td>SPIN(UNALLOC)</td>
<td></td>
</tr>
<tr>
<td>SYSOUT(class)</td>
<td></td>
</tr>
</tbody>
</table>

The following additional keys are unique to BPXWDYN.

Table 7. Additional keys used for dynamic unallocation

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG (WTP</td>
<td>default.S99MSG.</td>
</tr>
<tr>
<td>SHORTRC</td>
<td>If the dynamic allocation fails, the dynamic allocation error code (S99ERROR) is returned in R15 and the information code is not returned. This is useful if the application can only process R15 as a halfword, such as with PL/I.</td>
</tr>
</tbody>
</table>
Requesting allocation information

To request information about the current dynamic allocation environment, specify info at the beginning of the parameter string.

BPXWDYN currently supports the retrieval of the ddnames, data set names, and path names for current allocations; other allocation attributes are not supported.

BPXWDYN supports single data set allocation. You can use specific keys to specify which allocation requests information is to be returned for.

Table 8. Keys used to specify which allocation request information is to be returned for

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA</td>
<td>Specifies the data set name of the allocated resource about which you are requesting information. The data set name can contain special characters if the data set name is enclosed in apostrophes. It can also contain system symbols (see the section on using system symbols in z/OS MVS JCL Reference for more information). The maximum length of the data set name is 44 characters, excluding any enclosing apostrophes and compressing any double apostrophes within the data set name. This key is mutually exclusive with the ddname (FI, DD), path name (PATH), and relative entry (INRELNO) keys.</td>
</tr>
<tr>
<td>FI</td>
<td>Specifies the ddname of the allocated resource about which you are requesting information. name contains the ddname. This key is mutually exclusive with the data set name (DA, DSN), path name (PATH), and relative entry (INRELNO) keys.</td>
</tr>
<tr>
<td>PATH</td>
<td>Specifies the path name of the HFS file for which you are requesting information. pathname contains the path name and can contain system symbols (see the section on using system symbols in z/OS MVS JCL Reference for more information). This key is mutually exclusive with the data set name (DA, DSN), ddname (FI, DD), and relative entry (INRELNO) keys.</td>
</tr>
</tbody>
</table>

The following additional keys are unique to BPXWDYN.

Table 9. Additional keys used for dynamic information retrieval

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG</td>
<td>Directs messages to your job log (WTP), a REXX stem, or a file identified by a file descriptor number. If this key is not specified, messages will be returned in the S99MSG. stem, if possible. If BPXWDYN was not called from a REXX environment, the messages will be lost. WTP should be specified to obtain messages.</td>
</tr>
</tbody>
</table>
### Table 9. Additional keys used for dynamic information retrieval (continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INRELNO(request number)</td>
<td>Specifies the relative request number of the allocation for which you are requesting information. <em>request number</em> contains the relative request number. This key is mutually exclusive with the data set name (DA, DSN), ddname (FI, DD), and path name (PATH) keys.</td>
</tr>
<tr>
<td>INRTLST(variable)</td>
<td>Requests an indication of whether the relative entry request number, ddname, or data set name you specify is the last relative entry. Upon return to your program, the REXX variable <em>variable</em> contains one of the following values:</td>
</tr>
<tr>
<td></td>
<td>128 Last relative entry</td>
</tr>
<tr>
<td></td>
<td>0 Not the last relative entry</td>
</tr>
<tr>
<td>INRTDDN(variable)</td>
<td>Requests the ddname associated with the specified allocation. Upon return to your program, the REXX variable <em>variable</em> contains the requested ddname. If the data set you specify is a member of a concatenated group and is not the first member, there is no ddname associated with it.</td>
</tr>
<tr>
<td>INRTDSN(variable)</td>
<td>Requests the data set name of the specified allocation. Upon return to your program, the REXX variable <em>variable</em> contains the requested data set name.</td>
</tr>
<tr>
<td>INRTPATH(variable)</td>
<td>Requests the path name for the z/OS UNIX file associated with the specified allocation. Upon return to your program, the REXX variable <em>variable</em> contains the path name, if any, associated with the allocation.</td>
</tr>
</tbody>
</table>

### Requesting dynamic output

To request dynamic output, specify **outdes** at the beginning of the parameter string. This keyword takes an argument that names the output descriptor. The keyword is described in the following table. These are the supported keys:

### Table 10. Keys used for dynamic output

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTDES(descriptor name)</td>
<td>Names the output descriptor to be added. This must be the first key specified in the parameter string.</td>
</tr>
<tr>
<td>ADDRESS(address[,address...])</td>
<td>Specifies the delivery address. Dynamic output allows up to four arguments to be specified.</td>
</tr>
<tr>
<td>BUILDING(building)</td>
<td>Specifies the building location.</td>
</tr>
<tr>
<td>BURST</td>
<td>Directs output to a stacker.</td>
</tr>
<tr>
<td>CHAR$S(chars[,chars...])</td>
<td>Names the character arrangement tables.</td>
</tr>
<tr>
<td>CLASS(class)</td>
<td>Assigns the SYSOUT class.</td>
</tr>
</tbody>
</table>
### Table 10. Keys used for dynamic output (continued)

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONTROL(spacing)</strong></td>
<td>Specifies either line spacing or that the records begin with carriage control characters. The valid values are:</td>
</tr>
<tr>
<td></td>
<td>Single Single spacing</td>
</tr>
<tr>
<td></td>
<td>Double Double spacing</td>
</tr>
<tr>
<td></td>
<td>Triple Triple spacing</td>
</tr>
<tr>
<td></td>
<td>Program Records begin with carriage control characters</td>
</tr>
<tr>
<td><strong>COPIES(number of copies)</strong></td>
<td>Specifies the number of copies to be printed.</td>
</tr>
<tr>
<td><strong>DEFAULT</strong></td>
<td>Specifies that this is a default output descriptor.</td>
</tr>
<tr>
<td><strong>DEPT(department)</strong></td>
<td>Specifies the department identification.</td>
</tr>
<tr>
<td><strong>DEST(node[,user])</strong></td>
<td>Sends the SYSOUT to the specified destination.</td>
</tr>
<tr>
<td><strong>DPAGELBL</strong></td>
<td>Specifies that a security label be placed on the output.</td>
</tr>
<tr>
<td><strong>FCB(fcb name)</strong></td>
<td>Specifies the FCB image.</td>
</tr>
<tr>
<td><strong>FLASH</strong></td>
<td>Specifies the forms overlay.</td>
</tr>
<tr>
<td><strong>FORMDEF(formdef name)</strong></td>
<td>Names the formdef.</td>
</tr>
<tr>
<td><strong>FORMS(forms name)</strong></td>
<td>Names the forms to print on.</td>
</tr>
<tr>
<td><strong>MODIFY(trc number)</strong></td>
<td>Specifies which character arrangement table is to be used.</td>
</tr>
<tr>
<td><strong>NAME(owner name)</strong></td>
<td>Specifies the owner's name.</td>
</tr>
<tr>
<td><strong>NOTIFY([node[,user]])</strong></td>
<td>Sends print completion message to the destination.</td>
</tr>
<tr>
<td><strong>OUTDISP(disposition)</strong></td>
<td>Specifies the data set disposition.</td>
</tr>
<tr>
<td><strong>PAGEDEF(pagedef name)</strong></td>
<td>Names the pagedef.</td>
</tr>
<tr>
<td><strong>PRMODE(print mode)</strong></td>
<td>Identifies the process mode (LINE or PAGE).</td>
</tr>
<tr>
<td><strong>ROOM(room identification)</strong></td>
<td>Specifies the room identification.</td>
</tr>
<tr>
<td><strong>TITLE(separator title)</strong></td>
<td>Specifies the separator page title.</td>
</tr>
<tr>
<td><strong>TRC</strong></td>
<td>Specifies that the data set contains TRC codes.</td>
</tr>
<tr>
<td><strong>UCS(UCS name)</strong></td>
<td>Names the UCS or character arrangement table.</td>
</tr>
<tr>
<td><strong>WRITER(external writer name)</strong></td>
<td>Names an external writer to process the data set.</td>
</tr>
<tr>
<td><strong>USERDATA(userdata[,userdata...])</strong></td>
<td>Specifies installation-specific user data for a dynamic output statement. Dynamic output allows up to 16 1–60-character strings. Case is preserved if the string is quoted.</td>
</tr>
</tbody>
</table>
Freeing an output descriptor

To request that an output descriptor be freed, specify **free** at the beginning of the parameter string. Only one keyword is supported:

<table>
<thead>
<tr>
<th>Table 11. Key used to free an output descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTDES(descriptor name)</td>
</tr>
</tbody>
</table>

Examples: Calling BPXWDYN from a REXX program

Allocate

This example allocates SYS1.MACLIB to SYSLIB and directs messages to z/OS UNIX standard error (stderr):

call bpxwdyn "alloc fi(syslib) da(sys1.maclib) shr msg(2)"

Info

This example requests that the name of the data set allocated to ddname SYSLIB be returned in the REXX variable **dsnvar**.

call bpxwdyn "info fi(syslib) inrtdsn(dsnvar)"

Free

This example frees SYSLIB and traps messages in stem S99MSG.:

call bpxwdyn "free fi(syslib)"

Concatenate

This example concatenates SYS1.SBPXEXEC to SYSPROC:

```plaintext
if bpxwdyn("alloc fi(tmp) da(sys1.sbpexec) shr msg(2)")=0 then
call bpxwdyn "concat ddlist(sysproc,tmp) msg(2)"
```

Create dynamic output descriptor

This example creates descriptor P20 with distribution information:

call bpxwdyn "outdes(p20) dest(kgn.p20n10)
\"address(\"WJ Schoen\",\'M/S 619\")
\"name(wschoen) bin(0004) dept(64ba)"

Free descriptor

This example frees descriptor P20:

call bpxwdyn "free outdes(p20)"

Example: calling BPXWDYN from C

Allocate

This example allocates SYS1.MACLIB to SYSLIB and directs messages to z/OS UNIX standard error (stderr):

```c
typedef int EXTF();
#pragma linkage(EXTF,OS_UPSTACK)

int call_alloc()
{
    EXTF *bpxwdyn=(EXTF *)fetch("BPXWDYN");
    return bpxwdyn("alloc fi(syslib) da(sys1.maclib) shr msg(2)")
}
```
Chapter 7. 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 z/OS UNIX REXX syscall commands. Because 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 z/OS UNIX System Services 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 8.

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 across z/OS UNIX re-IPLs.

You must specify tokens as variable names, not as strings. See “Specifying a syscall command” on page 19 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.
**Function**

The `v_create` callable service is used 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 83.
- **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 "fstat" on page 178 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 126.
- **minor**: For a character special file (S_ISCHR), the device minor number. For a complete description, see "mknod" on page 126.

**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 filem and vnod were assigned values earlier in the exec:
"v_create vnod (filem)" s_isreg 777 "st. filetok"
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 181 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."
```
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 z/OS UNIX re-IPLs. 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```

Function

_vgetattr_ invokes the _vgetattr_ 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 178 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:

"_vgetattr vnod attr.\""
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"
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VL_REGLOCKER</td>
<td>Register the lock server (locker).</td>
</tr>
<tr>
<td>VL_UNREGLOCKER</td>
<td>Unregister the locker.</td>
</tr>
<tr>
<td>VL_LOCK</td>
<td>Set a lock in a specified byte range.</td>
</tr>
<tr>
<td>VL_LOCKWAIT</td>
<td>Set a lock in a specified byte range or wait to set the lock until the byte range is free.</td>
</tr>
<tr>
<td>VL_UNLOCK</td>
<td>Unlock all locks in a specified byte range.</td>
</tr>
<tr>
<td>VL_QUERY</td>
<td>Query for lock information about a file.</td>
</tr>
<tr>
<td>VL_PURGE</td>
<td>Release all locks on all files, held by a locker or a group of lockers.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VL_SERVERPID</td>
<td>Server's PID</td>
</tr>
<tr>
<td>VL_CLIENTPID</td>
<td>Server's client's process ID (PID)</td>
</tr>
<tr>
<td>VL_LOCKERTOK</td>
<td>Token for locker</td>
</tr>
<tr>
<td>VL_CLIENTTID</td>
<td>Client's thread's TID. The TID is the individual lock owner within a locker.</td>
</tr>
<tr>
<td>VL_OBJCLASS</td>
<td>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.</td>
</tr>
</tbody>
</table>
v_lockctl

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VL_OBJID</td>
<td>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.</td>
</tr>
<tr>
<td>VL_OBJTOK</td>
<td>A token to identify the object (locked file) on a subsequent lock request.</td>
</tr>
<tr>
<td>VL_DOSMODE</td>
<td>DOS file-sharing field</td>
</tr>
<tr>
<td>VL_DOSACCESS</td>
<td>DOS file-sharing field</td>
</tr>
</tbody>
</table>

For a description of the variables beginning with L_, see "f_getlk" on page 68.

**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 68.
   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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_RDLCK</td>
<td>A read lock, also known as a shared lock. This...</td>
</tr>
</tbody>
</table>
**v_lockctl**

A 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:

```plaintext
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:

```plaintext
/* 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 */
```
v_lockctl

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
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 178 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 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

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 178 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 that is 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"
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 178 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 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"
Function

\texttt{v\_readlink} invokes the \texttt{v\_readlink} callable service to read the symbolic link file represented by the vnode token and return its contents in \texttt{variable}.

Parameters

\texttt{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, \texttt{v\_getattr}) identifies whether the symbolic link is a link to an external name in the stem index \texttt{ST\_EXTLINK}. An external name is the name of an object outside the HFS.

\texttt{variable}

The name of the buffer that, on return, contains the contents of the symbolic link.

Example

In the following example, assume that \texttt{symtok} was assigned a value earlier in the exec:

\texttt{"v\_readlink symtok link"}
**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"
```
Function

\texttt{v\_rel} invokes the \texttt{v\_rel} callable service to accept a vnode token representing a file descriptor for a file or directory, and to release that token.

Parameters

\texttt{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 \texttt{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 \texttt{vntok} was assigned a value earlier in the exec:

\begin{verbatim}
"v\_rel vntok"
\end{verbatim}
v_remove

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 282.

Example

In the following example, assume that dirtok and file were assigned values earlier
in the exec:

"v_remove dirtok (file)"
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_remove

Function

v_remove invokes the v_remove 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_remove dirtok (dirname)"
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 99 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 178 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

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 38 for the numeric values, see Appendix A.

Usage notes

For usage notes, see "chattr" on page 38.

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
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.

directory name
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 267), 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 z/OS UNIX 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)"
Function

_v_r_dwr_ invokes the v_r_dwr 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 178 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 8. Examples: Using 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
/* 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
/* rexx */
if __argv.0<>3 then /* check for right number of args */
  do
    say 'directory and filename required'
    return
  end

i=__argv.1 /* _ _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. 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 format
- **Char**  Character
- **Dec**  Decimal
- **Hex**  Hexadecimal: 4-byte hexadecimal format
- **Tok**  Token
- **Oct**  Octal

The predefined variables listed alphabetically are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUD_FEXEC</td>
<td>Dec</td>
<td>512</td>
</tr>
<tr>
<td>AUD_FREAD</td>
<td>Dec</td>
<td>33554432</td>
</tr>
<tr>
<td>AUD_FWRITE</td>
<td>Dec</td>
<td>131072</td>
</tr>
<tr>
<td>AUD_SEXEC</td>
<td>Dec</td>
<td>256</td>
</tr>
<tr>
<td>AUD_SREAD</td>
<td>Dec</td>
<td>16777216</td>
</tr>
<tr>
<td>AUD_SWRITE</td>
<td>Dec</td>
<td>65536</td>
</tr>
<tr>
<td>E2BIG</td>
<td>Hex</td>
<td>91</td>
</tr>
<tr>
<td>EACCES</td>
<td>Hex</td>
<td>6F</td>
</tr>
<tr>
<td>EAGAIN</td>
<td>Hex</td>
<td>70</td>
</tr>
<tr>
<td>EBADF</td>
<td>Hex</td>
<td>71</td>
</tr>
<tr>
<td>EBUSY</td>
<td>Hex</td>
<td>72</td>
</tr>
<tr>
<td>ECHILD</td>
<td>Hex</td>
<td>73</td>
</tr>
<tr>
<td>EDEADLK</td>
<td>Hex</td>
<td>74</td>
</tr>
<tr>
<td>EDOM</td>
<td>Hex</td>
<td>01</td>
</tr>
<tr>
<td>EEXIST</td>
<td>Hex</td>
<td>75</td>
</tr>
<tr>
<td>EFAULT</td>
<td>Hex</td>
<td>76</td>
</tr>
<tr>
<td>EFBIG</td>
<td>Hex</td>
<td>77</td>
</tr>
<tr>
<td>EILSEQ</td>
<td>Hex</td>
<td>93</td>
</tr>
<tr>
<td>EINTR</td>
<td>Hex</td>
<td>78</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Hex</td>
<td>79</td>
</tr>
<tr>
<td>Variable</td>
<td>Data Type</td>
<td>Numeric Value</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>EIO</td>
<td>Hex</td>
<td>7A</td>
</tr>
<tr>
<td>EISDIR</td>
<td>Hex</td>
<td>7B</td>
</tr>
<tr>
<td>ELOOP</td>
<td>Hex</td>
<td>92</td>
</tr>
<tr>
<td>EMFILE</td>
<td>Hex</td>
<td>7C</td>
</tr>
<tr>
<td>EMLINK</td>
<td>Hex</td>
<td>7D</td>
</tr>
<tr>
<td>EMVSBADCHAR</td>
<td>Hex</td>
<td>A0</td>
</tr>
<tr>
<td>EMVSCATLG</td>
<td>Hex</td>
<td>99</td>
</tr>
<tr>
<td>EMVSCVAF</td>
<td>Hex</td>
<td>98</td>
</tr>
<tr>
<td>EMVSDYNALC</td>
<td>Hex</td>
<td>97</td>
</tr>
<tr>
<td>EMVSERR</td>
<td>Hex</td>
<td>9D</td>
</tr>
<tr>
<td>EMVSINITIAL</td>
<td>Hex</td>
<td>9C</td>
</tr>
<tr>
<td>EMVSNORTL</td>
<td>Hex</td>
<td>A7</td>
</tr>
<tr>
<td>EMVSNOTUP</td>
<td>Hex</td>
<td>96</td>
</tr>
<tr>
<td>EMVSPARM</td>
<td>Hex</td>
<td>9E</td>
</tr>
<tr>
<td>EMVSPATHOPTS</td>
<td>Hex</td>
<td>A6</td>
</tr>
<tr>
<td>EMVSPFSFILE</td>
<td>Hex</td>
<td>9F</td>
</tr>
<tr>
<td>EMVSPFSPERM</td>
<td>Hex</td>
<td>A2</td>
</tr>
<tr>
<td>EMVSSAF2ERR</td>
<td>Hex</td>
<td>A4</td>
</tr>
<tr>
<td>EMVSSAFEXTRERR</td>
<td>Hex</td>
<td>A3</td>
</tr>
<tr>
<td>EMVSTODNOTSET</td>
<td>Hex</td>
<td>A5</td>
</tr>
<tr>
<td>ENAMETOOLONG</td>
<td>Hex</td>
<td>7E</td>
</tr>
<tr>
<td>ENFILE</td>
<td>Hex</td>
<td>7F</td>
</tr>
<tr>
<td>ENODEV</td>
<td>Hex</td>
<td>80</td>
</tr>
<tr>
<td>ENOENT</td>
<td>Hex</td>
<td>81</td>
</tr>
<tr>
<td>ENOEXEC</td>
<td>Hex</td>
<td>82</td>
</tr>
<tr>
<td>ENOLCK</td>
<td>Hex</td>
<td>83</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Hex</td>
<td>84</td>
</tr>
<tr>
<td>ENOSPC</td>
<td>Hex</td>
<td>85</td>
</tr>
<tr>
<td>ENOSYS</td>
<td>Hex</td>
<td>86</td>
</tr>
<tr>
<td>ENOTDIR</td>
<td>Hex</td>
<td>87</td>
</tr>
<tr>
<td>ENOTEMPTY</td>
<td>Hex</td>
<td>88</td>
</tr>
<tr>
<td>ENOTTY</td>
<td>Hex</td>
<td>89</td>
</tr>
<tr>
<td>ENXIO</td>
<td>Hex</td>
<td>8A</td>
</tr>
<tr>
<td>EPERM</td>
<td>Hex</td>
<td>8B</td>
</tr>
<tr>
<td>EPIPE</td>
<td>Hex</td>
<td>8C</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Hex</td>
<td>02</td>
</tr>
<tr>
<td>EROFS</td>
<td>Hex</td>
<td>8D</td>
</tr>
<tr>
<td>ESC_A</td>
<td>Hex</td>
<td>2F</td>
</tr>
<tr>
<td>ESC_B</td>
<td>Hex</td>
<td>16</td>
</tr>
<tr>
<td>ESC_F</td>
<td>Hex</td>
<td>0C</td>
</tr>
</tbody>
</table>
Table 12. List of predefined variables (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC_N</td>
<td>Hex</td>
<td>15</td>
</tr>
<tr>
<td>ESC_R</td>
<td>Hex</td>
<td>0D</td>
</tr>
<tr>
<td>ESC_T</td>
<td>Hex</td>
<td>05</td>
</tr>
<tr>
<td>ESC_V</td>
<td>Hex</td>
<td>0B</td>
</tr>
<tr>
<td>ESPIPE</td>
<td>Hex</td>
<td>8E</td>
</tr>
<tr>
<td>ESRCB</td>
<td>Hex</td>
<td>8F</td>
</tr>
<tr>
<td>EXDEV</td>
<td>Hex</td>
<td>90</td>
</tr>
<tr>
<td>F_OK</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>F_RDLCK</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>F_UNLCK</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>F_WRLCK</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>GR_GID</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>GR_MEM</td>
<td>Char</td>
<td>4</td>
</tr>
<tr>
<td>GR_MEMBERS</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>GR_NAME</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>L_LEN</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>L_PID</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>L_START</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>L_TYPE</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>L_WHENCE</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>MNT_ASYNCHMOUNT</td>
<td>Dec</td>
<td>130</td>
</tr>
<tr>
<td>MNT_FILEACTIVE</td>
<td>Dec</td>
<td>0</td>
</tr>
<tr>
<td>MNT_FILEDEAD</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>MNT_FILEDRAIN</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>MNT_FILEFORCE</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>MNT_FILEIMMED</td>
<td>Dec</td>
<td>16</td>
</tr>
<tr>
<td>MNT_FILENORM</td>
<td>Dec</td>
<td>32</td>
</tr>
<tr>
<td>MNT_FILERESET</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>MNT_IMMEDTRIED</td>
<td>Dec</td>
<td>64</td>
</tr>
<tr>
<td>MNT_MODE_AUNMOUNT</td>
<td>Dec</td>
<td>64</td>
</tr>
<tr>
<td>MNT_MODE_CLIENT</td>
<td>Dec</td>
<td>32</td>
</tr>
<tr>
<td>MNT_MODE_EXPORT</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>MNT_MODE_NOAUTOMOVE</td>
<td>Dec</td>
<td>16</td>
</tr>
<tr>
<td>MNT_MODE_NOSEC</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>MNT_MODE_NOSETID</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>MNT_MODE_RDONLY</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>MNT_MODE_RDWR</td>
<td>Dec</td>
<td>0</td>
</tr>
<tr>
<td>MNT_MODE_SECACT</td>
<td>Dec</td>
<td>128</td>
</tr>
<tr>
<td>MNT_MOUNTINPROGRESS</td>
<td>Dec</td>
<td>129</td>
</tr>
<tr>
<td>MNT_QUIESCED</td>
<td>Dec</td>
<td>128</td>
</tr>
</tbody>
</table>
Table 12. List of predefined variables  (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNTE_AGGNAME</td>
<td>Char</td>
<td>30</td>
</tr>
<tr>
<td>MNTE_BYTESREADHW</td>
<td>Dec</td>
<td>25</td>
</tr>
<tr>
<td>MNTE_BYTESREADLW</td>
<td>Dec</td>
<td>26</td>
</tr>
<tr>
<td>MNTE_BYTESWRITTENHW</td>
<td>Dec</td>
<td>27</td>
</tr>
<tr>
<td>MNTE_BYTESWRITTENLW</td>
<td>Dec</td>
<td>28</td>
</tr>
<tr>
<td>MNTE_DD</td>
<td>Char</td>
<td>4</td>
</tr>
<tr>
<td>MNTE_DEV</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>MNTE_DIRIBC</td>
<td>Dec</td>
<td>22</td>
</tr>
<tr>
<td>MNTE_FILETAG</td>
<td>Char</td>
<td>29</td>
</tr>
<tr>
<td>MNTE_FROMSYS</td>
<td>Char</td>
<td>18</td>
</tr>
<tr>
<td>MNTE_FSNAME</td>
<td>Char</td>
<td>6</td>
</tr>
<tr>
<td>MNTE_FSTYPE</td>
<td>Char</td>
<td>5</td>
</tr>
<tr>
<td>MNTE_MODE</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>MNTE_PARDEV</td>
<td>Dec</td>
<td>9</td>
</tr>
<tr>
<td>MNTE_PARM</td>
<td>Char</td>
<td>13</td>
</tr>
<tr>
<td>MNTE_PATH</td>
<td>Char</td>
<td>7</td>
</tr>
<tr>
<td>MNTE_PFSSTATEXCP</td>
<td>Char</td>
<td>34</td>
</tr>
<tr>
<td>MNTE_PFSSTATUSNORMAL</td>
<td>Char</td>
<td>33</td>
</tr>
<tr>
<td>MNTE_QJOBNAME</td>
<td>Char</td>
<td>11</td>
</tr>
<tr>
<td>MNTE_QPID</td>
<td>Dec</td>
<td>12</td>
</tr>
<tr>
<td>MNTE_QSYSNAME</td>
<td>Char</td>
<td>16</td>
</tr>
<tr>
<td>MNTE_READCT</td>
<td>Dec</td>
<td>20</td>
</tr>
<tr>
<td>MNTE_READIBC</td>
<td>Dec</td>
<td>23</td>
</tr>
<tr>
<td>MNTE_RFLAGS</td>
<td>Dec</td>
<td>14</td>
</tr>
<tr>
<td>MNTE_ROOTINO</td>
<td>Dec</td>
<td>10</td>
</tr>
<tr>
<td>MNTE_ROSECLABEL</td>
<td>Char</td>
<td>32</td>
</tr>
<tr>
<td>MNTE_STATUS</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>MNTE_STATUS2</td>
<td>Dec</td>
<td>15</td>
</tr>
<tr>
<td>MNTE_SUCCESS</td>
<td>Dec</td>
<td>19</td>
</tr>
<tr>
<td>MNTE_SYSLIST</td>
<td>Char</td>
<td>31</td>
</tr>
<tr>
<td>MNTE_SYSNAME</td>
<td>Char</td>
<td>17</td>
</tr>
<tr>
<td>MNTE_TYPE</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>MNTE_WRITECT</td>
<td>Dec</td>
<td>21</td>
</tr>
<tr>
<td>MNTE_WRITEIBC</td>
<td>Dec</td>
<td>24</td>
</tr>
<tr>
<td>MTM_DRAIN</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>MTM_FORCE</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>MTM_IMMED</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>MTM_NORMAL</td>
<td>Dec</td>
<td>16</td>
</tr>
<tr>
<td>MTM_NOSUID</td>
<td>Dec</td>
<td>262144</td>
</tr>
<tr>
<td>MTM_RDONLY</td>
<td>Dec</td>
<td>128</td>
</tr>
</tbody>
</table>
Table 12. List of predefined variables (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTM_RDWR</td>
<td>Dec</td>
<td>64</td>
</tr>
<tr>
<td>MTM_REMOUNT</td>
<td>Dec</td>
<td>65536</td>
</tr>
<tr>
<td>MTM_RESET</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>MTM_SYNCHONLY</td>
<td>Dec</td>
<td>131072</td>
</tr>
<tr>
<td>O_APPEND</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>O_CREAT</td>
<td>Dec</td>
<td>128</td>
</tr>
<tr>
<td>O_EXCL</td>
<td>Dec</td>
<td>64</td>
</tr>
<tr>
<td>O_NOCITY</td>
<td>Dec</td>
<td>32</td>
</tr>
<tr>
<td>O_NONBLOCK</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>O_RDONLY</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>O_SYNC</td>
<td>Dec</td>
<td>256</td>
</tr>
<tr>
<td>O_TRUNC</td>
<td>Dec</td>
<td>16</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>PC_ACL</td>
<td>Dec</td>
<td>10</td>
</tr>
<tr>
<td>PC_ACL_MAX</td>
<td>Dec</td>
<td>11</td>
</tr>
<tr>
<td>PC_LINK_MAX</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>PC_MAX_CANON</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>PC_MAX_INPUT</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>PC_NAME_MAX</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>PC_PATH_MAX</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>PC_PIPE_BUF</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>PC_POSIX_CHOWN_RESTRICTED</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>PC_POSIX_NO_TRUNC</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>PC_POSIX_VDISABLE</td>
<td>Dec</td>
<td>9</td>
</tr>
<tr>
<td>PS_CHILD</td>
<td>Char</td>
<td>W</td>
</tr>
<tr>
<td>PS.Cmd</td>
<td>Dec</td>
<td>19</td>
</tr>
<tr>
<td>PS_CONTTY</td>
<td>Dec</td>
<td>17</td>
</tr>
<tr>
<td>PS_EGID</td>
<td>Dec</td>
<td>10</td>
</tr>
<tr>
<td>PS_EUID</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>PS_FGPID</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>PS_FORK</td>
<td>Char</td>
<td>X</td>
</tr>
<tr>
<td>PS_FREEZE</td>
<td>Char</td>
<td>E</td>
</tr>
<tr>
<td>PS_MAXVNODES</td>
<td>Dec</td>
<td>25</td>
</tr>
<tr>
<td>PS_MSGRCV</td>
<td>Char</td>
<td>A</td>
</tr>
<tr>
<td>PS_MSGSND</td>
<td>Char</td>
<td>B</td>
</tr>
<tr>
<td>PS_PATH</td>
<td>Char</td>
<td>18</td>
</tr>
<tr>
<td>PS_PAUSE</td>
<td>Char</td>
<td>G</td>
</tr>
<tr>
<td>PS_PGPID</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>PS_PID</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>Variable</td>
<td>Data Type</td>
<td>Numeric Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>PS_PPID</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>PS QUIESCE</td>
<td>Char</td>
<td>Q</td>
</tr>
<tr>
<td>PS_RGID</td>
<td>Dec</td>
<td>11</td>
</tr>
<tr>
<td>PS_RUID</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>PS_RUN</td>
<td>Char</td>
<td>R</td>
</tr>
<tr>
<td>PS SEMWT</td>
<td>Char</td>
<td>D</td>
</tr>
<tr>
<td>PS_SERVERFLAGS</td>
<td>Dec</td>
<td>27</td>
</tr>
<tr>
<td>PS_SERVERNAME</td>
<td>Dec</td>
<td>22</td>
</tr>
<tr>
<td>PS_SERVERTYPE</td>
<td>Dec</td>
<td>21</td>
</tr>
<tr>
<td>PS SGID</td>
<td>Dec</td>
<td>12</td>
</tr>
<tr>
<td>PS SID</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>PS SIZE</td>
<td>Dec</td>
<td>13</td>
</tr>
<tr>
<td>PS SLEEP</td>
<td>Char</td>
<td>S</td>
</tr>
<tr>
<td>PS STARTTIME</td>
<td>Dec</td>
<td>14</td>
</tr>
<tr>
<td>PS STAT</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>PS STATE</td>
<td>Char</td>
<td>20</td>
</tr>
<tr>
<td>PS SUID</td>
<td>Dec</td>
<td>9</td>
</tr>
<tr>
<td>PS SYSTIME</td>
<td>Dec</td>
<td>16</td>
</tr>
<tr>
<td>PS USERTIME</td>
<td>Dec</td>
<td>15</td>
</tr>
<tr>
<td>PS VNODECOUNT</td>
<td>Dec</td>
<td>26</td>
</tr>
<tr>
<td>PS_WAITC</td>
<td>Char</td>
<td>C</td>
</tr>
<tr>
<td>PS_WAITF</td>
<td>Char</td>
<td>F</td>
</tr>
<tr>
<td>PS WAITO</td>
<td>Char</td>
<td>K</td>
</tr>
<tr>
<td>PS ZOMBIE</td>
<td>Char</td>
<td>Z</td>
</tr>
<tr>
<td>PS ZOMBIE2</td>
<td>Char</td>
<td>L</td>
</tr>
<tr>
<td>PW_DIR</td>
<td>Char</td>
<td>4</td>
</tr>
<tr>
<td>PW GID</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>PW NAME</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>PW SHELL</td>
<td>Char</td>
<td>5</td>
</tr>
<tr>
<td>PW UID</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>R_OK</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>RLIMIT AS</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>RLIMIT_CORE</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>RLIMIT CPU</td>
<td>Dec</td>
<td>0</td>
</tr>
<tr>
<td>RLIMITFSIZE</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>RLIMIT_INFINITY</td>
<td>Dec</td>
<td>2147483647</td>
</tr>
<tr>
<td>RLIMIT_NOFILE</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>S FFBINARY</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>S FFCR</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>S FFCRRLF</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>Variable</td>
<td>Data Type</td>
<td>Numeric Value</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>S_FCRNL</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>S_FLF</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>S_FLFACR</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>S_FFNA</td>
<td>Dec</td>
<td>0</td>
</tr>
<tr>
<td>S_FFN</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>S_FFRECORD</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>S_ISCHR</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>S_ISDIR</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>S_ISFIFO</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>S_ISREG</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>S_ISSYM</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>SA_NOCLDSTOP</td>
<td>Dec</td>
<td>32768</td>
</tr>
<tr>
<td>SA_NOCLDWAIT</td>
<td>Dec</td>
<td>512</td>
</tr>
<tr>
<td>SA_NORESETHAND</td>
<td>Dec</td>
<td>4096</td>
</tr>
<tr>
<td>SC_2_CHAR_TERM</td>
<td>Dec</td>
<td>12</td>
</tr>
<tr>
<td>SC_ARG_MAX</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>SC_CHILD_MAX</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>SC_CLK_TCK</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>SC_JOB_CONTROL</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>SC_NGROUPS_MAX</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>SC_OPEN_MAX</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>SC_SAVED_IDS</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>SC_TREAD_TASKS_MAX_NPC</td>
<td>Dec</td>
<td>11</td>
</tr>
<tr>
<td>SC_THREADS_MAP_NPC</td>
<td>Dec</td>
<td>13</td>
</tr>
<tr>
<td>SC_TZNAME_MAX</td>
<td>Dec</td>
<td>9</td>
</tr>
<tr>
<td>SC_VERSION</td>
<td>Dec</td>
<td>10</td>
</tr>
<tr>
<td>SE_ERRNO</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>SE_REASON</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>SE_ACTION</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>SE_MODID</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>SEEK_CUR</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>SEEK_END</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>SEEK_SET</td>
<td>Dec</td>
<td>0</td>
</tr>
<tr>
<td>SIG_BLOCK</td>
<td>Dec</td>
<td>0</td>
</tr>
<tr>
<td>SIG_CAT</td>
<td>Dec</td>
<td>10</td>
</tr>
<tr>
<td>SIG_DFL</td>
<td>Dec</td>
<td>0</td>
</tr>
<tr>
<td>SIG_IGN</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>SIG_QRY</td>
<td>Dec</td>
<td>11</td>
</tr>
<tr>
<td>SIG_SETMASK</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>SIG_UNBLOCK</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>Variable</td>
<td>Data Type</td>
<td>Numeric Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>SIGABND</td>
<td>Dec</td>
<td>18</td>
</tr>
<tr>
<td>SIGABRT</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>Dec</td>
<td>14</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>Dec</td>
<td>10</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>Dec</td>
<td>20</td>
</tr>
<tr>
<td>SIGCONT</td>
<td>Dec</td>
<td>19</td>
</tr>
<tr>
<td>SIGDANGER</td>
<td>Dec</td>
<td>33</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>SIGHUP</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>SIGILL</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>SIGINT</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>SIGIO</td>
<td>Dec</td>
<td>23</td>
</tr>
<tr>
<td>SIGIOERR</td>
<td>Dec</td>
<td>27</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>Dec</td>
<td>9</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>Dec</td>
<td>13</td>
</tr>
<tr>
<td>SIGPOLL</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>SIGPROF</td>
<td>Dec</td>
<td>32</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>Dec</td>
<td>24</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>Dec</td>
<td>11</td>
</tr>
<tr>
<td>SIGSTOP</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>SIGSYS</td>
<td>Dec</td>
<td>12</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>Dec</td>
<td>15</td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>Dec</td>
<td>26</td>
</tr>
<tr>
<td>SIGTSTP</td>
<td>Dec</td>
<td>25</td>
</tr>
<tr>
<td>SIGTTIN</td>
<td>Dec</td>
<td>21</td>
</tr>
<tr>
<td>SIGTTOU</td>
<td>Dec</td>
<td>22</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>Dec</td>
<td>16</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>Dec</td>
<td>17</td>
</tr>
<tr>
<td>SIGURG</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>SIGVTALRM</td>
<td>Dec</td>
<td>31</td>
</tr>
<tr>
<td>SIGXCPU</td>
<td>Dec</td>
<td>29</td>
</tr>
<tr>
<td>SIGXFSZ</td>
<td>Dec</td>
<td>30</td>
</tr>
<tr>
<td>ST_AAUDIT</td>
<td>Dec</td>
<td>16</td>
</tr>
<tr>
<td>ST_ACCESSACL</td>
<td>Dec</td>
<td>29</td>
</tr>
<tr>
<td>ST_ATIME</td>
<td>Dec</td>
<td>9</td>
</tr>
<tr>
<td>ST_AUDITID</td>
<td>Char</td>
<td>20</td>
</tr>
<tr>
<td>ST_BLKSIZE</td>
<td>Dec</td>
<td>18</td>
</tr>
<tr>
<td>ST_BLOCKS</td>
<td>Dec</td>
<td>22</td>
</tr>
<tr>
<td>ST_CCSID</td>
<td>Char</td>
<td>21</td>
</tr>
<tr>
<td>ST_CRTIME</td>
<td>Dec</td>
<td>19</td>
</tr>
<tr>
<td>Variable</td>
<td>Data Type</td>
<td>Numeric Value</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>ST_CTIME</td>
<td>Dec</td>
<td>11</td>
</tr>
<tr>
<td>ST_DEV</td>
<td>Hex</td>
<td>4</td>
</tr>
<tr>
<td>ST_DMODELACL</td>
<td>Dec</td>
<td>31</td>
</tr>
<tr>
<td>ST_EXTLINK</td>
<td>Dec</td>
<td>24</td>
</tr>
<tr>
<td>ST_FID</td>
<td>Bin</td>
<td>27</td>
</tr>
<tr>
<td>ST_FILEFMT</td>
<td>Dec</td>
<td>28</td>
</tr>
<tr>
<td>ST_FMODELACL</td>
<td>Dec</td>
<td>30</td>
</tr>
<tr>
<td>ST_GENVALUE</td>
<td>Bin</td>
<td>25</td>
</tr>
<tr>
<td>ST_GID</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>ST_INO</td>
<td>Hex</td>
<td>3</td>
</tr>
<tr>
<td>ST_MAJOR</td>
<td>Dec</td>
<td>14</td>
</tr>
<tr>
<td>ST_MINOR</td>
<td>Dec</td>
<td>15</td>
</tr>
<tr>
<td>ST_MODE</td>
<td>Oct</td>
<td>2</td>
</tr>
<tr>
<td>ST_MTIME</td>
<td>Dec</td>
<td>10</td>
</tr>
<tr>
<td>ST_NLINK</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>ST_RTIME</td>
<td>Dec</td>
<td>26</td>
</tr>
<tr>
<td>ST_SECLABEL</td>
<td>Dec</td>
<td>32</td>
</tr>
<tr>
<td>ST_SETGID</td>
<td>Dec</td>
<td>13</td>
</tr>
<tr>
<td>ST_SETUID</td>
<td>Dec</td>
<td>12</td>
</tr>
<tr>
<td>ST_SIZE</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>ST_STICKY</td>
<td>Dec</td>
<td>23</td>
</tr>
<tr>
<td>ST_TYPE</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>ST_UAUDIT</td>
<td>Dec</td>
<td>17</td>
</tr>
<tr>
<td>ST_UID</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>STFS_AVAIL</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>STFS_BFREE</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>STFS_BLOCKSIZE</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>STFS_FAVAIL</td>
<td>Dec</td>
<td>9</td>
</tr>
<tr>
<td>STFS_FILES</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>STFS_FFREE</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>STFS_FRSIZE</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>STFS_FSID</td>
<td>Dec</td>
<td>10</td>
</tr>
<tr>
<td>STFS_INUSE</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>STFS_INVARSEC</td>
<td>Dec</td>
<td>15</td>
</tr>
<tr>
<td>STFS_NAMEMAX</td>
<td>Dec</td>
<td>13</td>
</tr>
<tr>
<td>STFS_NOSUID</td>
<td>Dec</td>
<td>12</td>
</tr>
<tr>
<td>STFS_RDONLY</td>
<td>Dec</td>
<td>11</td>
</tr>
<tr>
<td>STFS_TOTAL</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>TM_HOUR</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>TM_ISDST</td>
<td>Dec</td>
<td>9</td>
</tr>
<tr>
<td>Variable</td>
<td>Data Type</td>
<td>Numeric Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>TM_MDAY</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>TM_MIN</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>TM_MON</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>TM_SEC</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>TM_WDAY</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>TM_YDAY</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>TM_YEAR</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>TMS_CSTIME</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>TMS_CUTIME</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>TMS_STIME</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>TMS_UTIME</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>U_MACHINE</td>
<td>Char</td>
<td>5</td>
</tr>
<tr>
<td>U_NODENAME</td>
<td>Char</td>
<td>2</td>
</tr>
<tr>
<td>U_RELEASE</td>
<td>Char</td>
<td>3</td>
</tr>
<tr>
<td>U_SYSNAME</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>U_VERSION</td>
<td>Char</td>
<td>4</td>
</tr>
<tr>
<td>VL_CLIENTPID</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>VL_CLIENTTID</td>
<td>Char</td>
<td>9</td>
</tr>
<tr>
<td>VL_DOSMODE</td>
<td>Char</td>
<td>13</td>
</tr>
<tr>
<td>VL_DOSACCESS</td>
<td>Char</td>
<td>14</td>
</tr>
<tr>
<td>VL_LOCK</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>VL_LOCKERTOK</td>
<td>Tok</td>
<td>8</td>
</tr>
<tr>
<td>VL_LOCKWAIT</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>VL_OBJCLASS</td>
<td>Char</td>
<td>10</td>
</tr>
<tr>
<td>VL_OBJIID</td>
<td>Char</td>
<td>11</td>
</tr>
<tr>
<td>VL_OBJTOK</td>
<td>Tok</td>
<td>12</td>
</tr>
<tr>
<td>VL_PURGE</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>VL_QUERY</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>VL_REGLOCKER</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>VL_SERVERPID</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>VL_UNREGLOCKER</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>VL_UNLOCK</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>W_CONTINUED</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>W_EXITSTATUS</td>
<td>Dec</td>
<td>4</td>
</tr>
<tr>
<td>W_IFEXITED</td>
<td>Dec</td>
<td>3</td>
</tr>
<tr>
<td>W_IFSIGNALED</td>
<td>Dec</td>
<td>5</td>
</tr>
<tr>
<td>W_IFSTOPPED</td>
<td>Dec</td>
<td>7</td>
</tr>
<tr>
<td>W_NOHANG</td>
<td>Dec</td>
<td>1</td>
</tr>
<tr>
<td>W_OK</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>W_STAT3</td>
<td>Dec</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 12. List of predefined variables (continued)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data Type</th>
<th>Numeric Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>W_STAT4</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>W_STOPSIG</td>
<td>Dec</td>
<td>8</td>
</tr>
<tr>
<td>W_TERMSIG</td>
<td>Dec</td>
<td>6</td>
</tr>
<tr>
<td>W_UNTRACED</td>
<td>Dec</td>
<td>2</td>
</tr>
<tr>
<td>X_OK</td>
<td>Dec</td>
<td>1</td>
</tr>
</tbody>
</table>
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

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. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you may view the information through the z/OS Internet Library website or the z/OS Information Center. If you continue to experience problems, send an email to mhvrcfs@us.ibm.com or write to:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User's Guide, and z/OS ISPF User's Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer or Library Server versions of z/OS books in the Internet library at:

http://www.ibm.com/systems/z/os/zos/bkserv/

One exception is command syntax that is published in railroad track format, which is accessible using screen readers with the Information Center, as described in "Dotted decimal syntax diagrams."

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always
present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 \* FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* \* FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, for example 5? NOTIFY. If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only
one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

• * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:
1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

• + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.
Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel  
IBM Corporation  
2455 South Road  
Poughkeepsie, NY 12601-5400  
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Acknowledgments

InterOpen/POSIX Shell and Utilities is a source code product providing POSIX.2 (Shell and Utilities) functions to z/OS UNIX System Services. InterOpen/POSIX Shell and Utilities is developed and licensed by Mortice Kern Systems (MKS) Inc. of Waterloo, Ontario, Canada.
# Index

## Numerics

- 3270 data stream 139
- 3270 passthrough mode 139

## A

- access syscall command 23
- access time, set 196
- accessibility 305
- ACL
  - deleting 24
  - deleting a specific entry 25
  - obtain resources 29
  - read 27
  - read entry 28
  - release resources 26
  - replace 30
  - update entry 31
- ACL_DELETE 25, 28, 31
- ACL_ENTRY_GROUP 25, 28, 31
- ACL_ENTRY_TYPE 25, 28, 31
- ACL_ENTRY_USER 25, 28, 31
- ACL_EXECUTE 25, 28, 31
- ACL_ID 25, 28, 31
- ACL_READ 25, 28, 31
- ACL_TYPE_ACCESS 24, 27, 30
- ACL_TYPE_DIRDEFAULT 24, 27, 30
- ACL_TYPE_FILEDEFAULT 24, 27, 30
- ACL_WRITE 25, 28, 31
- acldelete syscall command 24, 25
- aclfree syscall command 26
- aclget syscall command 27
- aclgetentry syscall command 28
- aclinit syscall command 29
- aclset syscall command 30
- aclupdateentry syscall command 31
- ADDRESS TSO 5
- alarm syscall command 33
- appropriate privileges 14
- AUD_FEXEC 41, 56
- AUD_READ 41, 56
- AUD_WRITE 41, 56
- AUD_SEXEC 41, 56
- AUD_SREAD 41, 56
- AUD_SWRITE 41, 56
- audit flags 41, 56
- authorization 14
- automatic file conversion control 61

## B

- batch, running a REXX program 2
- BITAND 304
- BITOR 304
- BPX messages
  - looking up xi
- BPX.SUPERUSER FACILITY 14
- BPXWDYN 247
  - calling conventions 247
  - examples 259
  - freeing an output descriptor 259
  - keywords 249
  - message processing 250
  - request types 248
  - requesting allocation information 256
  - requesting dynamic allocation 251
  - requesting dynamic concatenation 254
  - requesting dynamic output 257
  - requesting dynamic unallocation 255
  - return codes 249
- BPXWRBLD module 15
- bpxwunix() function 219

## C

- calling process
  - effective group ID 153, 155
  - effective user ID 90, 154
  - making it a session leader 164
  - parent process ID 105
  - process group ID 103
  - process ID 104
  - real group ID 155
  - real user ID 113, 165
  - saved set group ID 155
  - saved set user ID 165
  - calling process, effective group ID 89
  - calling process, real group ID 91
  - calling process, supplementary group ID 95
  - calling thread
    - signal mask 170
    - suspending 136, 172, 173
  - catclose syscall command 34
  - catgets syscall command 35
  - catopen syscall command 36
  - cert syscall command 37
  - certificate registration 37
  - character set portable filename 19
  - character special file 126
  - charin() function 221
  - charout() function 222
  - chars() function 223
  - chattr syscall command 38
  - chaudit syscall command 41
  - chdir syscall command 42
  - child process 174, 177
  - chmod syscall command 43
  - chmod() function 224
  - chown syscall command 45
  - close syscall command 46
  - closedir syscall command 47
  - coding examples 205, 289
  - command
    - syntax diagrams xiii
    - command output trapping 219, 233
    - command, shell, su 14
- commands
  - stream() 240
  - configurable limit value 73, 134, 185
  - convd2e() function 225
  - creat syscall command 48
  - current directory returning or changing 226
  - customization
    - host command environment 13
    - performance 13
  - REXX environment 13
- CVT_QUERYCVT 61
- CVT_SETAUTOCVTON 61
- CVT_SETCVTOFF 61
- CVT_SETCVTON 61

## D

- database, group
  - retrieving an entry 92
  - rewinding 156
- database, user
  - entry retrieval 109
  - rewinding the 159
- delete a directory rmdir 152
- delete a file 192
- unlink 192
- directory
  - changing group 45
  - changing mode 43
  - changing owner 45
  - closing 47
  - creating 124
  - entry 192
  - file 192
  - group 59
  - mode, changing 58
  - name, changing 150
  - owner 59, 119
  - reading multiple entries 142, 146
  - removing 152
  - rewind 151
  - stream 133
- directory() function 226
- disability 305
- dup syscall command 49
- dup2 syscall command 50
- dynamic allocation 247
- dynamic output 247

## E

- effective group ID calling process 89, 153, 155
- effective user ID calling process 90, 154, 165
- entry, directory 192
- environment REXX 15

© Copyright IBM Corp. 1996, 2011
SH 3
SYSCALL 2
environment variable altering 227
querying 227
environment() function 227
Epoch, seconds since 186
ERRNO 22
ERRNOJR 22
error codes, retrieving text 183
error messages looking up xi
ESC_A 201
ESC_B 201
ESC_F 201
ESC_N 201
ESC_R 201
ESC_T 201
ESC_V 201
examples, coding 205, 289
EXECIO 7, 8, 16
exists() function 228
exit status from REXX 9
external functions 4
extlink syscall command 52
f_closfd syscall command 60
f_control_cvt syscall command 61
f_dupfd syscall command 64
f_dupfd2 syscall command 65
f_getfd syscall command 66
f_getfl syscall command 67
f_getlk syscall command 68
F_OK 23
F_RDLCK 69, 78, 81, 269
f_setfd syscall command 75
f_setfl syscall command 76
f_setlk syscall command 77
f_setlksw syscall command 80
f_settag syscall command 82
F_UNLCK 69, 78, 81, 270
F_WRLCK 69, 78, 81, 270
fchattr syscall command 53
fchaudit syscall command 56
fchmod syscall command 57
fchown syscall command 59
fcntl syscall command 63
FD_CLOEXEC 75, 138
FID 261
FIFO special file 125
file access and modification times 196
automatic conversion 61
changes on DASD 86
changing attributes 38, 53
changing group 45
changing mode 43, 224
changing owner 45, 119
changing size 87
closing 46
creating 48
deleting 192
erasing 192
group 59
file (continued)
locks 77
mode, changing 58, 224
name, changing 130
new, opening 48
offset 121
opening 131
owner 59
reading bytes 144
reading lines 147
removing 192
size 188
status 178
status flags 67, 76
status information 83, 123
file access
determining 23, 232
file descriptor 131
file descriptors
0, 1, and 2 7, 16
closing a range 60
duplicating 49, 50, 64, 65
error 7
file attributes, changing 53
flags 66, 75
REXX environment 17
file locks 68, 80
file mode creation mask 190
file stream
opening 217
file system
mounted 85, 98, 99, 182
mounting 127
quiescing 141
status 181
unmounting 193
unquiescing 195
file system server
creating a file 262
creating a link 267
creating a new directory 274
creating a symbolic link 285
file attributes, setting 284
getting file attributes 266
locking 268
reading file data 275
registering a process as a server 278
releasing a vnode token 279
removing a directory entry 280
removing an empty directory 282
renaming a file or directory 281
returning a vnode token 265
returning entries from a directory 276
returning file system status 264
returning the contents of a symbolic link 277
returning vnode and VFS tokens 283
searching for a file 273
syscall commands
FID 261
security 261
tokens 261
VFS 261
vnode 261
writing file data 287
file tag 82
fork syscall command 70
forkexecm syscall command 71
fpathconf syscall command 73
fstat syscall command 83
fstatvfs syscall command 85
fsync syscall command 86
ftrunc syscall command 86
functions
external 4
REXX 217
G
getcwd syscall command 88
getegid syscall command 89
geteuid syscall command 90
getgid syscall command 91
getgrent syscall command 92
getggrid syscall command 93
getgrnam syscall command 94
getgroups syscall command 95
groupsbyuname syscall command 96
groupsbyuid syscall command 97
groupsbyname syscall command 98
groupsbyusername syscall command 99
groupsbyusernamebyname syscall command 99
getopt function 206
getpass() function 229
getppgrp syscall command 103
getpgrp syscall command 104
getppid syscall command 105
getpgrp syscall command 106
getpgrp syscall command 109
getpwuid syscall command 110
getpwuid syscall command 111
getreuid syscall command 112
getreuid syscall command 113
gmtime syscall command 114
GR_GID 92, 93, 94
GR_MEM 92, 93, 94
GR_MEMBERS 92, 93, 94
GR_NAME 92, 93, 94
group 93
changing file or directory 45
file or directory 119
information 94
symbolic link 119
group database
retrieving an entry 92
rewinding 156
group, file or directory 59
H
hard link 120
removing 192
host command environment customizing 13
SH 3
input and output 7
return code 22
SYSCALL 2, 3
working directory 19
host command environments z/OS UNIX 1
I
 immediate commands 11
 input and output
 syscall commands 8
 z/OS UNIX 7
 ioctl syscall command 115
 IRXEXEC service 16
 IRXJCL service 16
 isatty syscall command 116
 ISPF editor 4

J
 JES
 submitting a job to 244

K
 keyboard 305
 kill syscall command 117

L
 L_LEN 69, 78, 81
 L_PID 69, 78, 81
 L_START 69, 78, 81
 L_TYPE 69, 78, 81
 lchown syscall command 119
 linein() function 230
 lineout() function 231
 lines() function 232
 link
 hard 120, 192
 symbolic 192, 285
 creating to an external name 52
 reading the contents 148
 link syscall command 120
 linkage 15
 LINKLIB 71
 locks, file 77, 80
 querying 68
 login name 97
 LPALIB 71
 lseek syscall command 121
 lstat syscall command 123
 lstat syscall command (continued)

M
 macro language, using REXX 15
 mainframe
 education xii
 message catalog 34, 36
 message, finding and returning 35
 messages
 looking up xi
 mkdir syscall command 124
 mkfifo syscall command 125
 mknod syscall command 126
 MNTASYNCHMOUNT 101
 MNT_FILEACTIVE 101
 MNT_FILEDEAD 101
 MNT_FILEDRAIN 101
 MNT_FILEFORCE 101
 MNT_FILEIMMDED 101
 MNTFILENORM 101
 MNTFILERESET 101
 MNTIMMEDTIED 101
 MNT_MODE_AUNMOUNT 100, 128, 129
 MNT_MODE_CLIENT 100
 MNT_MODE_EXPORT 100
 MNT_MODE_NOAUTOMOVE 100, 128, 129
 MNT_MODE_NOSEC 100, 128
 MNT_MODE_NOSETID 100, 128
 MNT_MODE_RDONLY 100, 128, 129
 MNT_MODE_RDWR 100, 128, 129
 MNT_MODE_SECACL 100
 MNT_MOUNTINPROGRESS 101
 MNT_QUEUESCED 101
 MNTE_AGGNAME 99
 MNTE_BYTESREADHW 99
 MNTE_BYTESREADLW 99
 MNTE_BYTESWRITTENHW 99
 MNTE_BYTESWRITTENLW 99
 MNTE_DD 99
 MNTE_DEV 99
 MNTE_DIRIBC 99
 MNTE_FILETAG 99, 128
 MNTE_FROMSYS 99
 MNTE_FSNAME 99, 128
 MNTE_FSTYPE 99, 128
 MNTE_MODE 100, 128, 129
 MNTE_PARM 100, 128
 MNTE_PATH 100, 128
 MNTE_PFSSTATUSEXCP 100
 MNTE_PFSSTATUSNORMAL 100
 MNTE_QJOBNAME 100
 MNTE_QPID 100
 MNTE_QSYSNAME 100
 MNTE_READCT 100
 MNTE_READIBC 100
 MNTE_RFLAGS 100, 129
 MNTE_ROOTINO 100
 MNTE_ROSECLABEL 100
 MNTE_STATUS 101
 MNTE_STATUS2 101
 MNTE_SUCCESS 101
 MNTE_SYSLIST 101, 128, 129
 MNTE_SYSNAME 101, 128, 129
 MNTE_TYPE 101
 MNTE_UID 101
 MNTE_WRITECT 101
 MNTE_WRITECT 101
 MNTE_WRITEIBIC 101
 mode changing 43, 58
 mode specifying 19
 modification time, set 196
 mount syscall command 127
 MNT_MODE_AUNMOUNT 128, 129
 MNT_MODE_NOAUTOMOVE 128, 129
 MNT_MODE_NOSEC 128
 MNT_MODE_NOSETID 128
 MNT_MODE_RDONLY 128, 129
 MNT_MODE_RDWR 128, 129
 MNTFILETAG 128
 MNTE_FILENAME 128
 MNTE_FSTYPE 128
 MNTE_MODE 128, 129
 MNTE_PARM 128

mount syscall command (continued)
 MNTE_PATH 128
 MNTE_RFLAGS 129
 MNTE_SYSLIST 128, 129
 MNTE_SYSNAME 128, 129
 mounted file systems 98, 99
 mountnt syscall command
 MNTE_FSTYPE 128

N
 Notices 309
 NUMBER OFF 4
 numeric GID 160
 numeric UID 161
 numerics 20

O
 O_APPEND 67, 76, 131
 O_CREAT 67, 131
 O_EXCL 67, 131
 O_NOCTTY 67, 131
 O_NONBLOCK 67, 76, 131, 138
 O_RDONLY 67, 131
 O_RDWR 67, 131
 O_SYNC 67, 76, 131
 O_TRUNC 67, 131
 O_WRONLY 67, 131
 octal numbers 303
 offset, file 121
 open syscall command 131
 opendir syscall command 133
 opening
 file stream 217
 process stream 218
 outtrap() function 233
 owner
 changing file or directory 45
 file or directory 119
 symbolic link 119
 owner, file or directory 59

P
 parent process ID calling 105
 PARSE EXTERNAL instruction 7, 16
 PARSE SOURCE instruction tokens 9
 passthrough mode, 3270 139
 password
 prompting for 229
 returning 229
 pathconf syscall command 134

Index 315
pathname
  returning a 228
  specifying 19
  terminal 189
pause syscall command 136
PC_ACL 135
PC_ACL_MAX 135
PC_LINK_MAX 74, 135
PC_MAX_CANON 74, 135
PC_MAX_INPUT 74, 135
PC_PATH_MAX 74, 135
PC_PIPE_BUF 74, 135
PC_POSIX_CHOWN_RESTRICTED 74, 135
PC_POSIX_NO_TRUNC 74, 135
PC_POSIX_VDISABLE 74, 135
pending signals 169
performance 13
permissions
  octal 303
  setting 303, 304
pfsctl syscall command 137
pipe syscall command 138
pipe, creating 138
portable filename character set 19
POSIX epoch time
  converting timestamp to 225
predefined variables
  data type and numeric value 291
  effect of PROCEDURE 7
  using 21
PROCEDURE instruction 7
process
  creating 174
  retrieving information about 234
  signal actions 166
  spawning a child 174, 177
  starting time 114
  status 106
  wait
  signals-enabled 239
process group ID 103
  assigning 158
process ID 104
process stream 218
processor time used 187
procoinfo() function 234
program
  executing from LINKLIB, LPALIB, or STEPLIB 71
  running a REXX program 4
programming services 15
  PS_CHILD 107
  PS_CMD 106
  PS_CONTTY 106
  PS_EGID 106
  PS_EUID 106
  PS_FGID 106
  PS_FORK 107
  PS_FREEZE 107
  PS_MAXVNODES 106
  PS_MSGRCV 107
  PS_MSGSND 107
  PS_PATH 106
  PS_PAUSE 107
  PS_PGPID 106
  PS_PID 106
  PS_PPID 106
  PS_QUIESCE 107
  PS_RGID 106
  PS_RUID 106
  PS_RUN 107
  PS_SEMWT 107
  PS_SERVERFLAGS 106
  PS_SERVERNAME 106
  PS_SERVERTYPE 106
  PS_SGID 106
  PS_SID 106
  PS_SIZE 106
  PS_SLEEP 107
  PS_STARTTIME 106
  PS_STAT 106
  PS_STATE 107
  PS_SUID 107
  PS_SYSTIME 107
  PS_USERTIME 107
  PS_VNODECOUNT 107
  PS_WAITC 107
  PS_WAITF 107
  PS_WAITO 107
  PS_ZOMBIE 107
  PS_ZOMBIE2 107
  pt3270 syscall command 139
publications
  on CD-ROM xi
  softcopy xi
PW_DIR 109, 110, 111
PW_GID 109, 110, 111
PW_NAME 109, 110, 111
PW_SHELL 109, 110, 111
PW_UID 109, 110, 111
Q
quiesce syscall command 141
R
R_OK 23
RACF (Resource Access Control Facility)
  BPX_SUPERUSER FACILITY 14
RC 21
  REXX variable 6
rddir syscall command 137
read syscall command 144
read dir syscall command 144
readlink syscall command 147
readlink syscall command 147
real group ID calling process 91, 155
real user ID calling process 113, 165
realpath syscall command 149
relative pathname working directory 19
removing a directory 19
rmrmdir syscall command
  removing or deleting a directory 152
routine, signal interface 3
S
S_FFBINARY 84, 180
S_FCCR 84, 180
S_FCCRFL 84, 180
S_FFLF 84, 180
S_FFLFCR 84, 180
S_FFNAL 84, 180
S_FFNAL 84, 180
S_FSNR 84, 180
S_ISDIR 84, 180
return code
  from a REXX program 9
  SH host command environment 22
  TSO/E 21
  z/OS shell 22
RETVAL 21
rewinddir syscall command 151
SIGNAL 21
SIGCHLD 134
SIGCONT 134
SIGSTOP 134
SIGTERM 134
SLEEP 107
SYN 183
SYSLINK 125
system() function 227
syscalls 245
REXX program
  exit status 9
  from TSO/E or batch 10
  moving from TSO/E to a z/OS shell 11
  querying options 238
  running a program 9
  running a z/OS shell or a program 4
  running from the z/OS shell 9
  running from TSO/E or batch 2
  setting options 238
REXX programming services 15
REXX service
  IRXEEXEC 16
  IRXJCL 16
  rexxopt() function 238
RLIM_INFINITY 162
RLIMIT_AS 112, 162
RLIMIT_CORE 112, 162
RLIMIT_CPU 112, 162
RLIMIT_FSIZE 112, 162
RLIMIT_NOFILE 112, 162
RLIMIT_RSS 112, 162
rmmdir syscall command
  removing or deleting a directory 152
S_FFSIZE 84, 180
S_FFTIME 84, 180
S_FTAL 84, 180
S_FTALK 84, 180
S_FTALLOK 84, 180
S_FTALOK 84, 180
S_FTALOKD 84, 180
S_FTAL 84, 180
S_FTAL 84, 180
S_FTAL 84, 180
S_FTAL 84, 180
S_FTAL 84, 180
Index 317