UNISYS

Product Information Announcement

o New Release ● Revision o Update o New Mail Code

Title

MCP/AS ALGOL and MCP Interfaces to POSIX® Features Programming Reference Manual (7011 8351–002)

This announces a retitling and reissue of the *A Series ALGOL and MCP Interfaces to POSIX® Features Programming Reference Manual*. No new technical changes have been introduced since the HMP 1.0 and SSR 43.2 release in June 1996.

This manual describes functions used to obtain certain POSIX-related features in programs not written in the C language. Essentially, each function mimics a C language POSIX function. Most functions call library procedures exported by the MCPSUPPORT library.

The POSIX interface was developed by the Institute of Electrical and Electronics Engineers, Inc. (IEEE).

- United States customers, call Unisys Direct at 1-800-448-1424.
- Customers outside the United States, contact your Unisys sales office.
- Unisys personnel, order through the electronic Book Store at http://iwww.bookstore.unisys.com.

Comments about documentation can be sent through e-mail to **doc@unisys.com**.

Announcement only:

Announcement and attachments:

System:	MCP/AS
Release:	HMP 4.0 and SSR 45.1
Date:	June 1998
Part number:	7011 8351–002

MCP/AS
ALGOL AND MCP INTERFACES TO POSIX® FEATURES

UNISYS

Programming Reference Manual

HMP 4.0 and SSR 45.1 June1998

Printed in USA

Priced Item 7011 8351–002

Correspondence regarding this publication should be forwarded to Unisys Corporation by addressing remarks to Unisys Corporation, Tredyffrin/Malvern Product Information, N155, 2476 Swedesford Road, Malvern, PA 19355-9700, U.S.A.

Comments about documentation can also be sent through e-mail to **doc@unisys.com**.

Contents

Tables

Examples

About This Manual

Purpose

This manual describes functions used to obtain certain POSIX interface features in programs not written in the C language. Essentially, each function mimics a C language POSIX interface function. Most functions call library procedures exported by the MCPSUPPORT library.

The manual describes:

- POSIX interface functions for ALGOL programs. A system-supplied include file (SYMBOL/POSIX/ALGOL/PROPERTIES) contains declarations required for these functions.
- POSIX interface related library procedures exported by the MCPSUPPORT library.

The *Master Control Program (MCP) System Interfaces Programming Reference Manual* previously described many of the POSIX interface related library procedures. These library procedures are:

- POSIX_INTEGERIDS
- POSIX_NANOALARM
- POSIX_NANOSLEEP
- POSIX_SEM_CLOSE
- POSIX_SEM_DESTROY
- POSIX_SEM_GETVALUE
- POSIX_SEM_INIT
- POSIX_SEM_OPEN
- POSIX_SEM_POST
- POSIX_SEM_TRYWAIT
- POSIX_SEM_UNLINK
- POSIX_SEM_WAIT
- POSIX_SETIDS
- POSIX_SIGHANDLER
- POSIX_STRINGIDS

Scope

The functions and library procedures described in this manual are for A Series systems. You can access them in programs to perform a variety of functions. Some of these functions include:

- Setting and retrieving POSIX user and group IDs
- Managing POSIX semaphores to synchronize programs or control shared resources
- Managing POSIX signals
- Translating system error messages

Audience

This document is a reference manual intended primarily for use by programmers. It is particularly relevant for ALGOL and NEWP programs written to be part of, or to interact with, C language programs.

Prerequisites

You should be familiar with:

- Using libraries as described in the *Task Management Programming Guide*
- POSIX interface concepts as defined in the *POSIX User's Guide*
- POSIX interface based functions as described in the C Programming Reference Manual, Volume 2: Headers and Functions

How to Use This Manual

This is a reference manual that can be read in any desired order. However, all users should first read Section 1. This section provides an overview of the manual's contents.

If you are an ALGOL programmer you should:

- Read the ALGOL include file description in Section 2.
- Use Section 3 to obtain reference information about supported POSIX interface functions. This section lists functions in alphabetical order.
- Read the "About this Section" portion of Section 5. This subsection describes available functions that you can only access with library procedures.
- Refer to Section 6 for representative programming examples.

If you are a NEWP programmer you should:

- Read the "About this Section" portion of Section 5. Table 5-1 lists the library procedure applicable to each function.
- Use Section 5 to obtain reference information about library procedures. (Note that all library declarations use ALGOL syntax.)
- Refer to Section 6 for representative programming examples.

All users should refer to Section 4 if they cannot find information about a particular function. Section 4 lists unsupported POSIX interface C functions.

Notes:

1. *The library procedures described in Section 5 are internal interfaces used by the system software. These interfaces might also be of use to sophisticated application programs. From one release to another, an internal interface might change in such a way that programs that use the internal interface will be required to make changes to operate correctly. Because internal interfaces are special system interfaces, they do not adhere to the compatibility policies described in the* SSR 42.3 Software Release Capabilities Overview. *You should examine all programs that use internal interfaces before installing a new release to ensure that the internal interface has not changed.*
2. *Wherever possible, ALGOL programmers should use the capabilities described in Sections 2 and 3. Normal policies support these capabilities.*

Organization

This document contains six sections and an index.

Section 1. Introduction

This section provides an overview of POSIX interface functions. It describes how to access these features in non C language programs.

Section 2. Reference Information

This section provides reference information required to support function descriptions in Section 3 and library procedure descriptions in Section 5.

Section 3. POSIX Functions in ALGOL

This section describes POSIX functions that are available in the ALGOL language. To access these functions, include the system-supplied SYMBOL/POSIX/ALGOL/PROPERTIES file in the ALGOL program. Section 3 does not contain detailed functional descriptions. If necessary, refer to the equivalent functional descriptions in the *C Programming Reference Manual, Volume 2: Headers and Functions*.

Section 4. Unsupported POSIX Functions

This section lists the POSIX interface based functions that are currently available only in C language programs.

Section 5. POSIX-Related Library Procedures

This section describes each library procedure that provides one or more POSIX interface related functions.

Section 6. Programming Examples

This section contains several sample programs that illustrate the use of POSIX interface functions in ALGOL programs.

Related Product Information

Unless otherwise stated, all documents referred to in this publication are MCP/AS documents. The titles have been shortened for increased usability and ease of reading.

The following documents are included with the software release documentation and provide general reference information:

- The *Glossary* includes definitions of terms used in this document.
- The *Documentation Road Map* is a pictorial representation of the Product Information (PI) library. You follow paths through the road map based on tasks you want to perform. The paths lead to the documents you need for those tasks. The *Road Map* is available on paper and on the PI Library CD-ROM. If you know what you want to do, but don't know where to find the information, start with the *Documentation Road* Map.
- The *Information Availability List* (IAL) lists all user documents, online help, and HTML files in the library. The list is sorted by title and by part number.

The following documents provide information that is directly related to the primary subject of this publication.

***ALGOL Programming Reference Manual, Volume 1: Basic Implementation* (8600 0098)**

This manual describes the basic features of the Extended ALGOL programming language. This manual is written for programmers who are familiar with programming concepts.

***C Programming Reference Manual, Volume 1: Basic Implementation* (8600 2268)**

This manual describes the C programming language. It includes descriptions of syntax, status messages, the preprocessor, compiling system, binding system, and run-time library. Extensions such as compiler control options and the A Series library facility are also documented. This manual is written for systems and applications programmers.

***C Programming Reference Manual, Volume 2: Headers and Functions* (8600 2278)**

This manual describes the C headers in detail, and the functions, macros, and types defined in those headers. This manual is written for systems and applications programmers.

***File Attributes Programming Reference Manual* (8600 0064)**

This manual contains information about each file attribute and each direct I/O buffer attribute. The manual is written for programmers and operations personnel who need to understand the functionality of a given attribute. The *I/O Subsystem Programming Guide* is a companion manual.

***I/O Subsystem Programming Guide* (8600 0056)**

This guide contains information about how to program for various types of peripheral files and how to program for interprocess communication, using port files. This guide is written for programmers who need to understand how to describe the characteristics of a file in a program. The *File Attributes Programming Reference Manual* is a companion manual.

***POSIX User's Guide* (7011 8328)**

This guide describes the basic concepts of the POSIX interface, including process control and file management. It also describes specifically how the POSIX.1 interface is implemented and used on the enterprise server. This guide is written for programmers and any user who wants to understand the POSIX interface.

***Task Attributes Programming Reference Manual* (8600 0502)**

This manual describes all the available task attributes. It also gives examples of statements for reading and assigning task attributes in various programming languages. The *Task Management Programming Guide* is a companion manual.

***Task Management Programming Guide* (8600 0494)**

This guide explains how to initiate, monitor, and control processes on an enterprise server. It describes process structures and process family relationships, introduces the uses of many task attributes, and gives an overview of interprocess communication techniques. The *Task Attributes Programming Reference Manual* is a companion manual.

Section 1 **Introduction**

Overview of POSIX Functions

This document provides reference information necessary to access POSIX functions in programs not written in the C language. Most of these functions call MCPSUPPORT library procedures. Two function categories are described:

If a function is . . .	Then . . .
Also implemented in C language	• The non C language function mimics the equivalent C language function. • This manual does NOT describe the non C language function in detail. • See the C Programming Reference Manual, Volume 2: Headers and Functions for detailed information about the function.
Not implemented in C language	This manual provides a complete description of the function.

Accessing POSIX Functions in ALGOL Programs

You can use many POSIX functions in ALGOL programs without any explicit declaration of library procedures. To do so, include the file SYMBOL/POSIX/ALGOL/PROPERTIES in the program.

The ALGOL program product contains the SYMBOL/POSIX/ALGOL/PROPERTIES file. This file provides the following functions:

- It declares the MCPSUPPORT library.
- It declares supported POSIX interface related library procedures.
- It defines a suite of POSIX functions.

Section 2 provides a more complete description of the ALGOL include file.

Section 3 describes all POSIX interface related functions currently supported by the ALGOL include file.

ALGOL programmers can access several additional POSIX functions by explicitly declaring appropriate library procedures. The introduction to Section 5 describes this concept.

Accessing POSIX Features with Library Procedures

Note: *In most cases, ALGOL programmers do not need to declare library procedures. The ALGOL include file provides these declarations. Programmers using other languages (primarily NEWP) must declare an appropriate library procedure to access a desired POSIX function.*

MCPSUPPORT library procedures provide the majority of POSIX interface related functions. Each library procedure provides a unique entry point into the library.

Section 5 of this manual provides:

- A table that lists all supported POSIX functions and associated MCPSUPPORT library procedures
- A description of every applicable library procedure

Section 5 defines some POSIX functions that the ALGOL include file does not support. ALGOL programmers can use the appropriate library procedure to access those functions.

Documentation Conventions

This document uses the following conventions:

- The term *POSIX* refers to the A Series POSIX implementation as described in the *POSIX User's Guide*. Therefore, the *C Programming Reference Manual*, *Volume 2*, might categorize a referenced function as any of the following:
  - "Implementation Extension"
  - "POSIX"
  - "X/Open"
- Terms in uppercase characters refer to POSIX functions defined in the ALGOL include file. *ACCESS* and *MKFIFO* are two examples.
- C language semantics reference equivalent C language functions. The terms *access( )* and *mkfifo( )* are two examples.

  Note that the *C Programming Reference Manual*, *Volume 2*, lists these functions without trailing parenthesis. The terms *access* and *mkfifo* are two examples.
- C language #include precompiler directives are not listed in this manual.
- In Section 5, equivalent C language function names describe most of the functions provided by MCPSUPPORT library procedures.

Section 2
Reference Information

About this Section

This section provides the reference information required to use the information presented in Sections 3 and 5. In general, there are multiple references to this information.

Section 2 describes:

- The ALGOL include file. Use this file to access POSIX functions in an ALGOL program.
- The "rules" required to specify POSIX function or library procedure parameters and to interpret the results.

  Every description within Section 3 and Section 5 includes a reference to one of these rules.
- Defined names and values associated with integer parameters. The ALGOL include file defines the indicated names.
- The layout of structures passed to or from POSIX functions and library procedures.

The ALGOL Include File

An include file (SYMBOL/POSIX/ALGOL/PROPERTIES) is now available to facilitate the use of POSIX functions in ALGOL programs. This file supports all functions described in Section 3 of this document.

The ALGOL program product contains the SYMBOL/POSIX/ALGOL/PROPERTIES file.

If an ALGOL program includes the SYMBOL/POSIX/ALGOL/PROPERTIES file:

- The MCPSUPPORT library is declared.
- Supported POSIX interface related library procedures are declared.
- POSIX functions defined in Section 3 can be used.

Notes:

1. *ALGOL programs can use several POSIX functions not listed in Section 3. See the introduction to Section 5 for further information.*
2. *In future releases, the include file will support additional POSIX functions.*

Specifying POSIX Functions in an ALGOL Program

An ALGOL program must include the SYMBOL/POSIX/ALGOL/PROPERTIES file to use the POSIX functions described in Section 3. To include this file, insert the following code at the start of the program:

```
$$ INCLUDE "SYMBOL/POSIX/ALGOL/PROPERTIES"
```

Contents of the File

The SYMBOL/POSIX/ALGOL/PROPERTIES file contains three parts:

- Part 1

  This part contains all POSIX interface related library procedure declarations and defined SELECTOR parameter values.
- Part 2

  This part provides global defines for data referenced by more than one library procedure. Information defined here includes:
  - Common constants
  - Structure definitions
- Part 3

  This part specifies each POSIX function and any required library procedure calls. See Section 3 for information about these POSIX functions.

Rules for Using Parameters and Results

Most of the functions described in Section 3 emulate POSIX interface related C language functions as described in the *C Programming Reference Manual, Volume 2: Headers and Functions.* In Section 3, each description includes:

- Reference to the corresponding C language function.
- Cross references between ALGOL parameters and equivalent C language arguments.
- References to the “rule” needed to match ALGOL parameters with equivalent C language arguments.

The following paragraphs describe all required parameter matching rules.

Note: *Library procedure descriptions (provided in Section 5) also refer to these rules.*

Call-by-reference Integer

Define this parameter as a call-by-reference integer. When the procedure is invoked, the system evaluates the *location* of the actual parameter and replaces the formal parameter with a reference to that location. Thereafter, any change in the formal parameter affects the actual parameter within the program.

The formal parameter is declared REFERENCE and INTEGER.

Call-by-reference Real

Define this parameter as a call-by-reference real number. When the procedure is invoked, the system evaluates the *location* of the actual parameter and replaces the formal parameter with a reference to that location. Thereafter, any change in the formal parameter affects the actual parameter within the program.

The formal parameter is declared REFERENCE and REAL.

Call-by-value Integer

Define this parameter as a call-by-value integer. A copy of the actual parameter value is passed to the procedure. Thereafter, any change to the formal parameter has no effect outside the procedure body.

The formal parameter is declared VALUE and INTEGER.

Call-by-value Real

Define this parameter as a call-by-value real number. A copy of the actual parameter is passed to the procedure. Thereafter, any change to the formal parameter has no effect outside the procedure body.

The formal parameter is declared VALUE and REAL.

EBCDIC Array Input

The library procedure expects a string of EBCDIC characters. You must define three parameters:

1. A call-by-reference EBCDIC array.

   Declare EBCDIC ARRAY (with [0] bounds) and REFERENCE.
2. A call-by-value integer (<name>_OFF) that specifies a byte-offset to the start of data within the array.
3. A call-by-value integer (<name>_LEN) that specifies the length (in bytes) of the data string.

You may specify a length of –1 if the data is “string-type.” With this specification, the system implicitly determines the string length by scanning the array for a null character (48“00”). An error occurs if no null character is detected.

EBCDIC Array Output

The program expects a string of EBCDIC characters from the library procedure. You must define three parameters:

1. A call-by-reference EBCDIC array.

   Declare EBCDIC ARRAY (with [0] bounds) and REFERENCE.
2. A call-by-value integer (<name>_OFF) that specifies a byte-offset to the start of data within the array.
3. A call-by-value integer (<name>_MAX) that specifies the maximum number of bytes (starting from the offset) available to store the character string.

A null character defines the end of the data.

An ERRNO value is set if there is not enough space to store all data, exclusive of the null character. If all data characters are stored but there is insufficient room for the null character, no error is set.

File

Define the applicable parameter as a file.

The formal parameter is declared REFERENCE and FILE.

Integer Array Input

The library procedure expects an integer or series of integers. You must define three parameters:

1. A call-by-reference integer array.

   Declare INTEGER ARRAY (with [0] bounds) and REFERENCE.
2. A call-by-value integer (<name>_OFF) that specifies a word-offset to the start of data within the array.
3. A call-by-value integer (<name>_LEN) that specifies the length (in words) of the data.

Integer Array Output

The program expects one or more integers from the library procedure. You must define three formal parameters:

1. A call-by-reference integer array.

   Declare INTEGER ARRAY (with [0] bounds) and REFERENCE.
2. A call-by-value integer (<name>_OFF) that specifies a word-offset to the start of data within the array.
3. A call-by-value integer (<name>_MAX) that specifies the length (in words) of the data area.

The MCP sets an ERRNO condition if there is not enough space to store all data.

Path Definition

The library procedure expects a pathname containing a string of EBCDIC characters. You must define five parameters:

1. A call-by-reference EBCDIC array.

   Declare EBCDIC ARRAY (with [0] bounds) and REFERENCE.
2. A call-by-value integer (PATH_OFF) that specifies a byte-offset from the beginning of the PATH array to the start of the pathname string.

3. A call-by-value integer (PATH_LEN) that specifies the length (in characters) of the pathname string. There are two ways to express the PATH_LEN value:

Value	Description
–1	Allow the MCP to calculate string length. The MCP will assume the string is null terminated and will calculate its length. If the MCP does not detect a null character, it sets an ERRNO value. To use this option, the PATH_TYPE parameter must be 0 (PATH_TYPE_PATHNAME).
> 0	Specified value is the length (in characters) of the input file name string.

4. A call-by-value integer (PATH_TYPE) that defines how the associated parameter string must be interpreted:

Value	Defined Name	Description
0	PATH_TYPE_PATHNAME	String contains a display form name conforming to the syntax of the PATHNAME file attribute. See the *File Attributes Programming Reference Manual* for details.
1	PATH_TYPE_TITLE	String contains a display form name conforming to the syntax of the TITLE file attribute. See the *File Attributes Programming Reference Manual* for details.
2	PATH_TYPE_STANDARD	String contains a standard form file name.

5. A call-by-value integer (PATH_SEARCHRULE) that defines rules to be followed when evaluating the defined string:

Value	Defined Name	Description
0	NATIVE	Use native platform rules to evaluate the pathname string when searching for an existing file or creating a new file. See the *File Attributes Programming Reference Manual* for details.
1	POSIX	Use POSIX interface defined rules to evaluate the pathname string when searching for an existing file or creating a new file. See the *POSIX User's Guide* for specific information on these rules.

Real Array Input

The library procedure expects one or more real numbers. You must define three parameters:

1. A call-by-reference real array.

   Declare REAL ARRAY (with [0] bounds) and REFERENCE.
2. A call-by-value integer (<name>_OFF) that specifies a word-offset to the start of data within the array.
3. A call-by-value integer (<name>_LEN) that specifies the length (in words) of the data.

Real Array Output

The program expects one or more real numbers from the library procedure. You must define three parameters:

1. A call-by-reference real array.

   Declare REAL ARRAY (with [0] bounds) and REFERENCE.
2. A call-by-value integer (<name>_OFF) that specifies a word-offset to the start of data within the array.
3. A call-by-value integer (<name>_MAX) that specifies the length (in words) of the data.

The MCP sets an ERRNO condition if there is not enough space to store all data.

Structure Array Input

The library procedure expects a data structure. You must define three parameters:

1. A call-by-reference real array.

   Declare REAL ARRAY (with [0] bounds) and REFERENCE.
2. A call-by-value integer (<name>_OFF) that specifies a word-offset to the start of the structure.
3. A call-by-value integer (<name>_LEN) that specifies the length of the structure in words.

If the structure contains a character array, it contains a structure member to specify the character length of that array. This member appears just before the character array. Unlike EBCDIC array input, the defined length must be greater than or equal to zero.

Structures are defined later in this section.

Structure Array Output

The program expects a data structure from the library procedure. You must define three parameters:

1. A call-by-reference real array.

   Declare REAL ARRAY (with [0] bounds) and REFERENCE.

2. A call-by-value integer (<name>_OFF) that specifies a word-offset to the start of the structure.
3. A call-by-value integer (<name>_MAX) that specifies the space (in words) available to store the structure.

A structure may “grow” from release to release. For example, an 8-word structure defined in release 42.3 may be redefined as a 10-word structure in release 43.1. The structure passing mechanism supports such structure growth as follows:

- The MCP can normally store the beginning of a “large” structure in a smaller array space. It ignores the unstored portion of the large structure and does not provide an error.
- An error occurs only if a member is partially stored.

If the structure contains a character array, its length is fixed in the definition of the structure. This length is passed in a member declared just before the character array. The MCP sets an ERRNO condition if the defined length does not accommodate the character string (including a terminating null character).

Structures are defined later in this section.

Signal Handler Procedure

This input parameter (ACT_PROC) defines a procedure to be performed when the specified signal occurs. A formal declaration is made as follows:

1. The ACT_PROC parameter is declared REFERENCE and INTEGER PROCEDURE.
2. The ACT_PROC integer procedure is declared as follows:

```
INTEGER PROCEDURE ACT_PROC (INFO1, INFO2, INFO3, INFO4, INFO5,
                            INFO6, INFO7, INFO8, INFO9, INFO10);
  VALUE         INFO1, INFO2, INFO3, INFO4, INFO5,
                INFO6, INFO7, INFO8, INFO9, INFO10;
  INTEGER       INFO1, INFO2, INFO3, INFO4, INFO5,
                INFO6, INFO7, INFO8, INFO9, INFO10;
```

   Table 2–1 defines the information required in each of the required INFO*n* parameters.

The sa_handler address (defined by the INFO2 parameter) is **always** passed to the library procedure. The INFO3_SIGINFOF bit (defined in the INFO3 parameter) specifies what additional arguments should be passed.

- If INFO3_SIGINFOF is set (1), then three additional arguments are used:

Argument Rule	Description
Call-by-value integer	Signal number (INFO1_SIGNALF)
Call-by-value integer	An offset within the heap to where the SIGINFO_T structure is stored. See “SIGINFO_T structure” within this section for a description of this structure.
Call-by-value integer	0

- If INFO3_SIGINFOF is reset (0), then one additional argument is used:

Argument Rule	Description
Call-by-value integer	Signal number (INFO1_SIGNALF)

Table 2–1. ACT_PROC Procedure Parameters

Parameter	Description
INFO1	An integer input parameter specifying the version and signal type (the current version value is 0). This word breaks down as follows: [23:08] INFO1_VERSIONF [07:08] INFO1_SIGNALF INFO1_SIGNALF corresponds to the si_signo member of the SIGINFO_T structure.
INFO2	An integer input parameter containing the value of the sa_handler word of the parent procedure's ACT array. See "DISP parameter" later in this section for a description of possible values.
INFO3	An integer input parameter containing additional information about the signal state. This word breaks down as follows: [37:01] INFO3_SIGINFOF [36:01] INFO3_HARDWAREGENF [35:12] INFO3_CODEF [23:24] INFO3_ERRF INFO3_CODEF corresponds to the si_code member of the SIGINFO_T structure. INFO3_ERRF corresponds to the si_errno member of the SIGINFO_T structure.
INFO4	An integer input parameter containing the process ID of the process causing the signal. This parameter corresponds to the si_pid member of the SIGINFO_T structure.
INFO5	An integer input parameter containing the user ID. This parameter corresponds to the si_uid member of the SIGINFO_T structure.
INFO6	Not used.
INFO7	Not used.
INFO8	Not used.
INFO9	Not used.
INFO10	Not used.

ERRNO

ERRNO must be declared as the last item in the formal parameter list of all POSIX interface related functions and library procedures. It is declared as a call-by-reference integer.

If an error occurs during the execution of a procedure, the calling process is normally notified in two ways:

- The procedure returns an error result, usually –1 (see "Result (integer or real)" later in this section)
- ERRNO is set to some non-zero code to identify the error

There are two ways to determine the meaning of a non-zero ERRNO code:

- See Table 2–2 for descriptions of ERRNO codes.
- Use the STRERROR function to obtain a string of descriptive text about a specified ERRNO value. The STRERROR function is described in Section 3.

Note: *Only general descriptions are provided in Table 2–2 and in the returned STRERROR function text. However, in many cases there is a function-specific meaning for the error code. These function-specific meanings are described in the* C Programming Reference Manual, Volume 2: Headers and Functions.

Table 2–2. ERRNO Descriptions

ERRNO Code	Error	Description
0	EOK	No error.
1	EDOM	Domain error. An input parameter was outside the domain of the mathematical function.
2	ERANGE	Result too large. The result was too large to fit in the available space.
3	EASSERT	An assert failure occurred in the file.
4	EHEAPERR	Dynamic memory allocation area (heap) was corrupted.
5	ESIGNALERR	Invalid signal value.
6	EHEAPFULL	Heap was full.

Table 2–2. ERRNO Descriptions

ERRNO Code	Error	Description
20	EMFILE	Too many open files. An attempt was made to open more than the maximum number of file descriptors allowed for this process. The maximum number of open file descriptors is defined by OPEN_MAX.
21	EINVLDMODE	Invalid mode specified.
22	EINVLDNAME	Invalid file name.
23	ENOENT	No such file or directory. A component of a specified pathname did not exist or the pathname was an empty string.
24	EACCES	Access permission denied. An attempt was made to access a file in a manner forbidden by its file access permissions.
25	EFILENOTAVAIL	A file was not available.
26	EFILEOPENERR	An error occurred while opening a file.
27	EFILERO	An attempt was made to write to a read-only file.
28	EFILEWO	An attempt was made to read from a write-only file.
29	EFILEPOSREQ	A file positioning operation is required.
30	EBADF	Bad file descriptor. A file descriptor parameter was out of range, did not refer to an open file, or a read (write) request was made to a file that was only open for writing (reading).
31	EIO	I/O error. Some physical input or output error occurred. This error may have occurred on a previous operation involving the current file descriptor.
32	EDATAERR	I/O data error.
33	EPARITYERR	I/O parity error.
34	EATTRLISTERR	A syntax error occurred in a file attribute list.
35	EINVLDATTR	Invalid file attribute.
36	EATTRRO	An attempt was made to set a read-only file attribute.

Table 2–2. ERRNO Descriptions

ERRNO Code	Error	Description
37	EINVLDATTRVAL	Invalid file attribute value.
38	EATTRERR	An error occurred when setting a file attribute.
39	ENOFILEPOS	File did not support positioning requests.
40	EFILECLOSEERR	An error occurred while closing a file.
41	EFTELLTOOLARGE	The ftell result was too large.
42	ENOSORTRESTART	The restart request was not for a disk only SORT/MERGE.
43	EINVALSORTVER	Inconsistent SORT/MERGE version.
44	EBADSORTRECLEN	SORT/MERGE was unable to determine record length.
45	EBADMERGEINPUTS	MERGE requires at least 2 but no more than 8 inputs.
46	EENDOFFILEERR	An attempt was made to write beyond the end of a file.
47	ENOHOST	Unreachable or unknown host was specified.
78	ENAMETOOLONG	Filename too long. The size of a pathname string or a pathname component exceeded the specified maximum. The pathname string maximum is defined by PATH_MAX and the pathname component maximum is defined by NAME_MAX.
82	ENOTSUP	Not supported.
83	EMSGSIZE	Inappropriate message buffer length.
84	EIOLOGIC	Internal I/O logic error.
85	EBADMSG	An unreadable message was sent.
86	ETIME	Timer expired.
87	ESPIPE	Invalid seek. A seek operation was attempted on a pipe or FIFO.
88	EROFS	Read-only file system. An attempt was made to modify a directory or file within a file system marked as read-only.

Table 2–2. ERRNO Descriptions

ERRNO Code	Error	Description
89	ENOSYS	Function not implemented. An attempt was made to use a function that is not available.
90	ELOOP	Too many symbolic links.
91	EPIPE	Broken pipe or FIFO. A write was attempted to a pipe or FIFO; however, no process was ready to read this data.
92	ENOSPC	No space left on device. The device did not have enough free space to allow a write operation or the extension of a directory.
93	ENOTEMPTY	Directory not empty. A directory with entries other than dot and dot-dot was supplied when an empty directory was expected.
94	ENOLCK	No locks available. The system has reached its predefined limit for simultaneous file and record locks. The request to lock another object cannot be honored at this time.
95	EBADSIG_ASERIES	The code improperly attempted to modify the signal environment. A SIGPUSH function must precede this attempt.
96	EMLINK	Too many file links. An attempt was made to establish a file link and the link count for the file would exceed a specified maximum. This maximum is defined by LINK_MAX.
97	ENOMSG	No message is available in the message queue.
98	EIDRM	ID was removed.
99	EDEADLK	Resource deadlock avoided. An attempt was made to get a lock that would have resulted in a deadlock situation.
100	EINPROGRESS	Operation in progress.

Table 2–2. ERRNO Descriptions

ERRNO Code	Error	Description
101	EPERM	Operation not permitted. The calling process did not have appropriate privileges or was not the owner of a defined file or other resource.
102	ECANCELED	Operation canceled.
103	ESRCH	No such process. Could not find a process that corresponds with the specified process ID.
104	EINTR	Interrupted function call (system service). An asynchronous signal (such as SIGINT or SIGQUIT) was caught during the execution of an interruptable function.
106	ENXIO	No such device or address. An I/O operation referred to a device that does not exist or is not ready (for instance, in an off-line state). The error is also set if a request is made beyond the limits of the device.
107	E2BIG	Argument list too long. The sum of the number of bytes used by a new process image argument list and environment list is greater than the system-imposed limit.
108	ENOEXEC	Exec format error. An attempt was made to execute a code file that was not valid for this implementation or took too many or the wrong type of parameters.
110	ECHILD	No child processes. A WAITP or WAITPID function was executed by a process without either of the following: • An existing child process • A terminated child process with unreported status
111	EAGAIN	Resource temporarily unavailable (later calls to this procedure may perform normally).

Table 2–2. ERRNO Descriptions

ERRNO Code	Error	Description
112	ENOMEM	Not enough space. A new process image required more memory than is available. This error is returned only if the situation is permanent. If the memory shortage is temporary, an ERRNO value of 111 (EAGAIN) is returned.
114	EFAULT	Bad address. The system detected an invalid index to a data structure.
115	ENOTBLK	Not block device. The requested function required a block device.
116	EBUSY	Resource busy. An attempt was made to use a system resource that was not available because it was being used by another process.
117	EEXIST	File exists. An existing file was specified in an inappropriate context.
118	EXDEV	Improper link. An attempt was made to link to a file on another file system.
119	ENODEV	No such device. An attempt was made to perform an inappropriate function to a device (e.g., an attempt to read data from a printer).
120	ENOTDIR	Not a directory. A specified pathname contained a component that was not a directory; a directory was expected.
121	EISDIR	Is a directory. An attempt was made to open a directory with write mode specified.
122	EINVAL	Invalid argument. An invalid parameter (argument) was specified.

Table 2–2. ERRNO Descriptions

ERRNO Code	Error	Description
123	ENFILE	Too many open files on system. The system has reached its predefined limit for simultaneously open files. The request to open another file cannot be honored at this time.
125	ENOTTY	Inappropriate I/O control operation. An inappropriate I/O control function was attempted on a file or special file.
126	ETXTBSY	Illegal code file access (text file busy). An attempt was made to open a code file with write access.
127	EFBIG	File too large. An attempt was made to expand a file to a length that would exceed its maximum size.

Result (Integer or Real)

Upon completion, almost all functions return an integer or real number *result*. The value returned indicates whether the function was successful. Depending on the function performed, it may also represent requested data. Actual returned values depend on the function being performed.

The result is not passed as a formal parameter. It is not mentioned in procedure declarations.

Refer to the *C Programming Reference Manual*, *Volume 2: Headers and Functions*, for appropriate information. For each function description, the Returns subsection describes the meaning of all possible result codes.

In most cases, the following rules apply:

Returned Value	Meaning
0	Function was successful.
–1	An error occurred. The ERRNO parameter contains the applicable error code.
Other values	Function-specific information.

Parameter Values and Structures

The remainder of this section provides reference information associated with specific POSIX function or library procedure parameters. This information includes:

- Integer parameter definitions

  These definitions include integer values and names defined in the ALGOL include file.
- Structure definitions

  These definitions include a description of every structure member.

Information appears alphabetically by parameter or structure name.

AMODE Parameter

ALGOL Function or Library Procedure Reference

AMODE is associated with the

- ACCESS function
- POSIX_ACCESS library procedure

Description

The following table lists valid AMODE parameter integers and associated defined names. The defined names are valid only with the ACCESS function.

Integer	Defined Name (Include File)	Description
0	F_OK	File existence
1	X_OK	Execute or search permission
2	W_OK	Write permission
4	R_OK	Read permission

CMD Parameter (FCNTL)

ALGOL Function or Library Procedure Reference

The POSIX_FCNTL library procedure uses this form of the CMD parameter.

Description

The following table lists valid CMD parameter integers and associated defined names.

Integer	Defined Name (Include File)	Description
0	F_DUPD	Duplicate file descriptor. Return duplicated file descriptor value in result.
1	F_GETFD	Return current value of file descriptor flag in result.
2	F_SETFD	Set file descriptor flag as defined by the INTARG parameter.
3	F_GETFL	Return current value of file status flags in result.
4	F_SETFL	Set file status flags as defined by the INTARG parameter.
5	F_GETLK	Get first lock that blocks the lock described by the FLOCK structure.
6	F_SETLK	Set or clear the lock specified by the FLOCK structure – do not wait.
7	F_SETLKW	Set or clear the lock specified by the FLOCK structure – if necessary, wait to set.

CMD parameter (SEMCTL)

ALGOL Function or Library Procedure Reference

CMD is associated with the

- SEMCTL function
- MCPX_SEMCTL library procedure

Description

The following table lists valid CMD parameter integers and associated defined names. The defined names are valid only with the ALGOL SEMCTL function.

Integer	Defined Name (Include File)	Description
0	IPC_STAT	Copy semaphore values into words 20 through 23 of the structure defined by the ARG* parameters.
1	IPC_SET	Set the SEMID's SEMID_DS data structure to the values defined in the structure defined by the ARG* parameters.
2	IPC_RMID	Remove the semaphore identifier specified by SEMID from the system and destroy the semaphores and SEMID_DS data structure associated with it.
3	SEM_GETNCNT	Return the value of SEMNCNT.
4	SEM_GETPID	Return the value of SEMPID.
5	SEM_GETVAL	Return SEMVAL value.
6	SEM_GETALL	Return all SEMVAL values in the semaphore set. Put these values in the array specified by the ARG* parameters.
7	SEM_GETZCNT	Return the value of SEMZCNT.
8	SEM_SETVAL	Set the value of SEMVAL to the value contained in the VAL parameter.
9	SEM_SETALL	Set the SEMVAL values according to the values contained in the array specified by the ARG* parameters.

DISP Parameter

ALGOL Function or Library Procedure Reference

DISP is associated with the

- SIGNAL function
- SIGSET function
- POSIX_SIGHANDLER library procedure

Description

The following table lists valid DISP parameter integers and associated defined names. The defined names are valid only with the SIGNAL and SIGSET functions.

Integer	Defined Name (Include File)	Description
Any positive value	–	Address of a signal-catching function.
–1	SIG_ERR	Indicates an error condition.
–2	SIG_DFL	Use default signal handling function.
–3	SIG_IGN	Ignore signal.
–4	SIG_HOLD	Add signal type to process signal mask.

FLOCK Structure

ALGOL Function or Library Procedure Reference

The POSIX_FCNTL library procedure passes the FLOCK structure.

Description

The following table defines the contents of the FLOCK structure.

Word Offset	Member	Description

0	L_TYPE	Type of lock. Possible values are:
		
		F_RDLCK = 1 Reader lock
		F_WRLCK = 2 Writer lock
		F_UNLCK = 3 Unlock (or not locked)

1	L_WHENCE	Specifies where to apply offset L-START. Valid codes are: SEEK_START = 0 Apply offset from beginning of file. SEEK_START = 1 Apply offset from current file pointer position. SEEK_START = 2 Apply offset from EOF (byte past last written).
2	L_START	Specifies relative offset (in bytes).
3	L_LEN	Specifies length of area (in bytes). A value of 0 indicates that the area runs to EOF.
4	L_PID	Specifies the process ID of the lock holder. This field is only valid as the F_GETLK command output parameter.
5	L_PAD1	–

GROUP Structure

ALGOL Function or Library Procedure Reference

The GROUP structure is passed by the

- GETGRGID function
- GETGRNAM function
- POSIX_GETGRINFO library procedure

Description

The following table defines the contents of the GROUP structure.

Word Offset	Member	Description
0	GR_NAME_LEN	Groupname length
1–3	GR_NAME	Groupname string (includes terminating null character)
4	GR_GID	Group ID
5	GR_MEM_TOTAL	Number of users in member list.
6 to end	GR_MEM	Group member list string (includes terminating null character)

HOW Parameter

ALGOL Function or Library Procedure Reference

HOW is associated with the

- SIGPROCMASK function
- POSIX_SIGHANDLER library procedure

Description

The following table lists valid HOW parameter integers and associated defined names. The defined names are valid only with the SIGPROCMASK function.

Integer	Defined Name (Include File)	Description
1	SIG_BLOCK	Add signals in the set specified by the SET parameter to the signal mask.
2	SIG_UNBLOCK	Remove signals in the set specified by the SET parameter from the signal mask.
3	SIG_SETMASK	Replace the current signal mask with the signals in the set specified by the SET parameter.
–1	SIG_ENQUIRE	Retrieve the current signal mask.

INFO Parameter

ALGOL Function or Library Procedure Reference

The POSIX_SETIDS library procedure uses the INFO parameter.

Description

The following table defines INFO parameter bit assignments for the umask( ) function (SELECTOR parameter value of 5). For other functions, the INFO parameter contains a user ID, group ID, or process group ID.

Bit(s)	Name	Description
[08:09]		File Mode Creation Mask:
[08:03]	S_IRWXOF	
[08:01]	S_IRUSRF	Mask out S_IRUSRF bit (file owner class read permission).
[07:01]	S_IWUSRF	Mask out S_IWUSRF bit (file owner class write permission).
[06:01]	S_IXUSRF	Mask out S_IXUSRF bit (file owner class search or execute permission).
[05:03]	S_IRWXGF	
[05:01]	S_IRGRPF	Mask out S_IRGRPF bit (file group class read permission).
[04:01]	S_IWGRPF	Mask out S_IWGRPF bit (file group class write permission).
[03:01]	S_IXGRPF	Mask out S_IXGRPF bit (file group class search or execute permission).
[02:03]	S_IRWXOF	
[02:01]	S_IROTHF	Mask out S_IROTHF bit (file other class read permission).
[01:01]	S_IWOTHF	Mask out S_IWOTHF bit (file other class write permission).
[00:01]	S_IXOTHF	Mask out S_IXOTHF bit (file other class search or execute permission).

Note: When a permission bit is masked out, it CANNOT be set when the file is created.

INTARG Parameter (FCNTL)

ALGOL Function or Library Procedure Reference

The POSIX_FCNTL library procedure uses this form of the INTARG parameter.

Description

INTARG provides supporting data for three POSIX_FCNTL library procedure functions (the CMD parameter specifies the function to be performed). The following table describes INTARG parameter usage for these functions.

Command	Function Provided by INTARG	INTARG Format
F_DUPD	Specify lowest file descriptor from which to search for available value.	File descriptor integer value.
F_SETFD	Specify the file descriptor flags to set.	File descriptor flags. Currently, only one flag is valid: [00:01] FD_CLOSEXEC
F_SETFL	Specify the file status flags to set. (Ignore Access Mode bits for this operation.)	[04:01] O_SYNCF [03:01] O_APPENDF [02:01] O_NONBLOCKF [01:02] O_ACCMODEF 0 = O_RDONLY 1 = O_WRONLY 2 = O_RDWR

MODE Parameter

ALGOL Function or Library Procedure Reference

SEMFLG is associated with the

- SEM_OPEN function
- POSIX_SEM_OPEN library procedure

Description

The following table defines MODE parameter bit assignments.

Bit(s)	Name	Description
[08:09]		Semaphore access permissions:
[08:01]	S_IRUSRF	Read by owner.
[07:01]	S_IWUSRF	Alter by owner.
[05:01]	S_IRGRPF	Read by group.
[04:01]	S_IWGRPF	Alter by group.
[02:01]	S_IROTHF	Read by others.
[01:01]	S_IWOTHF	Alter by others.

MCPSTAT Structure

ALGOL Function or Library Procedure Reference

The MCPSTAT structure is passed by the

- MCPSTAT function
- POSIX_FILESTATUS library procedure

Description

The following table defines the contents of the MCPSTAT structure.

Word Offset	Member	Description
0–13	STAT structure	See STAT structure description in this section.
Platform-Based Extensions		
14	MST_FAMINDEX_INX	Specifies family index.
15	MST_USERCODELEN_INX	Specifies length (in bytes) of MST_USERCODE.
16–18	MST_USERCODE_INX	Specifies usercode value.
19	MST_GROUPCODELEN_INX	Specifies length (in bytes) of MST_GROUPCODE.
20–22	MST_GROUPCODE_INX	Specifies groupcode value.
23	MST_FAMNAMELEN_INX	Specifies length (in bytes) of MST_FAMNAME
24–26	MST_FAMNAME_INX	Specifies family name value.
27	MST_HOSTNAMELEN_INX	Specifies length (in bytes) of MST_HOSTNAME.
28–30	MST_HOSTNAME_INX	Specifies host name value.
For Future Growth		
31	MST_PAD1_INX	
32	MST_PAD2_INX	
33	MST_PAD3_INX	
34	MST_PAD4_INX	

NAME Parameter (PATHCONF)

ALGOL Function or Library Procedure Reference

NAME is associated with the

- PATHCONF function
- POSIX_PATHCONF library procedure

Description

The following table lists valid NAME parameter integers and associated defined names. The defined names are valid only with the PATHCONF function.

Integer	Defined Name (Include File)	Description
1	PC_LINK_MAX	Return the maximum value of a file link count. If path refers to a directory, then this value is for the entire directory.
2	PC_MAX_CANON	Return the maximum number of bytes in a terminal canonical input line. The path must refer to a terminal.
3	PC_MAX_INPUT	Return the maximum number of bytes for which space will be available in an input queue. The path must refer to a terminal.
4	PC_NAME_MAX	Return the maximum length of a filename for this directory (exclusive of terminating null character).
5	PC_PATH_MAX	Return the maximum length of a relative pathname when this directory is the working directory (exclusive of terminating null character).
6	PC_PIPE_BUF	Return the maximum number of bytes that a process can write to a pipe without interruption.
7	PC_CHOWN_RESTRICTED	Return a value other than –1 if use of the CHOWN (or chown( )) function on this file is restricted. If the specified path refers to a directory, this restriction applies to all files in the directory. See the *POSIX User's Guide* for details about the CHOWN_RESTRICTED symbolic constant.
8	PC_NO_TRUNC	Return a value other than –1 if a pathname component longer than 17 characters (the NAME_MAX value) will cause an error. (Return –1 if pathname truncation is allowed.)
9	PC_VDISABLE	Return the value used to disable special character processing for the specified terminal file. (The value is 0 for this implementation.)

NAME Parameter (SYSCONF)

ALGOL Function or Library Procedure Reference

NAME is associated with the

- SYSCONF function
- POSIX_SYSCONF library procedure

Description

The following table lists valid NAME parameter integers and associated defined names. The defined names are valid only with the SYSCONF function.

Integer	Defined Name (Include File)	Description
1	SC_ARG_MAX	Return the maximum length of combined argument and environment list associated with the EXECVE function.
2	SC_CHILD_MAX	Return the maximum number of child processes allowed for a process.
3	SC_CLK_TCK	Return the number of clock ticks per second (3255 for the POSIX interface on this platform).
4	SC_NGROUPS_MAX	Return the maximum number of simultaneous supplementary group IDs per process.
5	SC_OPEN_MAX	Return the maximum number of files that a process can have open.
6	SC_JOB_CONTROL	Return a non-zero value if the system supports job control.
7	SC_SAVED_IDS	Return a non-zero value if the system saves user IDs and group IDs when a EXECVE function occurs.
8	SC_VERSION	Return version of the POSIX interface supported. Currently, this value is 199008 (for August 1990).
9	SC_PAGESIZE	Return the page size (in bytes) of host system.
10	SC_ADDRESS_MAX	Return the maximum array size (in bytes) of host system.
14	SC_TZNAME_MAX	Return the maximum size (in bytes) supported for the name of a time zone.

OPTION Parameter (CLOSE)

ALGOL Function or Library Procedure Reference

The POSIX_CLOSE library procedure uses this form of the OPTION parameter.

Description

File closing OPTION values are listed in the following table. The specified value is passed to the FIBCLOSE routine; it is effective only if the **last** file descriptor referencing the open file description is being closed. For option descriptions, see the CLOSE statement description within the *ALGOL Programming Reference Manual*, *Volume 1: Basic Implementation.*

Integer	Option	Description
0	PCO_NORMALV	No close option
1	PCO_LOCKV	This option is equivalent to the existing (non-POSIX) LOCK option.
2	PCO_PURGEV	This option is equivalent to the existing (non-POSIX) PURGE option.
3	PCO_CRUNCHV	This option is equivalent to the existing (non-POSIX) CRUNCH option.
4	PCO_DOWNSIZEV	This option causes the file's area length to be reduced if: • It is not opened by another program. • Neither AREASIZE nor AREALENGTH was set explicitly. • Unused space is greater than a percentage of the currently allocated area. The PCO_DOWNSIZEV option is not yet supported.
5	PCO_RETAINV	This option implements the existing (non-POSIX) concept of "Close with Retention." (The program retains the file descriptor and FIB after it closes the file.)

OPTION Parameter (OPEN)

ALGOL Function or Library Procedure Reference

The POSIX_OPEN library procedure uses this form of the OPTION parameter.

Description

File opening OPTION values are listed in the following table. The indicated OPTION values are passed to the FIBOPEN routine when opening or creating a file. For more detailed descriptions of these options, see the OPEN statement description in the *ALGOL Programming Reference Manual*, *Volume 1: Basic Implementation*.

Bit(s)	Option	Description
[38:01]	PO_TRUNCATEF	When set, this flag causes an opened existing file to be truncated to a length of zero under certain conditions. Basically, truncation occurs if FILEUSE is either OUT or IO; there is no duplicate file; no other process has the file open; and the file is not a code file.
[07:08]	PO_OPENTYPEF	This field contains a value that specifies one of the following encoded open types: 0 POO_WAITV This option is equivalent to the existing (non-POSIX) WAIT option. 1 POO_ATENDV This option is equivalent to the existing (non-POSIX) ATEND option. 2 POO_AVAILABLEV This option is equivalent to the existing (non-POSIX) AVAILABLE option.

Bit(s)	Option	Description
[07:08] (cont.)	PO_OPENTYPEF (cont.)	**Note:** *For the POO_CONDITIONALV and POO_MUSTBENEWV options, existing filenames are searched under the process's usercode (if any) in the specified family. If family substitution applies, only the primary family is searched. The search is restricted to resident files and does not consider archive or catalog backups.* 3 POO_CONDITIONALV This option modifies the open process as follows: If NEWFILE is true and the file already exists, open the existing file instead of creating a new file. If NEWFILE is false (or unspecified) and the file does not exist, return an open error message without issuing a "NO FILE" RSVP operator message. 4 POO_MUSTBENEWV This option modifies the open process. If NEWFILE is true and the file already exists, an error is returned and a new file is not created. 5 POO_OFFERV This option is equivalent to the existing (non-POSIX) OFFER option.

OPTION Parameter (WAITPID)

ALGOL Function or Library Procedure Reference

OPTION is associated with the

- WAITPID function
- POSIX_WAITPID library procedure

Description

The following table lists valid OPTION parameter integers and associated defined names. The defined names are valid only with the WAITPID function.

Bit	Defined Name (Include File)	Description
[05:01]	WNOHANGF	When set, do NOT suspend the calling process to wait for terminated or stopped child processes. Instead, report an ECHILD error condition if there are no such processes. When not set, suspend calling process until terminated or stopped child process status is available.
[02:01]	WUNTRACEDF	When set, report status of both stopped and terminated child processes. When not set, report only terminated child process status.

PASSWD Structure

ALGOL Function or Library Procedure Reference

The PASSWD structure is passed by the

- GETPWGID function
- GETPWNAM function
- POSIX_GETPWINFO library procedure

Description

The following table defines the contents of the PASSWD structure.

Word Offset	Member	Description
0	PW_NAME_LEN	Specifies usercode length.
1–3	PW_NAME	Specifies usercode string (includes terminating null character).
4	PW_UID	Specifies user ID.
5	PW_GR_NAME_LEN	Specifies groupname length.
6–8	PW_GR_NAME	Specifies groupname length (includes terminating null character).
9	PW_GID	Specifies group ID.
10	PW_DIR_LEN	Specifies initial working directory length.
11–53	PW_DIR	Specifies initial working directory string (includes terminating null character).
54	PW_COMMENT_LEN	Specifies user identity length.
55–309	PW_COMMENT	Specifies user identity string (includes terminating null character).
310	PW_SHELL_LEN	Specifies initial user program length.
311–353	PW_SHELL	Specifies initial user program string (includes terminating null character).

PID parameter (KILL)

ALGOL Function or Library Procedure Reference

PID is associated with the

- KILL function
- POSIX_SIGNALHANDLER library procedure

Description

The following table lists valid PID parameter values and the function associated with each.

Integer	Function
>0	Send signal to the process that has a process ID equal to PID.
0	Send signal to all processes that: • Have a process group ID equal to the calling process's process group ID. • Are NOT system processes.
–1	Not specified.
<–1	Send signal to all processes that: • Have a process group ID equal to the absolute value of PID. • Are NOT system processes.

PID Parameter (WAITPID)

ALGOL Function or Library Procedure Reference

PID is associated with the:

- WAITPID function
- POSIX_WAITPID library procedure

Description

The following table lists valid PID parameter integers and associated functions. The defined name (PIDANYV) is valid only with the WAITPID function.

Integer	Defined Name (Include File)	Function
>0	–	Accept status only from the specified child process.
0	–	Accept status from any child process that has the same process group ID as the calling process.
–1	PIDANYV	Accept status for any terminated or stopped child process.
<–1	–	Accept status from any child process that has the process group ID specified by the absolute value.

SEMBUF Structure

ALGOL Function or Library Procedure Reference

The SEMBUF structure is passed by the

- SEMOP function
- MCPX_SEMOP library procedure

Description

The following table defines the contents of the SEMBUF structure.

Word Offset	Member	Description
0	SEM_NUM	Specifies the semaphore number.
1	SEM_OP	Specifies the semaphore operation.
2	SEM_FLG	Specifies semaphore operation flags. Valid flag values: [12:01] SEM_UNDO (set up adjust on exit entry) [16:01] IPC_CREAT (create entry if key does not exist) [17:01] IPC_EXCL (fail if key exists) [18:01] IPC_NOWAIT (return error if request must wait)

SEMFLG Parameter

ALGOL Function or Library Procedure Reference

SEMFLG is associated with the:

- SEMGET function
- MCPX_SEMGET library procedure

Description

The following table defines valid SEMFLG parameter bit assignments and their meaning. Defined names are valid only with the SEMGET function.

Bit(s)	Defined Name	Description
[17:01]	IPC_EXCL	Exclusive usage flag.
[16:01]	IPC_CREAT	Create a semaphore if the key does not exist.
[08:09]	–	Semaphore access permissions:
[08:01]		Read by owner.
[07:01]		Alter by owner.
[05:01]		Read by group.
[04:01]		Alter by group.
[02:01]		Read by others.
[01:01]		Alter by others.

SEMID_DS Structure

ALGOL Function or Library Procedure Reference

The SEMID_DS structure is passed by the

- SEMCTL function
- MCPX_SEMCTL library procedure

Description

The following table defines the contents of the SEMID_DS structure.

Word Offset	Member	Description
0	SEMID_DS_UID	Specifies semaphore owner user ID.
1	SEMID_DS_GID	Specifies semaphore owner group ID.
2	SEMID_DS_CUID	Specifies semaphore creator user ID.
3	SEMID_DS_CGID	Specifies semaphore creator group ID.
4	SEMID_DS_MODE	Specifies semaphore MODE flags (defined by SEMFLG parameter).
5	SEMID_DS_UID_L	Specifies semaphore owner usercode length.
6–8	SEMID_DS_UID_S	Specifies semaphore owner usercode string (includes terminating null character).
9	SEMID_DS_GID_L	Specifies semaphore owner groupcode length.
10–12	SEMID_DS_GID_S	Specifies semaphore owner groupcode string (includes terminating null character).
13	SEMID_DS_CUID_L	Specifies semaphore creator usercode length.
14–16	SEMID_DS_CUID_S	Specifies semaphore creator usercode string (includes terminating null character).
17	SEMID_DS_CGID_L	Specifies semaphore creator groupcode length.
18–20	SEMID_DS_CGID_S	Specifies semaphore creator groupcode string (includes terminating null character).
21–23	SEMID_DS_PAD1_3	Padding
24	SEMID_DS_NSEMS	Specifies number of semaphores in set.
25	SEMID_DS_OTIME	Specifies time of last SEMOP.
26	SEMID_DS_CTIME	Specifies time of last creation.
27–29	SEMID_DS_PAD2_3	Padding

SIG Parameter

ALGOL Function or Library Procedure Reference

SIG is associated with the

- KILL function
- RAISE function
- SIGACTION function
- SIGADDSET function
- SIGDELSET function
- SIGHOLD function
- SIGIGNORE function
- SIGISMEMBER function
- SIGNAL function
- SIGPAUSE function
- SIGRELSE function
- SIGSET function
- POSIX_SIGHANDLER library procedure

Description

The following table indicates the signal type associated with each SIG parameter value.

Integer	Defined Name	Signal Function
01	SIGHUP	Hang up on controlling terminal.
02	SIGINT	Interactive Attention signal.
03	SIGQUIT	Interactive Termination signal.
04	SIGILL	Illegal hardware operation or bad stack arguments.
05	SIGTRAP	Trace trap.
06	SIGABRT	Abnormal Termination signal.
07	SIGEMT	Emulator Trap instruction.
08	SIGFPE	Erroneous arithmetic operation.
09	SIGKILL	Termination signal.
10	SIGBUS	Bus error.
11	SIGSEGV	Invalid memory reference.
12	SIGSYS	Bad system service argument without ERRNO.
13	SIGPIPE	Write on pipe with no reader.
14	SIGALRM	Time-out signal.
15	SIGTERM	Termination signal.
16	SIGUSR1	Reserved for user-defined signal number 1.
17	SIGUSR2	Reserved for user-defined signal number 2.
18	SIGCHLD	Child process terminated or stopped.
19	SIGPWR	Immediate scheduled shutdown.
20	SIGWINCH	Window change.
22	SIGPOLL	Pending selectable event on a stream.
23	SIGSTOP	Stop signal.
24	SIGTSTP	Interactive Stop signal.
25	SIGCONT	Continue if stopped.
26	SIGTTIN	Background process tried read from terminal.
27	SIGTTOU	Background process tried write to terminal.

SIGACTION Structure

ALGOL Function or Library Procedure Reference

The SIGACTION structure is passed by the:

- SIGACTION function
- POSIX_SIGHANDLER library procedure

Description

The following table defines the contents of the SIGACTION structure.

Word Offset	Member	Description
0	sa_handler	Specifies a signal action. This action is passed to the ACT_PROC procedure as the INFO2 parameter. See “DISP parameter” in this section for a description of allowable values.
1	sa_mask	Specifies the mask of signals to be blocked when the signal-catching function is executed.
2	sa_flag	Specifies flags that affect the behavior of the signal.

SIGINFO_T Structure

ALGOL Function or Library Procedure Reference

The SIGINFO_T structure is passed by the ACT_PROC procedure. ACT_PROC is used by the

- SIGACTION function
- SIGNAL function
- SIGSET function
- POSIX_SIGHANDLER library procedure

Description

The following table defines the contents of the SIGINFO_T structure.

Word Offset	Member	Description
0	si_signo	Signal number.
1	si_errno	If non-zero, an ERRNO value. For additional information about this error, see “ERRNO” within this section.
2	si_code	Signal code. Corresponds with INFO3_CODEF in INFO3 parameter of the ACT_PROC procedure.
3	si_pid	Process ID of the process causing the signal. Corresponds with INFO3 parameter of the ACT_PROC procedure.
4	si_uid	User ID of the process causing the signal. Corresponds with INFO4 parameter of the ACT_PROC procedure.
5	si_status	Exit value or signal.
6	si_band	Band event for POLL-IN, POLL-OUT, or POLL-MSG. Not yet supported.

STAT Structure

ALGOL Function or Library Procedure Reference

The STAT structure is passed by the

- STAT function
- POSIX_FILESTATUS library procedure

Description

The following table defines the contents of the STAT structure.

Word Offset	Member	Description
0	ST_MODE_IX	Contains the File Mode. See Table 2–3 for a detailed description.
1	ST_INO_INX	Specifies the file serial number (not yet supported).
2	ST_DEV_INX	Specifies the device ID.
3	ST_NLINK_INX	Specifies the number of links.
4	ST_UID_INX	Specifies the owner's user ID.
5	ST_GID_INX	Specifies the owner's group ID.
6	ST_SIZE_INX	Specifies the file size (in bytes).
7	ST_ATIME_INX	Specifies the last file access time.
8	ST_MTIME_INX	Specifies the last file modification time.
9	ST_CTIME_INX	Specifies the last file status change time.
UNIX-Based Extensions		
10	ST_BLKSIZE_INX	Specifies the block size (in bytes).
11	ST_BLOCKS_INX	Specifies the number of blocks.
For Future Growth		
12	ST_PAD1_INX	
13	ST_PAD2_INX	

Table 2–3. File Mode Layout (STAT Structure Word 0)

Bit(s)	Defined Name	Description
[28:01]	S_TEMPFILEF	When set, file is temporary.

Word Offset	Member	Description
[27:04]	S_IFMTF	Encoded file type: 1 = S_IFIFO FIFO special 2 = S_IFCHR Character special 4 = S_IFDIR Directory 6 = S_IFBLK Block special 8 = S_IFREG Regular file 10 = S_IFLNK Symbolic link
[23:04]	S_DIRECTORYKINDF	Encoded directory type: 0 = Traditional 1 = POSIX
[13:14]	S_IFAMODEF	File access mode bits:
[13:01]	S_IGUARDUSRF	Guard file permissions also apply to file owner.
[12:01]	S_IUSEGUARDF	Check guard file permissions.
[11:01]	S_ISUIDF	Upon execution of this file, set usercode of process to the file owner's usercode.
[10:01]	S_ISGIDF	Upon execution of this file, set groupcode of process to the file's groupcode.
[09:01]	–	Not used.
[08:03]	S_IRWXUF	File owner permissions:
[08:01]	S_IRUSRF	Read permission.
[07:01]	S_IWUSRF	Write permission.
[06:01]	S_IXUSRF	Execute or search permission.
[05:03]	S_IRWXGF	File group permissions:
[05:01]	S_IRGRPF	Read permission.
[04:01]	S_IWGRPF	Write permission.
[03:01]	S_IXGRPF	Execute or search permission.
[02:03]	S_IRWXOF	File other permissions:
[02:01]	S_IROTHF	Read permission.
[01:01]	S_IWOTHF	Write permission.
[00:01]	S_IXOTHF	Execute or search permission.

STATUS Parameter

ALGOL Function or Library Procedure Reference

STATUS is associated with

- WAITP function
- WAITPID function
- POSIX_WAITPID library procedure

Description

The STATUS parameter (or STAT_LOC parameter of the POSIX_WAITPID library procedure) references the memory location where a child process's status word is stored. This word indicates appropriate stopped or termination status for a child process.

You can analyze the termination word as follows:

If bits [07:08] are . . .	and bits [15:08] are . . .	Then . . .
0	–	The child process terminated normally. Bits [15:08] specify the status value provided by the exit( ) function.
Greater than 0	0	The child process terminated abnormally. Bits [06:07] indicate the signal that caused termination (refer to "SIG parameter" in this section).
0x7F	Greater than 0	The child process is stopped. Bits [15:08] indicate the signal that caused termination (refer to "SIG parameter" in this section).

TMS Structure

ALGOL Function or Library Procedure Reference

The TMS structure is passed by the

- TIMES function
- POSIX_TIMES library procedure

Description

The following table defines the contents of the TMS structure.

Word Offset	Member	Description
0	TMS_UTIME	Specifies the CPU time (in ticks) required for execution of user instructions by this process.
1	TMS_STIME	Specifies the CPU time (in ticks) used by the system on behalf of this process.
2	TMS_CUTIME	Specifies the accumulated TMS_UTIME and TMS_CUTIME for all child processes.
3	TMS_CSTIME	Specifies the accumulated TMS_STIME and TMS_CSTIME for all child processes.
4	TMS_ITIME	Specifies the I/O time (in ticks) for this process.
5	TMS_CITIME	Specifies the accumulated I/O time (in ticks) for all child processes.

UTSNAME Structure

ALGOL Function or Library Procedure Reference

The UTSNAME structure is passed by the

- UNAME function
- POSIX_UNAME library procedure

Description

The following table defines the contents of the UTSNAME structure. Each member is a string of up to 72 characters terminated with a null character.

Word Offset	Member	Description
0	UTSNAME_SYSNAME_LEN	Specifies the SYSNAME string length (in characters)
1–12	UTSNAME_SYSNAME	Specifies the SYSNAME string (includes terminating null character) This string should contain the name of the operating system (typically, “MCP/AS”).
13	UTSNAME_NODENAME_LEN	Specifies the NODENAME length (in characters)
14–25	UTSNAME_NODENAME	Specifies the NODENAME (includes terminating null character) This string should contain the name of the node within a BNA network. Typically, this is a value equivalent to the HOSTNAME identifier.
26	UTSNAME_RELEASE_LEN	Specifies the RELEASE string length (in characters)
27–38	UTSNAME_RELEASE	Specifies the RELEASE string (includes terminating null character) This string should contain the current release level of the operating system. For example, “42.450.5099.”
39	UTSNAME_VERSION_LEN	Specifies the VERSION string length (in characters)
40–51	UTSNAME_VERSION	Specifies the VERSION string (includes terminating null character)
52	UTSNAME_MACHINE_LEN	Specifies the MACHINE string length (in characters)

Word Offset	Member	Description
53–64	UTSNAME_MACHINE	Specifies the MACHINE string (includes terminating null character) This string should contain the hardware system type. For example, “A11.”

Section 3
POSIX Functions in ALGOL

About this Section

This section describes POSIX functions that are available when the ALGOL program includes the SYMBOL/POSIX/ALGOL/PROPERTIES file.

This document does not provide detailed descriptions of each function. Instead, it usually references an equivalent C language function. See the *C Programming Reference Manual*, *Volume 2: Headers and Functions*, for details about these functions.

Your program does not require library procedure declarations. The include file automatically provides all POSIX interface related library procedure declarations.

Within this section, each function description includes:

- A definition of the required ALGOL syntax
- A brief description of the function

  (In most cases, this description includes a reference to the equivalent C language function.)
- A "Comparison to C Function" table

  This table:

  - Matches ALGOL function parameters with corresponding C language arguments.
  - Indicates the "rule" required to code or use the ALGOL parameters. See Section 2 for detailed information on these rules.

    Provides additional information required to code the ALGOL parameters.
- A description of differences between the ALGOL function and its C language equivalent

Note: *The SYMBOL/POSIX/ALGOL/PROPERTIES file implements many of these functions as defines. Although the define will evaluate each argument one time, the order of evaluation may differ from the order indicated by the function's syntax.*

ACCESS

ALGOL Syntax

```
ACCESS (PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE,
        AMODE, ERRNO);
```

Description

The ACCESS function determines whether the calling process has a specified access permission for a particular file. The AMODE parameter must contain a value that defines a particular file access permission. If a file and a directory share the specified filename, this function determines access permission for the directory.

ACCESS is similar to the following C function:

```
int access (const char *path, int amode);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	*path*	Path definition.
AMODE	*amode*	Call-by-value integer. See “AMODE parameter” in Section 2 for a list of defined values. Note that file existence is always checked.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

An AMODE parameter value of F_OK is not recognized; file existence is always checked.

ALARM

ALGOL Syntax

```
ALARM (SECS, ERRNO);
```

Description

The ALARM function causes the system to send a signal (type SIGALRM) to the calling process after the specified number of seconds have elapsed. If SECS is zero, this function cancels any previously specified ALARM function.

ALARM is equivalent to the following C function:

```
unsigned int alarm(unsigned int seconds);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SECS	*seconds*	Call-by-value real number.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

CHDIR

ALGOL Syntax

```
CHDIR (PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, ERRNO);
```

Description

The CHDIR function causes a specified pathname to become the current working directory.

CHDIR is equivalent to the following C function:

```
int chdir(const char *path);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	*path*	Path definition.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

CHMOD

ALGOL Syntax

```
CHMOD (PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, MODE,
       ERRNO);
```

Description

Note: *The CHMOD function does not currently work with directories. Directory support is planned for a future release.*

The CHMOD function alters the SECURITYMODE file attribute for a specified disk file. The SECURITYMODE attribute is an encoded value that contains the following file-specific information:

- Owner class, Group class, and Other class file access permission flags
- Guard file flags
- Set Usercode flag
- Set Groupcode flag

CHMOD is equivalent to the following C function:

```
int chmod(const char *path, mode_t mode);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	*path*	Path definition.
MODE	*mode*	Call-by-value integer. This parameter defines the encoded SECURITYMODE file attribute value. SECURITYMODE is equivalent to the File Access Mode field ([13:14]) within the File Mode. See Table 2–3.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

CHOWN

ALGOL Syntax

```
CHOWN (PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, OWNER,
       GROUP, ERRNO);
```

Description

Note: *The CHOWN function does not currently work with directories. Directory support is planned for a future release.*

The CHOWN function changes the OWNER and/or GROUP file attribute of a specified file to the ID values defined by the function parameters. The specified user and group IDs map to attributes in the USERDATAFILE.

CHOWN is equivalent to the following C function:

```
int chown (const char *path, uid_t owner, git_t group);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	*path*	Path definition.
OWNER	*owner*	Call-by-value integer.
GROUP	*group*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

EXECVE

ALGOL Syntax

```
EXECVE (PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, ARGV,
        ARGVINDX, ARGVINDX_OFF, ARGVINDX_LEN, ARGVSZ, ARGVSZ_OFF,
        ARGVSZ_LEN, ENVP, ENVPINDX, ENVPINDX_OFF, ENVPINDX_LEN,
        ENVPSZ, ENVPSZ_OFF, ENVPSZ_LEN, ERRNO);
```

Description

The EXECVE function replaces the current process image with a new process image and executes a new code file. The PATH array specifies the pathname of the new executable file, the ARGV array specifies one or more strings of program arguments, and the ENVP array specifies zero or more strings of environment variables.

EXECVE is equivalent to the following C function:

```
int execve (const char *path, char *const argv[ ], char *const
           envp[ ]);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	*path*	Path definition.
ARGV	*argv*	EBCDIC array input. The ARGV_OFF and ARGV_LEN parameters are omitted (they must contain 0).
ARGVINDX, ARGVINDX_OFF, ARGVINDX_LEN	–	Integer array input. This array contains ARGV array indexes. Each index points to the start of an argument.
ARGVSZ, ARGVSZ_OFF, ARGVSZ_LEN	–	Integer array input. This array contains the length of each argument (located in ARGV) pointed to by the indexes in ARGVINDX.
ENVP	*envp*	EBCDIC array input. The ENVP_OFF and ENVP_LEN parameters are omitted (they must contain 0).

ALGOL	C	Rule for ALGOL
ENVPINDX, ENVPINDX_OFF, ENVPINDX_LEN	–	Integer array input. This array contains ENVP array indexes. Each index points to the start of an environment variable.
ENVPSZ, ENVPSZ_OFF, ENVPINDX_LEN	–	Integer array input. This array contains the length of each environment variable (located in ENVP) pointed to by the indexes in ENVPINDX.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result. If this function is successful, it returns no result.

Functional Differences

None

FORK

ALGOL Syntax

```
FORK (ERRNO);
```

Description

The FORK function creates a new process. The new process (known as a *child process*) inherits many attributes from the creating process.

FORK is equivalent to the following C function:

```
pid_t fork (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

The ALGOL and C-based FORK functions are essentially identical.

In an ALGOL program, the child process inherits:

- A copy of all stack data and arrays
- Declared files (in a closed state)
- TASK declarations (untouched)
- DIRECT ARRAYs (with reinitialized direct state)
- LIBRARIES (delinked)
- EVENT declarations (HAPPENED and AVAILABLE states are indeterminate)
- All software interrupt attachments to EVENTs

The operating system changes the child process's stack data structures:

- It modifies program control words (PCWs), stuffed indirect reference words (SIRWs), and data descriptors in active LIBRARY stack frames. The modified words do not have pointers into the library D1 or D2 stacks.
- It modifies indirect reference words (IRWs) that previously pointed into the forking (original) stack. The modified IRWs point into the new stack.
- It copies messages with mom descriptors in the forking stack. The new messages have mom descriptors in the new stack.

GETCWD

ALGOL Syntax

```
GETCWD (STR, OFFSET, MAX, ERRNO);
```

Description

The GETCWD function obtains a string that represents the absolute pathname of the current working directory.

GETCWD is equivalent to the following C function:

```
char *getcwd (char *buf, size_t size);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
STR, OFFSET, MAX	*buf* *size*	EBCDIC array output. Note that MAX is functionally equivalent to *size*.
ERRNO	–	ERRNO rule.
<result>	–	Integer result. Possible result values are: >0 Length of string (excluding NULL). –1 An error occurred.

Functional Differences

The ALGOL function receives a different result.

GETEGID

ALGOL Syntax

```
GETEGID (ERRNO);
```

Description

The GETEGID function returns the effective group ID of the calling process.

GETEGID is equivalent to the following C function:

```
gid_t getegid (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

GETEUID

ALGOL Syntax

```
GETEUID (ERRNO);
```

Description

The GETEUID function returns the effective user ID of the calling process.

GETEUID is equivalent to the following C function:

```
uid_t geteuid (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

GETGID

ALGOL Syntax

```
GETGID (ERRNO);
```

Description

The GETGID function returns the real group ID of the calling process.

GETGID is equivalent to the following C function:

```
gid_t getgid (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

GETGRGID

ALGOL Syntax

```
GETGRGID (GID, MEM, MEM_OFF, MEM_MAX, ERRNO);
```

Description

The GETGRGID function obtains the GROUP structure for a specified group ID. A GROUP structure consists of:

- Group name
- Group ID
- Group member list

GETGRGID is equivalent to the following C function:

```
struct group *getgrgid (gid_t gid);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
GID	*gid*	Call-by-value integer.
MEM, MEM_OFF, MEM_MAX	<result>	Structure array output. See “GROUP structure” in Section 2 for a definition of the returned structure.
ERRNO	–	ERRNO rule.
<result>	–	Integer result. Possible result values are: >0 Operation was successful. The positive result is the number of words returned in the GROUP structure. <–1 Defined MEM array space was insufficient. The absolute value of the negative result is the number of words required to hold the GROUP structure. –1 An error occurred.

Functional Differences

The ALGOL function receives a different result.

GETGRNAM

ALGOL Syntax

```
GETGRNAM (NAME, NAME_OFF, NAME_LEN, MEM, MEM_OFF, MEM_MAX, ERRNO);
```

Description

The GETGRNAM function obtains the GROUP structure for a specified group name. A GROUP structure consists of:

- Group name
- Group ID
- Group member list

GETGRGNAM is equivalent to the following C function:

```
struct group *getgrnam (const char *name);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
NAME, NAME_OFF, NAME_LEN	*name*	EBCDIC array input
MEM, MEM_OFF, MEM_MAX	<result>	Structure array output. See “GROUP structure” in Section 2 for a definition of the returned structure.
ERRNO	–	ERRNO rule.
<result>	–	Integer result. Possible result values are: >0 Operation was successful. The positive result is the number of words returned in the GROUP structure. <–1 Defined MEM array space was insufficient. The absolute value of the negative result is the number of words required to hold the GROUP structure. –1 An error occurred.

Functional Differences

The ALGOL function receives a different result.

GETGROUPS

ALGOL Syntax

```
GETGROUPS (GROUPLIST, GROUPLIST_OFF, GROUPLIST_MAX, ERRNO);
```

Description

The GETGROUPS function obtains all supplementary group IDs for the calling process.

GETGROUPS is equivalent to the following C function:

```
int getgroups (int gidsetsize, get_t grouplist [ ]);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
GROUPLIST, GROUPLIST_OFF, GROUPLIST_MAX	*grouplist* *gidsetsize*	Integer array output. Note that GROUPLIST_MAX is functionally equivalent to *gidsetsize*.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None.

GETLOGIN

ALGOL Syntax

```
GETLOGIN (STR, OFFSET, MAX, ERRNO);
```

Description

The GETLOGIN function obtains the login name associated with the calling process. For this implementation, login name and usercode are synonymous.

GETLOGIN is similar to the following C function:

```
char *getlogin (void);
```

Comparison to C Function

ALGOL	**C**	**Rule for ALGOL**
STR, OFFSET, MAX	<result>	EBCDIC array output.
ERRNO	–	ERRNO rule.
<result>	–	Integer result. Possible result values are: >0 The length (excluding NULL) of the name stored in STR. –1 An error occurred.

Functional Differences

The ALGOL function receives a different result.

GETPGID

ALGOL Syntax

```
GETPGID (PID, ERRNO);
```

Description

The GETPGID function returns the process group ID of the process specified by PID. If the PID parameter is 0, GETPGID returns the process group ID of the calling process.

If the function completes successfully, a process group ID is returned. Otherwise, –1 is returned and the ERRNO value is set.

There is no equivalent C language function.

Parameters

Parameter	Description
PID	A call-by-value input integer.
ERRNO	ERRNO rule.
<result>	Integer result.

GETPGRP

ALGOL Syntax

```
GETPGRP (ERRNO);
```

Description

The GETPGRP function returns the process group ID of the calling process.

GETPGRP is equivalent to the following C function:

```
pid_t getpgrp (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None.

GETPID

ALGOL Syntax

```
GETPID (ERRNO);
```

Description

The GETPID function returns the process ID of the calling process.

GETPID is equivalent to the following C function:

```
pid_t getpid (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None.

GETPPID

ALGOL Syntax

```
GETPPID (ERRNO);
```

Description

The GETPPID function returns the parent process ID of the calling process.

GETPPID is equivalent to the following C function:

```
pid_t getppid (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None.

GETPWNAM

ALGOL Syntax

```
GETPWNAM (NAME, NAME_OFF, NAME_LEN, PASSWD, PASSWD_OFF, PASSWD_MAX,
          ERRNO);
```

Description

The GETPWNAM function obtains the PASSWD structure associated with a specified user name.

GETPWNAM is equivalent to the following C function:

```
struct passwd *getpwnam (const char *name);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
NAME, NAME_OFF, NAME_LEN	*name*	EBCDIC array input.
PASSWD, PASSWD_OFF, PASSWD_MAX	<result>	Structure array output. See “PASSWD structure” in Section 2 for a definition of the returned structure.
ERRNO	–	ERRNO rule.
<result>	–	Integer result. Possible result values are: 0 Operation was successful. –1 An error occurred.

Functional Differences

The ALGOL function receives a different result.

GETPWUID

ALGOL Syntax

```
GETPWUID (UID, PASSWD, PASSWD_OFF, PASSWD_MAX, ERRNO);
```

Description

The GETPWUID function obtains the PASSWD structure associated with a specified user ID.

GETPWUID is equivalent to the following C function:

```
struct passwd *getpwuid (uid_t uid);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
UID	*uid*	Call-by-value integer.
PASSWD, PASSWD_OFF, PASSWD_MAX	<result>	Structure array output. See "PASSWD structure" in Section 2 for a definition of the returned structure.
ERRNO	–	ERRNO rule.
<result>	–	Integer result. Possible result values are: 0 Operation was successful. –1 An error occurred.

Functional Differences

The ALGOL function receives a different result.

GETSID

ALGOL Syntax

```
GETSID (PID, ERRNO);
```

Description

The GETSID function returns the session ID of the process specified by PID. If the PID parameter is 0, GETSID returns the session ID of the calling process.

If the function completes successfully, a session ID is returned. Otherwise, –1 is returned and the ERRNO value is set.

There is no equivalent C language function.

Parameters

Parameter	Description
PID	A call-by-value input integer.
ERRNO	ERRNO rule.
<result>	Integer result.

GETUID

ALGOL Syntax

```
GETUID (ERRNO);
```

Description

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

GETUID is equivalent to the following C function:

```
uid_t getuid (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None.

GETUSERID

ALGOL Syntax

```
GETUSERID (STR, STR_OFF, STR_LEN, ERRNO);
```

Description

The GETUSERID function returns the user ID associated with the user name specified in array STR.

There is no equivalent C language function.

Parameters

Parameter	Description
STR, STR_OFF, STR_LEN	STR is an EBCDIC input array. It must contain a string representing the applicable user name.
ERRNO	ERRNO rule.
<result>	Integer result. Possible result values are: >0 Successful completion of function. Returned integer is the requested user ID. –1 An error occurred.

GETUSERNAME

ALGOL Syntax

```
GETUSERNAME (UID, STR, STR_OFF, STR_MAX, ERRNO);
```

Description

The GETUSERNAME function returns the user name associated with a specified user ID. The user name string is placed in array STR.

There is no equivalent C language function.

Parameters

Parameter	**Description**
UID	A call-by-value input integer that specifies a user ID.
STR, STR_OFF, STR_MAX	STR is an EBCDIC output array used to hold the requested user name.
ERRNO	ERRNO rule.
<result>	Integer result. Possible result values are: >0 Successful completion of function. Returned integer is the length of the output string (excluding NULL) in STR. –1 An error occurred.

KILL

ALGOL Syntax

```
KILL (SIG, PID, ERRNO);
```

Description

The KILL function sends a signal to the process or group of processes defined by the PID parameter. The SIG parameter specifies the signal to be sent.

Note: *The calling process must have sending permission to send a signal to any process.*

KILL is equivalent to the following C function:

```
int kill (pid_t pid, int sig);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*sig*	Call-by-value integer. See “SIG parameter” in Section 2 for a listing of signal names and SIG parameter values.
PID	*pid*	Call-by-value integer. See “PID parameter (KILL)” in Section 2 for additional information.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None.

MKFIFO

ALGOL Syntax

```
MKFIFO (PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, MODE,
        DEV_INFO, ERRNO);
```

Description

The MKFIFO function creates a new FIFO special file with a filename defined by the pathname specified in the PATH array. The MODE parameter defines the permission bits for this file.

MKFIFO is equivalent to the following C function:

```
int mkfifo (const char *path, mode_t mode);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	*path*	Path definition.
MODE	*mode*	Call-by-value integer. See Table 2–3 for information on the file permission bits defined by this parameter.
DEV_INFO	–	Call-by-value integer. This field specifies the type of character format to be transferred. The following defines are valid: EBCDICV ASCIIV
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None.

NICE

ALGOL Syntax

```
NICE (VAL, ERRNO);
```

Description

The NICE function changes the running (current) priority of the calling process. The following formula defines the running priority and nice value relationship:

```
(Running Priority) = (Original Priority) - (Nice Value)
```

To change the running priority, the NICE function generates a new nice value. The function's VAL parameter (a positive or negative integer) is added to the existing nice value to form the new value.

Nice values can range between 0 (the original nice value of a process) and 255. An attempt to modify the nice value outside of these limits has the following effect:

- If the nice value is made negative, a –1 result is returned and the EPERM value is set in ERRNO.
- If the nice value is made higher than 255, results are unspecified.

A process's running priority can range between 0 and its original priority value.

There is no equivalent C language function.

Parameters

Parameter	Description
VAL	Call-by-value integer. This input parameter contains a positive or negative value that is added to the process's current nice value.
ERRNO	ERRNO rule.
<result>	Integer result. If the operation is successful, the value returned is the newly calculated nice value.

PATHCONF

ALGOL Syntax

```
PATHCONF (PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, NAME,
          ERRNO);
```

Description

The PATHCONF function returns information about a configurable variable for an open file. This file is specified by the path definition. The NAME parameter specifies a configurable variable.

PATHCONF is equivalent to the following C function:

```
long pathconf (const char path, int name);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	*path*	Path definition.
NAME	*name*	By-value integer. See “NAME parameter (for PATHCONF function)” in Section 2 for a summary of configurable variables and associated NAME parameter integers.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

PAUSE

ALGOL Syntax

```
PAUSE (ERRNO);
```

Description

The PAUSE function suspends the calling process. The process remains suspended until receipt of a signal that does one of the following:

- Executes a signal-catching function
- Terminates the process

PAUSE is equivalent to the following C function:

```
int pause (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

RAISE

ALGOL Syntax

```
RAISE (SIG, ERRNO);
```

Description

The RAISE function sends a signal to the calling process. The SIG parameter specifies the signal type.

RAISE is equivalent to the following C function:

```
int raise (int sig);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*sig*	Call-by-value integer. See "SIG parameter" in Section 2 for a listing of signal types and SIG parameter values.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_CLOSE

ALGOL Syntax

```
SEM_CLOSE (SEM, ERRNO);
```

Description

The SEM_CLOSE function closes a named semaphore that is currently open to the calling process. A closed semaphore is no longer available to the this process. However, it is not removed from the system.

SEM_CLOSE is equivalent to the following C function:

```
int sem_close (sem_t *sem);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEM	*sem*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEMCTL

ALGOL Syntax

```
SEMCTL (SEMID, SEMNUM, CMD, VAL, ARG, ARG_OFF, ARG_LEN, ERRNO);
```

Description

The SEMCTL function provides a set of control operations for an X/Open defined semaphore or semaphore set. The CMD parameter specifies the operation (command) to be performed.

SEMCTL is equivalent to the following C function:

```
int semctl (int semid, int semnum, int cmd, . . .);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEMID	*semid*	Call-by-value integer.
SEMNUM	*semnum*	Call-by-value integer.
CMD	*cmd*	Call-by-value integer. See “CMD parameter” in Section 2 for a description of commands.
VAL	Fourth argument	Call-by-value integer (for SEM_SETVAL command only).
ARG, ARG_OFF, ARG_LEN	Fourth argument	Real array output for SEM_GETALL command. Real array input for SEM_SETALL command. Structure array input for IPC_SET command. Structure array output for IPC_STAT command. See “SEMID_DS structure” Section 2 for a description of the structure contained here for the IPC_SET and IPC_STAT commands.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_DESTROY

ALGOL Syntax

```
SEM_DESTROY (SEM, ERRNO);
```

Description

The SEM_DESTROY function removes an unnamed semaphore from the system. The SEM parameter specifies this semaphore.

SEM_DESTROY is equivalent to the following C function:

```
int sem_destroy (sem_t *sem);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEM	*sem*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEMGET

ALGOL Syntax

```
SEMGET (KEY, NSEMS, SEMFLG, ERRNO);
```

Description

The SEMGET function performs either of the following operations relative to X/Open defined semaphores:

- It returns the semaphore ID associated with the KEY parameter.
- It creates a new set of semaphores and initializes its SEMID_DS structure.

The KEY and SEMFLG input parameters define the operation that is performed.

SEMGET is equivalent to the following C function:

```
int semget (key_t key, int nsems, int semflg);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
KEY	*key*	Call-by-value integer. The following defined value is valid: IPC_PRIVATE
NSEMS	*nsems*	Call-by-value integer.
SEMFLG	*semflg*	Call-by-value integer. See “SEMFLG parameter” in Section 2 for a description of allowable values.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_GETVALUE

ALGOL Syntax

```
SEM_GETVALUE (SEM, SVAL, ERRNO);
```

Description

The SEM_GETVALUE function retrieves the value of the named or unnamed semaphore indicated by the SEM parameter. The state or value of the semaphore is not affected.

SEM_GETVALUE is equivalent to the following C function:

```
int sem_getvalue (sem_t *sem, int *sval);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEM	*sem*	Call-by-value integer.
SVAL	*sval*	Call-by-reference integer (output).
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_INIT

ALGOL Syntax

```
SEM_INIT (SEM, PSHARED, VAL, ERRNO);
```

Description

The SEM_INIT function initializes an unnamed semaphore. An initialized semaphore is available for use.

SEM_INIT is equivalent to the following C function:

```
int sem_init (sem_t *sem, int pshared, unsigned int value);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEM	*sem*	Call-by-reference integer (output).
PSHARED	*pshared*	Call-by-value integer. If non-zero, multiple processes will share the semaphore.
VAL	*value*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEMOP

ALGOL Syntax

```
SEMOP (SEMID, SOPS, SOPS_OFF, SOPS_MAX, NSOPS, ERRNO);
```

Description

The SEMOP function performs user-specified operations on a specified group of X/Open defined semaphores. Information in the SOPS array defines this operation.

SEMOP is equivalent to the following C function:

```
int semop (int semid, struct sembuf *sops, unsigned nsops);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEMID	*semid*	Call-by-value integer.
SOPS, SOPS_OFF, SOPS_MAX	*sops*	Structure array input. See “SEMBUF structure” in Section 2 for a definition of the structure contained in this array.
NSOPS	*nsops*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_OPEN

ALGOL Syntax

```
SEM_OPEN (NAME, NAME_OFF, NAME_LEN, OFLAG, MODE, VAL, ERRNO);
```

Description

The SEM_OPEN function opens the named semaphore referred to in the NAME array. The function can do either of the following:

- Access an existing semaphore.
- Create a new semaphore.

The process can use an opened semaphore until it closes the semaphore with the SEM_CLOSE function.

SEM_OPEN is equivalent to the following C function:

```
sem_t *sem_open (const char *name, int oflag, . . .);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
NAME, NAME_OFF, NAME_LEN	*name*	EBCDIC array input.
OFLAG	*oflag*	Call-by-value integer. The following defines are valid: O_CREAT O_EXCL
MODE	Third argument if O_CREAT is set	Call-by-value integer. This input parameter contains permission bits for the new semaphore. See “MODE parameter” in Section 2 for a description of bit assignments.
VAL	Fourth argument if O_CREAT is set	Call-by-value integer. This input parameter contains the value of the new semaphore. A negative value cannot be specified.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_POST

ALGOL Syntax

```
SEM_POST (SEM, ERRNO);
```

Description

The SEM_POST function unlocks the named or unnamed semaphore indicated by the SEM parameter. An unlock operation increments the value of a semaphore by 1. A semaphore is unlocked when it has a value greater than 0.

If a process is waiting to lock the semaphore, it can proceed once the semaphore's value is greater than 0.

SEM_POST is equivalent to the following C function:

```
int sem_post (sem_t *sem);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEM	*sem*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_TRYWAIT

ALGOL Syntax

```
SEM_TRYWAIT (SEM, ERRNO);
```

Description

The SEM_TRYWAIT function attempts to decrement the value of the named or unnamed semaphore referred to by the SEM parameter. This operation occurs only if the semaphore's value is greater than 0. A semaphore is locked when its value is 0.

SEM_TRYWAIT is similar to the SEM_WAIT function; it attempts to decrement the value of a semaphore by 1. However, the SEM_TRYWAIT function does not wait if it cannot decrement the value immediately. Instead, it returns an EAGAIN error.

SEM_TRYWAIT is equivalent to the following C function:

```
int sem_trywait (sem_t *sem);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEM	*sem*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_UNLINK

ALGOL Syntax

```
SEM_UNLINK (NAME, NAME_OFF, NAME_LEN, ERRNO);
```

Description

The SEM_UNLINK function deletes the name of a semaphore from the system table. Once unlinked, the named semaphore cannot be accessed by subsequent open operations.

SEM_UNLINK is equivalent to the following C function:

```
int sem_unlink (const char *name);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
NAME, NAME_OFF, NAME_LEN	*name*	EBCDIC array input.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SEM_WAIT

ALGOL Syntax

```
SEM_WAIT (SEM, ERRNO);
```

Description

The SEM_WAIT function operates on a semaphore referenced by the SEM parameter as indicated in the following chart. A semaphore is locked when its value is 0.

If the named or unnamed semaphore value is . . .	Then SEM_WAIT . . .
greater than 0	decrements the semaphore value by 1 and the function returns immediately.
0 (zero)	waits until the semaphore value becomes greater than 0 (that is, until another process unlocks it). SEM_WAIT then decrements the semaphore value by 1 and returns. SEM_WAIT waits until it can decrement the semaphore value or until one of the following events occurs: • A signal interrupts this operation. In this case, the state of the semaphore is unchanged by the calling process. • The semaphore is destroyed.

SEM_WAIT is similar to the SEM_TRYWAIT function; however, SEM_TRYWAIT does not wait if the semaphore is currently locked.

SEM_WAIT is equivalent to the following C function:

```
int sem_wait (sem_t *sem);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SEM	*sem*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SETGID

ALGOL Syntax

```
SETGID (GID, ERRNO);
```

Description

The SETGID function changes the effective group ID of the calling process.

SETGID is equivalent to the following C function:

```
int setgid (gid_t gid);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
GID	*gid*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SETPGID

ALGOL Syntax

```
SETPGID (PID, PGID, ERRNO);
```

Description

The SETPGID function changes the process group ID of a specified process (defined by PID) to a specified value (defined by PGID). This function can be used to do either of the following:

- Create a new process group within the session of the calling process.
- Move a specified process to another process group.

SETPGID is equivalent to the following C function:

```
int setpgid (pid_t pid, pid_t pgid);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PID	*pid*	Call-by-value integer.
PGID	*pgid*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SETPGRP

ALGOL Syntax

```
SETPGRP (ERRNO);
```

Description

If the calling process is not already a session leader, the SETPGRP function establishes it as a session leader and process group leader. SETPGRP establishes the following environment:

- A new session exists. The calling process is session leader. There are no other processes in the session.
- A new process group exists. The calling process is its process group leader. There are no other processes in the process group.
- The new process group ID is the process ID of the calling process.
- There is no controlling terminal.

There is no equivalent C language function.

Parameters

Parameter	Description
ERRNO	ERRNO rule.
<result>	Integer result. Possible result values are: >0 Successful completion of function. The returned integer is the new process group ID. –1 An error occurred.

SETSID

ALGOL Syntax

```
SETSID (ERRNO);
```

Description

The SETSID function establishes the calling process as a session leader and process group leader. SETSID is identical to the SETPGRP function.

SETSID is equivalent to the following C function:

```
pid_t setsid (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SETUID

ALGOL Syntax

```
SETUID (UID, ERRNO);
```

Description

The SETUID function changes the effective user ID of the calling process.

SETUID is equivalent to the following C function:

```
int setuid (uid_t uid);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
UID	*uid*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGACTION

ALGOL Syntax

```
SIGACTION (SIG, ACT_PROC, ACT, ACT_OFF, ACT_LEN, OACT, OACT_OFF,
           OACT_MAX, ERRNO);
```

Description

The SIGACTION function allows the calling process to specify and/or examine the action associated with an indicated signal type.

SIGACTION is equivalent to the following C function:

```
int sigaction (int sig, const struct sigaction *act,
    struct sigaction *oact);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*sig*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ACT_PROC	–	Signal handler procedure. ACT_PROC is the procedure invoked when the specified signal occurs.
ACT, ACT_OFF, ACT_LEN	*act*	Structure array input. See “SIGACTION structure” in Section 2 for additional information on this structure.
OACT, OACT_OFF, OACT_MAX	*oact*	Structure array output. See “SIGACTION structure” in Section 2 for additional information on this structure.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGADDSET

ALGOL Syntax

```
SIGADDSET (SET, SIG, ERRNO);
```

Description

The SIGADDSET function adds a signal (specified by the SIG parameter) to a signal set (specified by the SET parameter).

SIGADDSET is equivalent to the following C function:

```
int sigaddset (sigset_t *set, int signo);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SET	*set*	Call-by-reference integer.
SIG	*signo*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGDELSET

ALGOL Syntax

```
SIGDELSET (SET, SIG, ERRNO);
```

Description

The SIGDELSET function removes a signal type (specified by the SIG parameter) from a signal set (specified by the SET parameter).

SIGDELSET is equivalent to the following C function:

```
int sigdelset (sigset_t *set, int signo);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SET	*set*	Call-by-reference integer.
SIG	*signo*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGEMPTYSET

ALGOL Syntax

```
SIGEMPTYSET (SET, ERRNO);
```

Description

The SIGEMPTYSET function initializes the signal set specified by the SET parameter. The initialized set excludes all signal types.

SIGEMPTYSET is equivalent to the following C function:

```
int sigemptyset (sigset_t *set);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SET	*set*	Call-by-reference integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGFILLSET

ALGOL Syntax

```
SIGFILLSET (SET, ERRNO);
```

Description

The SIGFILLSET function initializes a signal set (specified by the SET parameter) so that all signal types are included.

SIGFILLSET is equivalent to the following C function:

```
int sigfillset (sigset_t *set);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SET	*set*	Call-by-reference integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGHOLD

ALGOL Syntax

```
SIGHOLD (SIG, ERRNO);
```

Description

The SIGHOLD function adds a signal type (specified by the SIG parameter) to the process's signal mask. The signal mask contains signal types that are blocked for delivery.

SIGHOLD is equivalent to the following C function:

```
int sighold (int signo);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*signo*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGIGNORE

ALGOL Syntax

```
SIGIGNORE (SIG, ERRNO);
```

Description

The SIGIGNORE function establishes the ignore action for the signal type specified by the SIG parameter.

SIGIGNORE is equivalent to the following C function:

```
int sigignore (int signo);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*signo*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGISMEMBER

ALGOL Syntax

```
SIGISMEMBER (SET, SIG, ERRNO);
```

Description

The SIGISMEMBER function tests whether a signal type (specified by the SIG parameter) is a member of the signal set specified by the SET parameter.

SIGISMEMBER is equivalent to the following C function:

```
int sigismember (const sigset_t *set, int signo);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SET	*set*	Call-by-value integer.
SIG	*signo*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGNAL

ALGOL Syntax

```
SIGNAL (SIG, ACT_PROC, DISP, ERRNO);
```

Description

The SIGNAL function defines the action to be taken upon receipt of the signal type specified by the SIG parameter. The DISP parameter references a signal handling procedure address or specifies a particular function.

SIGNAL is equivalent to the following C function:

```
void (*signal (int sig, void (*func)(int sig))) (int sig);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*sig*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ACT_PROC	–	Signal handler procedure. ACT_PROC is the procedure invoked when the specified signal occurs.
DISP	*func*	Call-by-value integer. See “DISP parameter” in Section 2 for information on allowed values.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGPAUSE

ALGOL Syntax

```
SIGPAUSE (SIG, ERRNO);
```

Description

The SIGPAUSE function removes a signal type (specified by the SIG parameter) from the process's signal mask and then suspends the process until that signal occurs.

SIGPAUSE is equivalent to the following C function:

```
int sigpause (int sig);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*signo*	Call-by-value integer. See "SIG parameter" in Section 2 for an enumeration of signal types.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGPENDING

ALGOL Syntax

```
SIGPENDING (SET, ERRNO);
```

Description

The SIGPENDING function retrieves a set of signal types that are pending for the calling process. Upon completion, output parameter SET references this signal set.

SIGPENDING is equivalent to the following C function:

```
int sigpending (sigset_t *set);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SET	*set*	Call-by-reference integer (output).
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGPROCMASK

ALGOL Syntax

```
SIGPROCMASK (HOW, SET, OSET, ERRNO);
```

Description

The SIGPROCMASK function allows the calling process to examine or change its set of blocked signals (signal mask). This function can do either of the following:

- Modify the process's signal mask.
- Retrieve the current signal mask.

SIGPROCMASK is equivalent to the following C function:

```
int sigprocmask (int how, const sigset_t *set, sigset_t *oset);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
HOW	*how*	Call-by-value integer. See "HOW parameter" in Section 2 for a description of allowable values.
SET	*set*	Call-by-value integer.
OSET	*oset*	Call-by-reference integer (output). This parameter returns the original (unaltered) signal mask.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

The ALGOL and C functions require different syntax to retrieve the current signal mask:

- With SIGPROCMASK, you specify SIG_ENQUIRE (–1) in the HOW parameter.
- With sigprocmask( ), you specify a NULL pointer in the *set* argument.

SIGPUSH

ALGOL Syntax

```
SIGPUSH (ERRNO);
```

Description

The SIGPUSH function creates a new signal environment for the calling process.

SIGPUSH is equivalent to the following platform-specific C macro:

```
int sigpush (void);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGRELSE

ALGOL Syntax

```
SIGRELSE (SIG, ERRNO);
```

Description

The SIGRELSE function removes a signal type (specified by the SIG parameter) from the calling process's signal mask. The process can then receive that signal type.

SIGRELSE is equivalent to the following C function:

```
int sigrelse (int sig);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*sig*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGSET

ALGOL Syntax

```
SIGSET (SIG, ACT_PROC, DISP, ERRNO);
```

Description

The SIGSET function blocks a signal type or specifies an action for a signal type.

SIGSET is equivalent to the following C function:

```
void (*sigset (int sig, void (*func) (int sig))) (int sig);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIG	*sig*	Call-by-value integer. See “SIG parameter” in Section 2 for an enumeration of signal types.
ACT_PROC	–	Signal handler procedure. ACT_PROC is the procedure invoked when the specified signal occurs.
DISP	*func*	Call-by-value integer. See “DISP parameter” in Section 2 for information on allowed values.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SIGSUSPEND

ALGOL Syntax

```
SIGSUSPEND (SIGMASK, ERRNO);
```

Description

The SIGSUSPEND function replaces the signal mask of the calling process and then suspends the process. The process will then wait for a signal to awaken it.

SIGSUSPEND is identical to the following C function:

```
int sigsuspend (const sigset_t *sigmask);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SIGMASK	*sigmask*	Call-by-value integer.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

SLEEP

ALGOL Syntax

```
SLEEP (SECS, ERRNO);
```

Description

The SLEEP function suspends the calling process for the number of real-time seconds specified by the SECS parameter. A signal may also reactivate the suspended process.

SLEEP is equivalent to the following C function:

```
unsigned int sleep (unsigned int seconds);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
SECS	*seconds*	Call-by-value real.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

STAT

ALGOL Syntax

```
STAT (PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, BUF,
      BUF_OFF, BUF_MAX, ERRNO);
```

Description

The STAT function obtains file status for the file specified in the PATH array. This status is returned to the BUF array.

STAT is equivalent to the following C function:

```
int stat (const char *path, struct stat *buf);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	*path*	Path definition.
BUF, BUF_OFF, BUF_MAX	*buf*	Structure array output. See “STAT structure” in Section 2 for a definition of the returned structure.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

STRERROR

ALGOL Syntax

```
STRERROR (BUFF, BUFF_OFF, BUFF_MAX, ERRNO);
```

Description

The STRERROR function obtains a string of descriptive text associated with the error value specified in the ERRNO parameter. This string is placed in the BUFF array.

STRERROR is equivalent to the following C function:

```
char *strerror (int errnum);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
BUFF, BUFF_OFF, BUFF_MAX	<result>	EBCDIC array output.
ERRNO	*errnum*	Call-by-value integer. ERRNO is a required input parameter.
<result>	–	Integer result. Possible values are: >=0 Operation was successful. The returned value is the length of the text string. –1 An error occurred.

Functional Differences

The ALGOL function receives a different result.

SYSCONF

ALGOL Syntax

```
SYSCONF (NAME, ERRNO);
```

Description

The SYSCONF function obtains the current value of the configurable system variable defined by NAME.

SYSCONF is equivalent to the following C function:

```
long sysconf (int name);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
NAME	*name*	Call-by-value integer. See “NAME parameter (for SYSCONF function)” in Section 2 for an enumeration of configurable system variables.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

TIMEP

ALGOL Syntax

```
TIMEP (ERRNO);
```

Description

The TIMEP function returns a value that represents the current time.

The returned value is the number of seconds that have elapsed since 00:00:00 Greenwich Mean Time on January 1, 1970.

TIMEP is equivalent to the following C function:

```
time_t time (time_t *tloc);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
ERRNO	–	ERRNO rule.
<result>	<result>	Real result.

Functional Differences

The ALGOL function does not support *tloc*. Only the returned value provides time information.

TIMES

ALGOL Syntax

```
TIMES (BUF, BUF_OFF, BUF_MAX, BUF_LEN, ERRNO);
```

Description

The TIMES function obtains time accounting information (expressed in clock ticks) for the current process and its child processes.

TIMES is equivalent to the following C function:

```
clock_t times (struct tms *buffer);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
BUF, BUF_OFF, BUFF_MAX,	*buffer*	Real array output. See “TMS structure” in Section 2 for a definition of the data returned to this array.
BUF_LEN	–	Call-by-value integer (output).
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

BUF_LEN indicates the number of words transferred into the BUF array. The C language times( ) function does not return this value.

UNAME

ALGOL Syntax

```
UNAME (NAME, NAME_OFF, NAME_MAX, ERRNO);
```

Description

The UNAME function obtains information about the system's current hardware and software environment.

UNAME is equivalent to the following C function:

```
int uname (struct utsname *name);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
NAME, NAME_OFF, NAME_MAX	*name*	Real array output. See "UTSNAME structure" in Section 2 for a definition of information returned to the NAME array.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

WAITP

ALGOL Syntax

```
WAITP (STATUS, ERRNO);
```

Description

The WAITP function suspends execution of the calling process until one of the following occurs:

- Status information becomes available for any terminated child process.
- A received signal causes execution of a signal-catching function or terminates the process.

The value returned is the process ID of the terminated child process.

WAITP is equivalent to the following C function:

```
pid_t wait (int *stat_loc);
```

Comparison to C Function

ALGOL	C	Rule for ALGOL
STATUS	*stat_loc*	Call-by-reference integer. See “STATUS parameter” in Section 2 for an analysis of the returned termination status.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

WAITPID

ALGOL Syntax

```
WAITPID (PID, STATUS, OPTIONS, ERRNO);
```

Description

The WAITPID function suspends execution of the calling process until a child process (defined by the PID parameter) returns status. The OPTIONS parameter allows you to define certain aspects of this function's operation.

WAITPID is equivalent to the following C function:

```
pid_t waitpid (pid_t pid, int *stat_loc, int options);
```

The OPTIONS parameter allows you to modify the operation of this function:

- If you specify the WNOHANG option, the function does not wait if termination status is not immediately available. Instead, a value of –1 is returned.
- If you specify the WUNTRACED option, the function reports status of stopped or terminated child processes. By default, only terminated task status is reported.

Comparison to C Function

ALGOL	C	Rule for ALGOL
PID	*pid*	Call-by-value integer. See "PID parameter" in Section 2 for information on allowed values.
STATUS	*stat_loc*	Call-by-reference integer. See "STATUS parameter" in Section 2 for an analysis of the returned termination status.
OPTIONS	*options*	Call-by-value integer. See "OPTIONS parameter (for WAITPID)" in Section 2 for information on allowed values.
ERRNO	–	ERRNO rule.
<result>	<result>	Integer result.

Functional Differences

None

Section 4
Unsupported POSIX Functions

Table 4-1 lists those POSIX interface based C language functions not currently supported by the ALGOL include file or through a library procedure. Some of these functions will be supported in a future release.

There are two primary reasons for an unsupported C language function:

1. The function is implemented mainly in the C compiler (not in the MCP).
2. The function has side effects that could cause unexpected or incorrect results. In particular, many of the I/O-related functions could cause program data structure corruption if invoked in an ALGOL library called by a C program.

Note: *The ALGOL include file code implements the sigaddset( ), sigdelset( ), sigemptyset( ), and sigfillset( ) functions. MCPSUPPORT library procedures are not invoked.*

Table 4–1. Unsupported POSIX Interface Based C Language Functions

	Supported . . .	
C Language Function	**By the ALGOL Include File?**	**Through a Library Procedure?**
creat( )	No	Yes
cuserid( )	No	Yes
dup( )	No	Yes
dup2( )	No	Yes
execl( )	No	No
execle( )	No	No
execlp( )	No	No
execv( )	No	No
execvp( )	No	No
exit( )	No	Yes

Table 4–1. Unsupported POSIX Interface Based C Language Functions

C Language Function	Supported . . . By the ALGOL Include File?	Supported . . . Through a Library Procedure?
_exit( )	No	Yes
fcntl( )	No	Yes
fileno( )	No	No
fpathconf( )	No	Yes
fstat( )	No	Yes
getenv( )	No	No
lseek( )	No	Yes
_MCPfstat( )	No	Yes
_MCPstat( )	No	Yes
open( )	No	Yes
pipe( )	No	Yes
putenv( )	No	No
read( )	No	Yes
rename( )	No	Yes
setegid( )	No	No
sigaddset( )	Yes	No
sigfillset( )	Yes	No
siglongjmp( )	No	No
tzset( )	No	No
umask( )	No	Yes
write( )	No	Yes

Section 5
POSIX-Related Library Procedures

About this Section

This section describes library procedures that provide POSIX interface related functions in non-C language programs (typically, ALGOL or NEWP). In all cases, the library procedures are exported by the MCPSUPPORT library.

Note: *The library procedures described in this section are internal interfaces used by the system software. These interfaces might also be of use to sophisticated application programs. From one release to another, an internal interface might change in such a way that programs that use the internal interface will be required to make changes to operate correctly. Because internal interfaces are special system interfaces, they do not adhere to the compatibility policies described in the* SSR 42.3 Software Release Capabilities Overview. *You should examine all programs that use internal interfaces before installing a new release to ensure that the internal interface has not changed.*

References to POSIX Functions

The MCPSUPPORT library supports most functions that provide POSIX features for non C language programs. Within this section, these functions are referenced in two ways:

- Most functions are referred to by equivalent C language function names (for example, pipe( ) and stat( )). See the *C Programming Reference Manual*, *Volume 2: Headers and Functions* for details about C language functions.
- A few functions (not available in the C language) are listed in uppercase letters (for example, GETPGID and GETUSERID). See Section 3 for details about these functions.

Table 5–1 lists currently supported POSIX functions and the library procedure associated with each function.

Caution

This section contains references to functions that are “not yet supported.” Do **not** attempt to use these functions. Unsupported functions return an unpredictable result; possible results include an ENOSYS error or a logical program fault.

Use of Library Procedure Information

Systems programmers should use this information as follows:

Language	Function Required	Coding Required
ALGOL	Any function defined in Section 3.	Do NOT explicitly specify a library procedure. Include the SYMBOL/POSIX/ALGOL/PROPERTIES file in the program to declare all POSIX-related library procedures. Use POSIX functions as described in Section 3.
ALGOL	Any of the following: • close( ) • creat( ) • dup( ) • dup2( ) • exit( ) • _exit( ) • fcntl( ) • fpathconf( ) • fstat( ) • lseek( ) • open( ) • pipe( ) • read( ) • write( )	Include the SYMBOL/POSIX/ALGOL/PROPERTIES file in the program to declare all POSIX-related library procedures. Within the program, use the library procedure calls that provide the required functions.
Any language except C or ALGOL (primarily NEWP)	Any function.	Use library procedures defined in this section. Formally declare required library procedures. Do NOT use the defined names listed in Section 2 for INTEGER parameters.

Table 5–1. POSIX Functions and Related Library Procedures

Function	Related Library Procedure	Comment
access( )	POSIX_ACCESS	
alarm( )	POSIX_NANOALARM	
chdir( )	POSIX_CHANGEDIR	
chmod( )	POSIX_CHANGEMODE	
chown( )	POSIX_CHANGEOWNER	

Table 5–1. POSIX Functions and Related Library Procedures

Function	Related Library Procedure	Comment
close( )	POSIX_CLOSE	
creat( )	POSIX_FILE_TO_FD, POSIX_FILEATTRIBAGENT, and POSIX_OPEN	POSIX_FILEATTRIBAGENT support is planned for a future release.
cuserid( )	POSIX_STRINGIDS	
dup( )	POSIX_FCNTL	
dup2( )	POSIX_FCNTL	
execve( )	POSIX_EXECVE	
exit( )	POSIX_EXIT	
_exit( )	POSIX_EXIT	
fchmod( )	POSIX_CHANGEMODE	
fchown( )	POSIX_CHANGEOWNER	
fcntl( )	POSIX_FCNTL	
fork( )	POSIX_FORK	
fpathconf( )	POSIX_PATHCONF	
fstat( )	POSIX_FILESTATUS	
getcwd( )	POSIX_STRINGIDS	
getegid( )	POSIX_INTEGERIDS	
geteuid( )	POSIX_INTEGERIDS	
getgid( )	POSIX_INTEGERIDS	
getgrgid( )	POSIX_GETGRINFO	
getgrnam( )	POSIX_GETGRINFO	
getgroups( )	POSIX_GROUPLIST	
getlogin( )	POSIX_STRINGIDS	
GETPGID	POSIX_INTEGERIDS	
getpgrp( )	POSIX_INTEGERIDS	
getpid( )	POSIX_INTEGERIDS	
getppid( )	POSIX_INTEGERIDS	
getpwnam( )	POSIX_GETPWINFO	
getpwuid( )	POSIX_GETPWINFO	
getsgid( )	POSIX_INTEGERIDS	

Table 5–1. POSIX Functions and Related Library Procedures

Function	Related Library Procedure	Comment
GETSID	POSIX_INTEGERIDS	
getsuid( )	POSIX_INTEGERIDS	
GETUSERID	POSIX_STRINGIDS	
GETUSERNAME	POSIX_STRINGIDS	
kill( )	POSIX_SIGHANDLER	
lchmod( )	POSIX_CHANGEMODE	
lchown( )	POSIX_CHANGEOWNER	
lseek( )	POSIX_SEEK	
lstat( )	POSIX_FILESTATUS	
_MCPfstat( )	POSIX_FILESTATUS	
_MCPlstat( )	POSIX_FILESTATUS	
_MCPstat( )	POSIX_FILESTATUS	
mkfifo( )	MCPX_MKNOD	
NICE	POSIX_SETIDS	
open( )	POSIX_FILE_TO_FD, POSIX_FILEATTRIBAGENT, and POSIX_OPEN	POSIX_FILEATTRIBAGENT support is planned for a future release.
pathconf( )	POSIX_PATHCONF	
pause( )	POSIX_SIGHANDLER	
pipe( )	POSIX_PIPE	
raise( )	POSIX_SIGHANDLER	
read( )	POSIX_SREAD_E or POSIX_SREAD_R	
readlink( )	POSIX_FILESTATUS	
semctl( )	MCPX_SEMCTL	
semget( )	MCPX_SEMGET	
semop( )	MCPX_SEMOP	
sem_close( )	POSIX_SEM_CLOSE	
sem_destroy( )	POSIX_DESTROY	
sem_getvalue( )	POSIX_GETVALUE	
sem_init( )	POSIX_SEM_INIT	
sem_open( )	POSIX_SEM_OPEN	

Table 5–1. POSIX Functions and Related Library Procedures

Function	Related Library Procedure	Comment
sem_post( )	POSIX_SEM_POST	
sem_trywait( )	POSIX_SEM_TRYWAIT	
sem_unlink( )	POSIX_SEM_UNLINK	
sem_wait( )	POSIX_SEM_WAIT	
setegid( )	POSIX_SETIDS	
seteuid( )	POSIX_SETIDS	
setgid( )	POSIX_SETIDS	
setpgid( )	POSIX_SETIDS	
SETPGRP	POSIX_SETIDS	
setsid( )	POSIX_SETIDS	
setuid( )	POSIX_SETIDS	
sigaction( )	POSIX_SIGHANDLER	
sigaddset( )	–	The ALGOL include file contains required code. No library procedure is available.
sigdelset( )	–	The ALGOL include file contains required code. No library procedure is available.
sigemptyset( )	–	The ALGOL include file contains required code. No library procedure is available.
sigfillset( )	–	The ALGOL include file contains required code. No library procedure is available.
sighold( )	POSIX_SIGHANDLER	
sigignore( )	POSIX_SIGHANDLER	
sigismember( )	–	The ALGOL include file contains required code. No library procedure is available.
signal( )	POSIX_SIGHANDLER	
sigpause( )	POSIX_SIGHANDLER	
sigpending( )	POSIX_SIGHANDLER	
sigprocmask( )	POSIX_SIGHANDLER	
sigpush( )	POSIX_SIGHANDLER	

Table 5–1. POSIX Functions and Related Library Procedures

Function	Related Library Procedure	Comment
sigrelse( )	POSIX_SIGHANDLER	
sigset( )	POSIX_SIGHANDLER	
sigsuspend( )	POSIX_SIGHANDLER	
sleep( )	POSIX_NANOSLEEP	
stat( )	POSIX_FILESTATUS	
sysconf( )	POSIX_SYSCONF	
umask( )	POSIX_SETIDS	
uname( )	POSIX_UNAME	
wait( )	POSIX_WAITPID	
waitpid( )	POSIX_WAITPID	
write( )	POSIX_SWRITE_E or POSIX_SWRITE_R	

Format of Library Procedure Descriptions

Library procedures are described in alphabetical order. Each procedure description includes:

- A brief description of the procedure's purpose.
- A listing of POSIX functions supported by the procedure. In most cases, the equivalent C language function is referenced.
- Formal procedure declaration syntax.
- A summary of coding requirements for individual parameters. This summary includes:
  - A reference to the "rule" to be followed when using the parameter or parameter group. Section 2 defines these rules.
  - A brief description of the parameter's purpose.

    There is no detailed ERRNO parameter or result value description. See Section 2 for additional information about these items.

MCPX_MKNOD

The MCPX_MKNOD procedure creates various types of POSIX special files.

Supported Functions

The File Type value (provided in the MODE parameter) defines the C language function invoked by the procedure:

C Function	File Type Specified in MODE
mkfifo( )	S_IFIFO
mkdir( )	S_IFDIR – (not yet supported)
symlink( )	S_IFLNK – (not yet supported)

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE MCPX_MKNOD (PATH, PATH_OFF, PATH_LEN, PATH_TYPE,
                              PATH_SEARCHRULE, MODE, DEV, DEV_OFF,
                              DEV_LEN, DEV_INFO, DEV_SEARCHRULE, ERRNO);
  VALUE        PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, MODE,
               DEV_OFF, DEV_LEN, DEV_INFO, DEV_SEARCHRULE;
  REFERENCE    PATH, DEV, ERRNO;
  EBCDIC ARRAY PATH, DEV [0]
  INTEGER      PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, MODE,
               DEV_OFF, DEV_LEN, DEV_INFO, DEV_SEARCHRULE, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	Path definition	The array contains a string that identifies the special file.
MODE	Call-by-value integer	This input parameter contains the full-word POSIX File Mode object for the file (see Table 2-3). For directory creation requests (not yet supported), additional bits are defined: • A 1 in the sign bit causes the directory to use default security mode values (read, write, search permission for owner; search permission for group and others). • A 1 in [47:01] specifies a POSIX directory.
DEV, DEV_OFF, DEV_LEN	EBCDIC array input	Required only for FIFO creation requests (S_IFIFO specified for file type in the MODE parameter). For these requests, this array contains a string that defines device-dependent attributes in the disk file header.
DEV_INFO	Call-by-value integer	Required only for FIFO creation requests (S_IFIFO specified for file type in the MODE parameter). For these requests, this integer word contains a single field: [11:08] = CHAR_MODEF CHAR_MODEF can contain the following values: 4 = VALUE(EBCDIC) 5 = VALUE(ASCII)
DEV_SEARCHRULE	Call-by-value integer	Not used with currently supported functions.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

MCPX_SEMCTL

The MCPX_SEMCTL procedure provides a set of functions for an X/Open-defined semaphore or semaphore set. Another procedure (MCPX_SEMOP) provides functions for a flexible subset of semaphores.

Supported Functions

MCPX_SEMCTL provides a function equivalent to the C language semctl( ) function.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE MCPX_SEMCTL (SEMID, SEMNUM, CMD, VAL, ARG, ARG_OFF,
                               ARG_LEN, ERRNO);
  VALUE        SEMID, SEMNUM, CMD, VAL, ARG_OFF, ARG_LEN;
  REFERENCE    ARG, ERRNO;
  REAL ARRAY   ARG [0]
  INTEGER      SEMID, SEMNUM, CMD, VAL, ARG_OFF, ARG_LEN, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEMID	Call-by-value integer	This input parameter indicates the semaphore set.
SEMNUM	Call-by-value integer	This input parameter indicates a semaphore member number within the set.
CMD	Call-by-value integer	This input parameter defines the function. See “CMD parameter (SEMCTL)” in Section 2 for additional information.
VAL	Call-by-value integer	Only valid for the SETVAL command For SETVAL, this input parameter contains the new SEMVAL value.
ARG, ARG_OFF, ARG_LEN	Structure array input (for IPC_SET) Structure array output (for IPC_STAT) Real array input (for SEM_SETALL) Real array output (for SEM_GETALL)	See “SEMID_DS structure” Section 2 for a description of the structure contained here for the IPC_SET and IPC_STAT commands.

Parameter	Rule	Description
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

MCPX_SEMGET

The MCPX_SEMGET procedure creates a new set of X/Open-defined semaphores or connects to an existing set of semaphores.

Supported Functions

MCPX_SEMGET provides a function equivalent to the C language semget( ) function.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE MCPX_SEMGET (KEY, NSEMS, SEMFLG, ERRNO);
  VALUE          KEY, NSEMS, SEMFLG;
  REFERENCE      ERRNO;
  INTEGER        KEY, NSEMS, SEMFLG, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
KEY	Call-by-value integer	This input parameter specifies a key value. The following special value is used: 0 IPC_PRIVATE
NSEMS	Call-by-value integer	This input parameter defines the number of semaphores in a set.
SEMFLG	Call-by-value integer	This input parameter specifies semaphore flags and access permission bits. See “SEMFLG Parameter” in Section 2 for a description of allowable values.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

MCPX_SEMOP

The MCPX_SEMOP procedure performs user-defined operations on a specified group of X/Open-defined semaphores.

Supported Functions

MCPX_SEMOP provides a function equivalent to the C language semop( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE MCPX_SEMOP (SEMID, SOPS, SOPS_OFF, SOPS_MAX, NSOPS,
                              ERRNO);
  VALUE          SEMID, SOPS_OFF, SOP_MAX, NSOPS;
  REFERENCE      SOPS, ERRNO;
  REAL ARRAY     SOPS [0];
  INTEGER        SEMID, SOPS_OFF, SOP_MAX, NSOPS, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEMID	Call-by-value integer	This input parameter contains a semaphore identifier.
SOPS, SOPS_OFF, SOPS_MAX	Structure array input	This array contains a semaphore operation structure. See “SEMBUF structure” in Section 2 for a definition of the structure contained in this array.
NSOPS	Call-by-value integer	This input parameter indicates the number of semaphore operation structures.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_ACCESS

The POSIX_ACCESS procedure determines if specified access permissions are available for a particular file or directory.

Supported Functions

POSIX_ACCESS provides a function equivalent to the C language access( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_ACCESS (SELECTOR, PATH, PATH_OFF, PATH_LEN,
                                PATH_TYPE, PATH_SEARCHRULE, AMODE, ERRNO);
  VALUE          SELECTOR, PATH_OFF, PATH_LEN, PATH_TYPE,
                 PATH_SEARCHRULE, AMODE;
  REFERENCE      PATH, ERRNO:
  EBCDIC ARRAY   PATH[0];
  INTEGER        SELECTOR, PATH_OFF, PATH_LEN, PATH_TYPE,
                 PATH_SEARCHRULE, AMODE, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This parameter is always 1.
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	Path definition	The array contains a string that identifies the file or directory to be checked. Directories are not yet supported.
AMODE	Call-by-value integer	See “AMODE parameter” in Section 2 for a definition of this parameter. Note that this function always checks for file existence.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_ALLOCATE_FD

The POSIX_ALLOCATE_FD procedure performs the initial phase of a POSIX.1 open( ) or creat( ) function. The POSIX_ALLOCATE_FD procedure:

- Initializes the FD_VECTOR array of the calling process (only required for the first file descriptor request).
- Searches the FD_VECTOR array for lowest available file descriptor. It resizes the array if necessary.
- Searches SYSTEM_FILE_VECTOR stack for an available entry. It resizes the stack if necessary.
- Creates a new file information block (FIB) and places the mom descriptor into the SYSTEM_FILE_VECTOR stack.
- Associates the newly allocated file descriptor with the newly allocated file located in the SYSTEM_FILE_VECTOR STACK.
- Returns the lowest available file descriptor value to the calling process.

Supported Functions

POSIX_ALLOCATE_FD is one of three library procedures necessary to perform an operation equivalent to the C language open( ) or creat( ) function. These procedures are:

- POSIX_ALLOCATE_FD
- POSIX_FILEATTRIBAGENT
- POSIX_OPEN

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
  INTEGER PROCEDURE POSIX_ALLOCATE_FD (ERRNO);
  REFERENCE  ERRNO;
  INTEGER    ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_CHANGEDIR

The POSIX_CHANGEDIR procedure changes the current working directory of the calling process. The CURRENTDIRECTORY task attribute stores this value.

Supported Functions

POSIX_CHANGEDIR provides a function equivalent to the C language chdir( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_CHANGEDIR (PATH, PATH_OFF, PATH_LEN, PATH_TYPE,
                                   PATH_SEARCHRULE, ERRNO),
  VALUE          PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE;
  REFERENCE      PATH, ERRNO;
  EBCDIC ARRAY   PATH [0];
  INTEGER        PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	Path definition	The array contains a string that identifies the new working directory.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_CHANGEMODE

The POSIX_CHANGEMODE procedure alters the SECURITYMODE file attribute for a specified disk file. This attribute is an encoded value that provides:

- Owner, group, and other file access permissions flags.
- Set User ID on Execution flag.
- Set Group ID on Execution flag.
- Guard file flags.

The calling process must have appropriate permissions or the effective user ID must match the owner of the file.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_CHANGEMODE (SELECTOR, FILDES, PATH, PATH_OFF,
                                    PATH_LEN, PATH_TYPE, PATH_SEARCHRULE,
                                    MODE, ERRNO);
  VALUE          SELECTOR, FILDES, PATH_OFF, PATH_LEN, PATH_TYPE,
                 PATH_SEARCHRULE, MODE;
  REFERENCE      PATH, ERRNO;
  EBCDIC ARRAY   PATH [0]
  INTEGER        SELECTOR, FILDES, PATH_OFF, PATH_LEN, PATH_TYPE,
                 PATH_SEARCHRULE, MODE, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	Defines the equivalent function: 1 fchmod( ) – (not yet supported) 2 chmod( ) – (directory operations not yet supported) 3 lchmod( ) – (not yet supported)
FILDES	Call-by-value integer	Used only when SELECTOR value is 1. This value is a file descriptor that refers to the open file description for which access permissions will be modified.
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	Path definition	Used only when SELECTOR value is 2 or 3. The array contains a string that identifies the file for which access permissions will be modified.
MODE	Call-by-value integer	This parameter defines the new SECURITYMODE file attribute value. SECURITYMODE is equivalent to the File Access Mode field ([13:14]) within the File Mode. See Table 2-3.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_CHANGEOWNER

The POSIX_CHANGEOWNER procedure changes the OWNER and/or GROUP attribute of a disk file. The calling process must have appropriate permissions or the effective user ID must match the existing owner of the file.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_CHANGEOWNER (SELECTOR, FILDES, PATH, PATH_OFF,
                                     PATH_LEN, PATH_TYPE,
                                     PATH_SEARCHRULE, OWNER, GROUP,
                                     ERRNO);
  VALUE          SELECTOR, FILDES, PATH_OFF, PATH_LEN, PATH_TYPE,
                 PATH_SEARCHRULE, OWNER, GROUP;
  REFERENCE      PATH, ERRNO;
  EBCDIC ARRAY   PATH [0];
  INTEGER        SELECTOR, FILDES, PATH_OFF, PATH_LEN, PATH_TYPE,
                 PATH_SEARCHRULE, OWNER, GROUP, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 fchown( ) - (not yet supported) 2 chown( ) - (directory operations not yet supported) 3 lchown( ) - (not yet supported)
FILDES	Call-by-value integer	Used only when SELECTOR value is 1. This value is a file descriptor that references the open file description for which ownership will be changed.
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	Path definition	Used only when SELECTOR value is 2 or 3. The array contains a string that identifies the file for which ownership will be changed.
OWNER	Call-by-value integer	This input parameter contains the user ID associated with the new OWNER attribute of the file.
GROUP	Call-by-value integer	This input parameter contains the group ID associated with the new GROUP attribute of the file.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_CLOSE

The POSIX_CLOSE procedure closes a specified file descriptor. The POSIX_ALLOCATE_FD or POSIX_FILE_TO_FD procedures allocate these file descriptors.

Supported Functions

POSIX_CLOSE provides a function equivalent to the C language close( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_CLOSE (FILDES, OPTION, CLOSERESULT, ERRNO);
  VALUE     FILDES, OPTION;
  REFERENCE CLOSERESULT, ERRNO;
  INTEGER   FILDES, OPTION, CLOSERESULT, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
FILDES	Call-by-value integer	This input parameter contains the file descriptor.
OPTION	Call-by-value integer	See “Option parameter (CLOSE)” in Section 2 for a list of defined values.
CLOSERESULT	Call-by-reference integer	This output parameter contains the result returned by the FIBCLOSE procedure (if applicable). This result format is equivalent to the format provided in the AVAILABLE attribute.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_EXECVE

The POSIX_EXECVE procedure executes a specified code file. There is no return from a successful operation—the new process image overlays the previous process image.

Supported Functions

POSIX_EXECVE provides a function equivalent to the C language execve( ) function; there is no support for other exec( ) family functions.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_EXECVE (PATH, PATH_OFF, PATH_LEN, PATH_TYPE,
                                PATH_SEARCHRULE, ARGV, ARGV_OFF,
                                ARGV_LEN, ARGVINDX, ARGVINDX_OFF,
                                ARGVINDX_LEN, ARGVSZ, ARGVSZ_OFF,
                                ARGVSZ_LEN, ENVP, ENVP_OFF, ENVP_LEN,
                                ENVPINDX, ENVPINDX_OFF, ENVPINDX_LEN,
                                ENVPSZ, ENVPSZ_OFF, ENVPSZ_LEN, ERRNO);
  VALUE         PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE,
                ARGV_OFF, ARGV_LEN, ARGVINDX_OFF, ARGVINDX_LEN,
                ARGVSZ_OFF, ARGVSZ_LEN, ENVP_OFF, ENVP_LEN,
                ENVPINDX_OFF, ENVPINDX_LEN, ENVPSZ_OFF, ENVPSZ_LEN;
  REFERENCE     PATH, ARGV, ARGVINDX, ARGVSZ, ENVP, ENVPINDX, ENVPSZ,
                ERRNO;
  EBCDIC ARRAY  PATH, ARGV, ENVP [0];
  INTEGER ARRAY ARGVINDX, ARGVSZ, ENVPINDX, ENPSZ [0];
  INTEGER       PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE,
                ARGV_OFF, ARGV_LEN, ARGVINDX_OFF, ARGVINDX_LEN,
                ARGVSZ_OFF, ARGVSZ_LEN, ENVP_OFF, ENVP_LEN, ENVPINDX_OFF,
                ENVPINDX_LEN, ENVPSZ_OFF, ENVPSZ_LEN, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
PATH, PATH_OFF, PATH_TYPE, PATH_TYPE, PATH_SEARCHRULE	Path definition	The array contains a string that defines the pathname of the code file (program).
ARGV, ARGV_OFF, ARGV_LEN	EBCDIC array input	This array contains a string defining program arguments (parameters). The ARGV_OFF and ARGV_LEN parameters must contain 0.
ARGVINDX, ARGVINDX_OFF, ARGVINDX_LEN	Integer array input	The array contains ARGV array indexes. Each index points to the start of an argument.
ARGVSZ, ARGVSZ_OFF, ARGVSZ_LEN	Integer array input	The array contains the length of each argument (located in ARGV) pointed to the indexes in ARGVINDX.
ENVP, ENVP_OFF, ENVP_LEN	EBCDIC array input	The array contains a string that defines environment variables for the program. The ENVP_OFF and ENVP_LEN parameters must contain 0.
ENVPINDX, ENVPINDX_OFF, ENVPINDX_LEN	Integer array input	This array contains ENVP array indexes. Each index points to the start of an environment variable.
ENVPSZ, ENVPSZ_OFF, ENVPSZ_LEN	Integer array input	This array contains the length of each environment variable (located in ENVP) pointed to the indexes in ENVPINDX.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_EXIT

The POSIX_EXIT procedure terminates the calling process. It then stores exit status in the EXIT_STATUS TAB word and passes control back to the system. POSIX_EXIT never returns a result or sets an ERRNO value.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_EXIT (SELECTOR, STATUS, ERRNO);
  VALUE     STATUS;
  REFERENCE ERRNO;
  INTEGER   STATUS, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 _exit( ) 2 exit( )
STATUS	Call-by-value integer	This input parameter contains termination status. Termination status values can range from –127 to 127. A value of 0 indicates success; all other values indicate failure.
ERRNO	ERRNO	Exit functions never return an error value.
<result>	Integer result	No result is returned.

POSIX_FCNTL

The POSIX_FCNTL procedure can perform a variety of control functions on a specified open file.

Supported Functions

POSIX_FCNTL provides functions equivalent to the C language fcntl( ) function as well as two derivative functions, dup( ) and dup2( ).

The CMD parameter specifies the function to be performed.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_FCNTL (FILDES, CMD, INTARG, STRUCTARG,
                               STRUCTARG_OFF, STRUCTARG_LEN, ERRNO);
  VALUE       FILDES, CMD, INTARG, STRUCTARG_OFF, STRUCTARG_LEN;
  REFERENCE   STRUCTARG, ERRNO;
  REAL ARRAY  STRUCTARG [0];
  INTEGER     FILDES, CMD, INTARG, STRUCTARG_OFF, STRUCTARG_LEN, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
FILDES	Call-by-value integer	This input parameter contains a file descriptor that references the applicable open file description.
CMD	Call-by-value integer	See “CMD parameter (FCNTL)” in Section 2 for a summary of defined values.
INTARG	Call-by-value integer	See “INTARG parameter (FCNTL)” in Section 2 for information on this parameter.
STRUCTARG, STRUCTARG_OFF, STRUCTARG_LEN	Structure array input Structure array output	This structure argument is only used with the following commands: F_GETLK command: Input parameter: Contains description of a lock. See “FLOCK structure” in Section 2. Output parameter: The parameter is overwritten with information about the lock that blocks the lock description provided in the input parameter. If a blocking lock was not found, only the L_TYPE field contains valid data. F_SETLK and F_SETLKW commands: Input parameter: Specifies a file segment region to be locked or unlocked. See “FLOCK structure” in Section 2.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_FILESTATUS

The POSIX_FILESTATUS procedure obtains file status information for a specified file. The file can be specified by filename or by file descriptor.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_FILESTATUS (SELECTOR, FILDES, PATH, PATH_OFF,
                                    PATH_LEN, PATH_TYPE, PATH_SEARCHRULE,
                                    BUF, BUF_OFF, BUF_MAX, ERRNO);
  VALUE          SELECTOR, FILDES, PATH_OFF, PATH_LEN,
                 PATH_TYPE, PATH_SEARCHRULE, BUF_OFF, BUF_MAX;
  REFERENCE      BUF, ERRNO;
  EBCDIC ARRAY   PATH [0];
  REAL ARRAY     BUF [0]
  INTEGER        SELECTOR, FILDES, PATH_OFF, PATH_LEN, PATH_TYPE,
                 PATH_SEARCHRULE, BUF_OFF, BUFF_MAX, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 fstat( ) 2 stat( ) 3 lstat( ) - (not yet supported) 5 _MCPfstat( ) 6 _MCPstat( ) 7 _MCPlstat( ) - (not yet supported) 9 readlink( ) - (not yet supported)

FILDES	Call-by-value integer	Used only with the following equivalent functions:
		fstat( )
		_MCPfstat( )
		This input parameter contains a file descriptor that refers to an open file description.

PATH,
PATH_OFF,
PATH_LEN,
PATH_TYPE,
PATH_SEARCHRULE
	Path definition	Used only with the following equivalent functions:
		stat( )
		lstat( )
		_MCPstat( )
		_MCPlstat( )
		readlink( )
		The array contains a string that identifies the filename.

BUF, BUF_OFF, BUF_MAX	Structure array output	An array output parameter that contains the STAT or MCPSTAT data structure. See “STAT structure” or “MCPSTAT structure” in Section 2 for a definition of this information.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_FILE_TO_FD

The POSIX_FILE_TO_FD procedure allocates a new file descriptor and attaches it to a specified existing file. This procedure cannot process direct files.

The POSIX_FILE_TO_FD procedure:

- Initializes the FD_VECTOR array of the calling process (only required on the first file descriptor request).
- Searches the FD_VECTOR array for lowest available file descriptor. It resizes the array if necessary.
- Associates newly allocated file descriptor with the file specified in the FYLE parameter.
- Initializes the specified file and applies all file attributes provided in the FILE declaration.
- Sets the FD_CLOEXEC flag for the file descriptor.

Supported Functions

There is no equivalent C language function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_FILE_TO_FD (FYLE, ERRNO);
  REFERENCE FYLE, ERRNO;
  FILE      FYLE;
  INTEGER   ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
FYLE	File declaration	This input parameter defines a file. This file must be programmatically declared and located in the current process stack.
ERRNO	Call-by-value integer	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_FORK

The POSIX_FORK procedure creates a new process. This new process is known as a *child process*.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure:

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_FORK (SELECTOR, ERRNO);
  VALUE       SELECTOR;
  REFERENCE   ERRNO;
  INTEGER     SELECTOR, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 fork( )
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_GETGRINFO

The POSIX_GETGRINFO procedure obtains the GROUP structure associated with a specified group ID or group name.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure:

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_GETGRINFO (SELECTOR, GID, NAME, NAME_OFF,
                                   NAME_LEN, MEM, MEM_OFF, MEM_MAX,
                                   ERRNO);
  VALUE        SELECTOR, GID, NAME_OFF, NAME_LEN, MEM_OFF, MEM_MAX;
  REFERENCE    NAME, MEM, ERRNO;
  EBCDIC ARRAY NAME [0];
  REAL ARRAY   MEM [0];
  INTEGER      SELECTOR, GID, NAME_OFF, NAME_LEN, MEM_OFF, MEM_MAX,
               ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 getgrgid( ) 2 getgrnam( )
GID	Call-by-value integer	Used only the when the SELECTOR value is 1. This input parameter contains a group ID.
NAME, NAME_OFF, NAME_LEN	EBCDIC array input	Used only when the SELECTOR value is 2. The array contains a string that identifies a group name.
MEM, MEM_OFF, MEM_MAX	Structure array output	This real array contains the requested structure. See “GROUP structure” in Section 2 for a definition of this structure.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_GETPWINFO

The POSIX_GETPWINFO procedure returns the PASSWD structure for a specified user ID or user name. This structure does not include the actual password.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure:

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_GETPWINFO (SELECTOR, NAME, NAME_OFF, NAME_LEN,
                                   UID, PASSWD, PASSWD_OFF, PASSWD_MAX,
                                   ERRNO);
  VALUE        SELECTOR, NAME_OFF, NAME_LEN, UID, PASSWD_OFF, PASSWD_MAX;
  REFERENCE    NAME, PASSWD, ERRNO;
  EBCDIC ARRAY NAME [0];
  REAL ARRAY   PASSWD [0];
  INTEGER      SELECTOR, NAME_OFF, NAME_LEN, UID, PASSWD_OFF,
               PASSWD_MAX, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 getpwuid( ) 2 getpwnam( )
NAME, NAME_OFF, NAME_LEN	EBCDIC array input	Used only when the SELECTOR value is 2. The array contains the user name string for which a PASSWD structure is required.
UID	Call-by-value integer	Used only when the SELECTOR value is 1. This input parameter contains the user ID for which a PASSWORD structure is required.
PASSWD, PASSWD_OFF, PASSWD_MAX	Structure array output	This real array contains the requested structure. See “PASSWD structure” in Section 2 for a definition of this structure.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_GROUPLIST

The POSIX_GROUPLIST procedure obtains all supplementary group IDs associated with the calling process.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_GROUPLIST (SELECTOR, GROUPLIST, GROUPLIST_OFF,
                                   GROUPLIST_MAX, ERRNO);
  VALUE          SELECTOR, GROUPLIST_OFF, GROUPLIST_MAX;
  REFERENCE      GROUPLIST, ERRNO;
  INTEGER ARRAY  GROUPLIST [0];
  INTEGER        SELECTOR, GROUPLIST_OFF, GROUPLIST_MAX, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 getgroups( ) 2 setgroups( ) - (not yet supported)
GROUPLIST, GROUPLIST_OFF, GROUPLIST_MAX	Integer array output	This integer array receives the requested supplementary group IDs. Note that GROUPLIST_MAX provides a function equivalent to the C function *gidsetsize* argument.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_INTEGERIDS

The POSIX_INTEGERIDS procedure obtains various ID values associated with the calling process.

Supported Functions

The SELECTOR parameter defines the functions supported by this procedure. Note that SELECTOR values 8 and 13 provide a pair of special functions:

- SELECTOR value = 8 (Get Exit Type)

  This function determines the last exit type performed by a child process. This returned value indicates the exit type:

  - 0 indicates child process performed an exit( ) function (cleanup is required)
  - 1 indicates child process performed an _exit( ) function

- SELECTOR value = 13 (FD Vector Allocated)

  This function determines if the POSIX_ALLOCATE_FD or POSIX_FILE_TO_FD library procedure has allocated an FD vector array to the calling process. This returned value indicates allocation status:

  - 0 indicates FD vector is not allocated.
  - 1 indicates FD vector is allocated.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_INTEGERIDS (SELECTOR, INFO, ERRNO);
  VALUE          SELECTOR, INFO;
  REFERENCE      ERRNO;
  INTEGER        SELECTOR, INFO, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 getpid( ) 2 GETPGID - (not a C function - described in Section 3) 2 getpgrp( ) 3 getppid( ) 4 getuid( ) 5 geteuid( ) 6 getgid( ) 7 getegid( ) 8 Get Exit Type 9 GETSID - (not a C function - described in Section 3) 10 getsgid( ) - (not yet supported) 11 getsuid( ) - (not yet supported) 13 FD Vector Allocated
INFO	Call-by-value integer	Not used.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_NANOALARM

The POSIX_NANOALARM procedure causes the system to send a signal (type SIGALRM) to the calling process following a specified period of time. This procedure can also cancel a pending request of this type.

Supported Functions

POSIX_NANOALARM provides a function equivalent to the C language alarm( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
REAL PROCEDURE POSIX_NANOALARM (SECS, ERRNO);
  VALUE          SECS;
  REFERENCE      ERRNO;
  REAL           SECS;
  INTEGER        ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SECS	Call-by-value real	This input parameter specifies a delay period (in seconds). If this value is 0, any pending signal is canceled.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_NANOSLEEP

The POSIX_NANOSLEEP procedure suspends the calling process for a specified amount of time. A signal may reactivate the suspended process.

Supported Functions

POSIX_NANOSLEEP provides a function equivalent to the C language sleep( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
REAL PROCEDURE POSIX_NANOSLEEP (SECS, ERRNO);
  VALUE          SECS;
  REFERENCE      ERRNO;
  REAL           SECS;
  INTEGER        ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SECS	Call-by-value real	This input parameter specifies the process suspension time (in seconds).
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_OPEN

The POSIX_OPEN procedure opens a file (referenced by file descriptor) with specified truncate and open type options.

This procedure supports the final phase of a POSIX.1 open( ) or creat( ) function. The following procedures must precede POSIX_OPEN:

- The POSIX_ALLOCATE_FD procedure (to establish a file descriptor value)
- Multiple POSIX_FILEATTRIBAGENT procedures (to get or set required file attributes)

Note that the POSIX_FILEATTRIBAGENT procedure is not yet supported.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_OPEN (FILDES, OPTION, OPENRESULT, ERRNO);
  VALUE     FILDES, OPTION;
  REFERENCE OPENRESULT, ERRNO;
  INTEGER   FILDES, OPTION, OPENRESULT, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
FILDES	Call-by-value integer	This input parameter specifies a file descriptor for the file to be opened.
OPTION	Call-by-value integer	This input parameter contains the option values passed to the FIBOPEN routine. See “Option parameter (OPEN)” in Section 2 for additional information.
OPENRESULT	Call-by-reference integer	This output parameter contains the result returned by the logical I/O open performed by FIBOPEN. The format is identical to that defined for the AVAILABLE attribute. Refer to the *File Attributes Programming Reference Manual* for information on results returned for the open operation.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_PATHCONF

The POSIX_PATHCONF procedure obtains configuration variable information about a specified open file or path. The path can be specified directly or by file descriptor.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_PATHCONF (SELECTOR, FILDES, PATH, PATH_OFF,
                                  PATH_LEN, PATH_TYPE, PATH_SEARCHRULE,
                                  NAME, ERRNO);
  VALUE         SELECTOR, FILDES, PATH_OFF, PATH_LEN, PATH_TYPE,
                PATH_SEARCHRULE, NAME;
  REFERENCE     PATH, ERRNO;
  EBCDIC ARRAY  PATH [0];
  INTEGER       SELECTOR, FILDES, PATH_OFF, PATH_LEN, PATH_TYPE,
                PATH_SEARCHRULE, NAME, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function to be performed: 1 fpathconf( ) 2 pathconf( )
FILDES	Call-by-value integer	Used only when the SELECTOR parameter value is 1. This input parameter contains a file descriptor that refers to an open file description.
PATH, PATH_OFF, PATH_LEN, PATH_TYPE, PATH_SEARCHRULE	Path definition	Used only when the SELECTOR parameter value is 2. The array contains a string that identifies a pathname.
NAME	Call-by-value integer	This input parameter specifies the configurable variable to be interrogated. See "NAME parameter (for PATHCONF function)" in Section 2 for a summary of configurable variables and associated NAME parameter integers.
ERRNO	ERRNO	This returned value indicates error status. See "ERRNO" in Section 2.
<result>	Integer result	A function-specific result is returned. See "Result (Integer or Real)" in Section 2.

POSIX_PIPE

The POSIX_PIPE procedure creates a half-duplex interprocess channel known as a *pipe*. Two file descriptors (one read-only and the other write-only) are passed back to the program.

Supported Functions

POSIX_PIPE provides a function equivalent to the C language pipe( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_PIPE (FILDES_IN, FILDES_OUT, ERRNO);
  REFERENCE FILDES_IN, FILDES_OUT, ERRNO;
  INTEGER   FILDES_IN, FILDES_OUT, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
FILDES_IN	Call-by-reference integer	This output parameter contains the file descriptor associated with the read side of the pipe.
FILDES_OUT	Call-by-reference integer	This output parameter contains the file descriptor associated with the write side of the pipe.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEEK

The POSIX_SEEK procedure repositions the current record pointer in an open file description.

Supported Functions

POSIX_SEEK provides a function equivalent to the lseek( ) or seekdir( ) C language functions. The seekdir( ) function is not yet supported.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEEK (FILDES, RELATIVERECORD, WHENCE, IORESULT,
                              ERRNO);
  VALUE     FILDES, RELATIVERECORD, WHENCE;
  REFERENCE IORESULT, ERRNO;
  INTEGER   FILDES, RELATIVERECORD, WHENCE, ERRNO;
  REAL      IORESULT;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
FILDES	Call-by-value integer	This input parameter specifies a file descriptor that references the applicable open file description.
RELATIVERECORD	Call-by-value integer	This input parameter specifies a record count. See the WHENCE parameter.
WHENCE	Call-by-value integer	This input parameter specifies a seek method. There are three possible values: 0 = SEEK_SET Seek to record specified by RELATIVERECORD. 1 = SEEK_CUR Space RELATIVERECORD records (negative values cause backward space). 2 = SEEK_END Space RELATIVERECORD records from the current end-of-file (negative values indicate backward space).
IORESULT	Call-by-reference real	This output parameter receives the result returned by FIBSTACK.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_CLOSE

The POSIX_SEM_CLOSE procedure closes a named semaphore that is currently open.

When a process closes a semaphore, that process no longer has access to it. However, closing a named semaphore does not remove it from the system.

Supported Functions

POSIX_SEM_CLOSE provides a function equivalent to the C language sem_close( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_CLOSE (SEM, ERRNO);
  VALUE          SEM;
  REFERENCE      ERRNO;
  INTEGER        SEM, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEM	Call-by-value integer	This input parameter contains a semaphore identifier.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_DESTROY

The POSIX_SEM_DESTROY procedure destroys an unnamed semaphore. Destroying a semaphore removes it from the system.

Supported Functions

POSIX_SEM_DESTROY provides a function equivalent to the C language sem_destroy( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_DESTROY (SEM, ERRNO);
  VALUE          SEM;
  REFERENCE      ERRNO;
  INTEGER        SEM, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEM	Call-by-value integer	This input parameter contains a semaphore identifier.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_GETVALUE

The POSIX_SEM_GETVALUE procedure retrieves the value of a specified named or unnamed semaphore.

The retrieved value reflects an actual value of the semaphore at some time during the call. This may not be the actual value of the semaphore at the time the procedure returns.

Supported Functions

POSIX_SEM_GETVALUE provides a function equivalent to the C language sem_getvalue( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_GETVALUE (SEM, SVAL, ERRNO);
  VALUE          SEM;
  REFERENCE      SVAL, ERRNO;
  INTEGER        SEM, SVAL, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEM	Call-by-value integer	This input parameter contains a semaphore identifier.
SVAL	Call-by-reference integer	This output parameter contains the value of the specified semaphore. When the procedure completes successfully, this parameter contains either: • A positive integer, indicating that the semaphore is unlocked. • A zero, indicating that the semaphore is locked.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_INIT

The POSIX_SEM_INIT procedure creates an unnamed semaphore.

Supported Functions

POSIX_SEM_INIT provides a function equivalent to the C language sem_init( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_INIT (SEM, PSHARED, VAL, ERRNO);
  VALUE          PSHARED, VAL;
  REFERENCE      SEM, ERRNO;
  INTEGER        SEM, PSHARED, VAL, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEM	Call-by-reference integer	This output parameter contains a returned value that identifies the unnamed semaphore.
PSHARED	Call-by-value integer	This input parameter indicates if the semaphore can be shared. Possible values are: Zero No sharing allowed Non zero Sharing allowed
VAL	Call-by-value integer	This input parameter indicates the initial value of the semaphore (it must be positive).
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_OPEN

The POSIX_SEM_OPEN procedure opens a named semaphore. The procedure can either:

- Access an existing named semaphore
- Create a new named semaphore

Once opened, a process can use the semaphore until it closes the semaphore with a POSIX_SEM_CLOSE call.

Supported Functions

POSIX_SEM_OPEN provides a function equivalent to the C language sem_open( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_OPEN (NAME, NAME_OFF, NAME_LEN, OFLAG,
                                  MODE, VAL, ERRNO);
  VALUE         NAME_OFF, NAME_LEN, OFLAG, MODE, VAL;
  REFERENCE     NAME, ERRNO;
  EBCDIC ARRAY  NAME [0]
  INTEGER       NAME_OFF, NAME_LEN, OFLAG, MODE, VAL, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
NAME, NAME_OFF, NAME_LEN	EBCDIC array input	The array contains the name of the semaphore. This name must conform to POSIX pathname naming rules.
OFLAG	Call-by-value integer	This input parameter specifies whether the procedure is to open an existing semaphore or create a new semaphore. There are two flag bits: [15:01] O_CREAT [17:01] O_EXCL
MODE	Call-by-value integer	This input parameter contains permission bits for the new semaphore. It is valid only when OFLAG specifies the O_CREAT flag bit. See “MODE parameter” in Section 2 for a description of bit assignments.
VAL	Call-by-value integer	This input parameter contains the initial value of a new semaphore. It is valid only when OFLAG specifies the O_CREAT flag bit.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_POST

The POSIX_SEM_POST procedure adds one to the value of a named or unnamed semaphore. A semaphore is unlocked when it has a value greater than 0.

Supported Functions

POSIX_SEM_POST provides a function equivalent to the C language sem_post( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_POST (SEM, ERRNO);
  VALUE          SEM;
  REFERENCE      ERRNO;
  INTEGER        SEM, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEM	Call-by-value integer	This input parameter contains a semaphore identifier.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_TRYWAIT

The POSIX_SEM_TRYWAIT operates on named or unnamed semaphores as follows:

If the semaphore value is . . .	**Then POSIX_SEM_TRYWAIT . . .**
greater than 0	decrements the semaphore value by 1 and returns immediately.
0	returns an EAGAIN error. The value of the semaphore is not changed.

Supported Functions

POSIX_SEM_TRYWAIT provides a function equivalent to the C language sem_trywait( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_TRYWAIT (SEM, ERRNO);
  VALUE          SEM;
  REFERENCE      ERRNO;
  INTEGER        SEM, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEM	Call-by-value integer	This input parameter contains a semaphore identifier.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_UNLINK

The POSIX_SEM_UNLINK procedure deletes the name of a semaphore from the system table.

Supported Functions

POSIX_SEM_UNLINK provides a function equivalent to the C language sem_unlink( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_UNLINK (NAME, NAME_OFF, NAME_LEN, ERRNO);
  VALUE          NAME_OFF, NAME_LEN;
  REFERENCE      NAME, ERRNO;
  EBCDIC ARRAY   NAME [0]
  INTEGER        NAME_OFF, NAME_LEN, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
NAME, NAME_OFF, NAME_LEN	EBCDIC array input	The array contains the name of the applicable semaphore. This name must conform to POSIX pathname naming rules.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SEM_WAIT

The POSIX_SEM_WAIT procedure decrements the value of an unlocked named or unnamed semaphore. If the semaphore is already locked (that is, has a value of 0), the procedure waits until it is unlocked by another process.

Supported Functions

POSIX_SEM_WAIT provides a function equivalent to the C language sem_wait( ) function.

Procedure Declaration

You declare the procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SEM_WAIT (SEM, ERRNO);
  VALUE         SEM;
  REFERENCE     ERRNO;
  INTEGER       SEM, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SEM	Call-by-value integer	This input parameter contains a semaphore identifier.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SETIDS

The POSIX_SETIDS procedure changes either:

- Various values associated with the calling process
- The process group ID of a specified process.

Supported Functions

The SELECTOR parameter defines the functions supported by this procedure.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SETIDS (SELECTOR, PID, INFO, ERRNO);
  VALUE      SELECTOR, PID, INFO;
  REFERENCE  ERRNO;
  INTEGER    SELECTOR, PID, INFO, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 setuid( ) 2 setgid( ) 3 setsid( ) 3 SETPGRP - (not a C function - described in Section 3) 4 setpgid( ) 5 umask( ) 6 NICE - (not a C function - described in Section 3) 7 seteuid( ) - (not yet supported) 8 setegid( ) - (not yet supported)
PID	Call-by-value integer	This input parameter is only valid if the SELECTOR value is 4. For the setpgid( ) function, it contains a process ID.
INFO	Call-by-value integer	This input parameter identifies a value appropriate for the specified SELECTOR function. For SELECTOR values 1 or 7, this value represents a user ID. For SELECTOR values 2 or 8, this value represents a group ID. For SELECTOR value 4, this value represents a process group ID. For SELECTOR value 5, this value represents the file mode creation mask. See “INFO parameter” in Section 2 for a description of this mask.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SIGHANDLER

The POSIX_SIGHANDLER procedure provides a number of functions associated with signals.

Supported Functions

The SELECTOR parameter defines the C function invoked by this procedure.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SIGHANDLER (SELECTOR, SIG, ACT_PROC, VAR1, VAR2,
                                    OVAR1, ACT, ACT_OFF, ACT_LEN, OACT,
                                    OACT_OFF, OACT_MAX, ERRNO);
  VALUE          SELECTOR, SIG, VAR1, VAR2, ACT_OFF, ACT_LEN,
                 OACT_OFF, OACT_MAX;
  REFERENCE      ACT_PROC, OVAR1, ACT, OACT, ERRNO;
  INTEGER        SELECTOR, SIG, VAR1, VAR2, OVAR1, ACT_OFF, ACT_LEN,
                 OACT_OFF, OACT_MAX, ERRNO;
  REAL ARRAY     ACT, OACT [0];

INTEGER PROCEDURE ACT_PROC (INFO1, INFO2, INFO3, INFO4, INFO5,
                            INFO6, INFO7, INFO8, INFO9, INFO10);
  VALUE          INFO1, INFO2, INFO3, INFO4, INFO5,
                 INFO6, INFO7, INFO8, INFO9, INFO10;
  INTEGER        INFO1, INFO2, INFO3, INFO4, INFO5,
                 INFO6, INFO7, INFO8, INFO9, INFO10;
FORMAL;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent C function or macro: 1 sigaction( ) 2 sigset( ) 3 signal( ) 4 sighold( ) 5 sigrelse( ) 6 sigignore( ) 7 sigprocmask( ) 8 sigpending( ) 9 pause( ) 10 sigsuspend( ) 11 raise( ) 12 kill( ) 13 sigpause( ) 14 sigpush( ) macro No function uses all parameters. See "Parameter Usage for Specific Functions" following this table for a summary of the parameters used by each function.
SIG	Call-by-value integer	This input parameter defines the signal type. See "SIG Parameter" in Section 2 for an enumeration of signal types.
ACT_PROC	Signal handler procedure	ACT_PROC defines the procedure invoked when the specified signal occurs. It is an integer procedure with ten integer variables. See "Signal handler procedure" in Section 2 for additional information.
VAR1	Call-by-value integer	The meaning of this input parameter depends on the function. See "Parameter Usage for Specific Functions" following this table for further information.
VAR2	Call-by-value integer	The meaning of this input parameter depends on the function. See "Parameter Usage for Specific Functions" following this table for further information
OVAR1	Call-by-value integer	The meaning of this output parameter depends on the function. See "Parameter Usage for Specific Functions" following this table for further information

Parameter	Rule	Description
ACT, ACT_OFF, ACT_LEN	Structure array input	The array contains a SIGACTION structure defining action to be assigned to the signal type. See “SIGACTION structure” in Section 2 for additional information.
OACT, OACT_OFF, OACT_MAX	Structure array output	The array contains the SIGACTION structure currently assigned to the signal type. It consists of the same three words detailed in the ACT array description.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

Parameter Usage for Specific Functions

Equivalent C Function	Parameters Used
sigaction( )	SELECTOR SIG ACT_PROC ACT, ACT_OFF, ACT_LEN OACT, OACT_OFF, OACT_LEN ERRNO
sigset( ), signal( )	SELECTOR SIG ACT_PROC VAR1 - Serves as the equivalent of the disp argument. See “DISP parameter” in Section 2 for additional information. ERRNO
sighold( ), sigrelse( ), sigignore( ), raise( ), sigpause( )	SELECTOR SIG ERRNO

Equivalent C Function	Parameters Used
sigprocmask( )	SELECTOR VAR1- Serves as the equivalent of the how argument: See “HOW parameter” in Section 2 for additional information. VAR2 - Serves as the equivalent of the set argument. Identifies the set of signal types to be added to, removed from, or used as, the process’s signal mask (blocked signals). OVAR1 - Serves as the equivalent of the oset argument. Identifies the current process’s signal mask (blocked signals). ERRNO
sigpending( )	SELECTOR OVAR1 - Serves as the equivalent of the set argument. Identifies the set of pending signals. ERRNO
pause( ), sigpush( ) macro	SELECTOR ERRNO
sigsuspend( )	SELECTOR VAR1 - Serves as the equivalent of the sigmask argument. Identifies the local set of signal types. ERRNO
kill( )	SELECTOR SIG VAR1 - Serves as the equivalent of the pid argument (indicates receiving process or processes. See “PID parameter (KILL)” in Section 2 for additional information. ERRNO

POSIX_SREAD_x

Two procedures are available to read data from a file associated with a specified file descriptor.

- POSIX_SREAD_E transfers data into a specified EBCDIC buffer array.
- POSIX_SREAD_R transfers data into a specified real buffer array.

These procedures are cover functions to the FIBSTACK routines; each procedure obtains a FIB reference from the specified file descriptor, generates appropriate parameters, and enters the appropriate FIBSTACK routine.

Functions Supported:

The POSIX_SREAD_x procedures provide a function equivalent to the C language read( ) function.

POSIX_SREAD_E Procedure Declaration

You declare a POSIX_SREAD_E procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SREAD_E (FILDES, EBUF, EBUF_OFF, SYZE, IORESULT,
                                 ERRNO);
  VALUE       FILDES, EBUF_OFF, SYZE;
  REFERENCE   EBUF, IORESULT, ERRNO;
  INTEGER     FILDES, EBUF_OFF, ERRNO;
  REAL        SYZE, IORESULT;
  EBCDIC ARRAY EBUF [0];
LIBRARY MCPSUPPORT;
```

POSIX_SREAD_R Procedure Declaration

You declare a POSIX_SREAD_R procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SREAD_R (FILDES, RBUF, RBUF_OFF, SYZE, IORESULT,
                                 ERRNO);
  VALUE     FILDES, RBUF_OFF, SYZE;
  REFERENCE RBUF, IORESULT, ERRNO;
  INTEGER   FILDES, RBUF_OFF, ERRNO;
  REAL      SYZE, IORESULT;
  REAL ARRAY RBUF [0];
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
FILDES	Call-by-value integer	This input parameter contains a file descriptor that references the applicable open file description.
xBUF, xBUF_OFF	EBCDIC array input or Real array input	The array that receives serial read data (EBCDIC or real).
SYZE	Call-by-value real	This input parameter specifies the number of FRAMESIZE units to transfer into the array.
IORESULT	Call-by-reference real	This output parameter receives the FIBSTACK result from the I/O operation. The format is identical to that defined for the STATE attribute.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_STRINGIDS

The POSIX_STRINGIDS procedure retrieves information associated with the calling process or a specified file descriptor.

Supported Functions

The SELECTOR parameter defines the seven functions supported by this procedure.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_STRINGIDS (SELECTOR, INFO, S, S_OFF, S_MAX, ERRNO);
  VALUE         SELECTOR, INFO, S_OFF, S_MAX;
  REFERENCE     S, ERRNO;
  EBCDIC ARRAY  S [0];
  INTEGER       SELECTOR, INFO, S_OFF, S_MAX, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter defines the equivalent function: 1 ctermid( ) - (not yet supported) 2 cuserid( ) 3 getlogin( ) 4 ttyname( ) - (not yet supported) 5 getcwd( ) 6 GETUSERNAME - (not a C function - see Section 3) 7 GETUSERID - (not a C function - see Section 3)
INFO	Call-by-value integer	This input parameter is only valid if the SELECTOR value is 4 or 6. For the ttyname( ) function, INFO specifies the file descriptor associated with the terminal device. For the GETUSERNAME function, INFO specifies a user ID value.
S, S_OFF, S_MAX	EBCDIC array output EBCDIC array input	For all functions except GETUSERID, the output array receives the requested sting. If the SELECTOR value is 7, the input array contains a user name.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SWRITE_x

Two procedures are available to write data to the file associated with a specified file descriptor.

- POSIX_SWRITE_E transfers data from a specified EBCDIC buffer array.
- POSIX_SWRITE_R transfers data from a specified real buffer array.

These procedures are cover functions to the FIBSTACK routines; each procedure obtains a FIB reference from the specified file descriptor, generates appropriate parameters, and enters the appropriate FIBSTACK routine.

Functions Supported:

The POSIX_SWRITE_x procedures provide a function equivalent to the C language write( ) function.

POSIX_SWRITE_E Procedure Declaration

You declare a POSIX_SWRITE_E procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SWRITE_E (FILDES,  EBUF, EBUF_OFF, SYZE, IORESULT,
                                 ERRNO);
  VALUE        FILDES, EBUF_OFF, SYZE;
  REFERENCE    EBUF, IORESULT, ERRNO;
  INTEGER      FILDES, EBUF_OFF, ERRNO;
  REAL         SYZE, IORESULT;
  EBCDIC ARRAY EBUF [0];
LIBRARY MCPSUPPORT;
```

POSIX_SWRITE_R Procedure Declaration

You declare a POSIX_SWRITE_R procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_SWRITE_R (FILDES,  RBUF, RBUF_OFF, SYZE, IORESULT,
                                 ERRNO);
  VALUE      FILDES, RBUF_OFF, SYZE;
  REFERENCE  RBUF, IORESULT, ERRNO;
  INTEGER    FILDES, RBUF_OFF, ERRNO;
  REAL       SYZE, IORESULT;
  REAL ARRAY RBUF [0];
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
FILDES	Call-by-value integer	This input parameter contains a file descriptor that references the applicable open file description.
xBUF, xBUF_OFF	EBCDIC array input or Real array input	The array is the source of serial write data (EBCDIC or real).
SYZE	Call-by-value real	This input parameter specifies the number of FRAMESIZE units to transfer from the array.
IORESULT	Call-by-reference real	This output parameter receives the FIBSTACK result from the I/O operation. The format is identical to that defined for the STATE attribute.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_SYSCONF

The POSIX_SYSCONF procedure determines the current value of a specified configurable system variable.

Supported Functions

POSIX_SYSCONF provides a function equivalent to the C language sysconf( ) function.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
REAL PROCEDURE POSIX_SYSCONF (NAME, ERRNO);
  VALUE          NAME;
  REFERENCE      ERRNO;
  INTEGER        NAME, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
NAME	Call-by-value integer	This input parameter specifies a value representing the applicable configurable system variable. See “Name parameter (SYSCONF)” in Section 2 for a description of the variable associated with each integer value.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_TIME

The POSIX_TIME procedure obtains a value (number of seconds since 00:00:00 Greenwich Mean Time on January 1, 1970) that represents the current time. This value is returned as an integer result.

Supported Functions

POSIX_TIME provides a function equivalent to the C language time( ) function.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_TIME (ERRNO);
  REFERENCE ERRNO;
  INTEGER   ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	<real result>	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_TIMES

The POSIX_TIMES procedure returns time accounting information for the current process and its child processes.

Supported Functions

POSIX_TIMES provides a function equivalent to the C language times( ) function.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_TIMES (BUF, BUF_OFF, BUF_MAX, BUF_LEN, ERRNO);
  VALUE         BUF_OFF, BUF_MAX;
  REFERENCE     BUF, BUF_LEN, ERRNO;
  REAL ARRAY    BUF [0];
  INTEGER       BUF_OFF, BUF_MAX, BUF_LEN, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
BUF, BUF_OFF, BUF_MAX	Real array output	The array holds requested time accounting information. See “TMS structure” in Section 2 for a definition of this data.
BUF_LEN	Call-by-reference integer	This output parameter indicates the number of words written into the BUF array.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_UNAME

The POSIX_UNAME procedure returns hardware and software information about the host system processing environment.

Supported Functions

POSIX_UNAME provides a function equivalent to the C language uname( ) function.

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_UNAME (NAME, NAME_OFF, NAME_MAX, ERRNO);
  VALUE         NAME_OFF, NAME_MAX;
  REFERENCE     NAME, ERRNO;
  REAL ARRAY    NAME [0];
  INTEGER       NAME_OFF, NAME_MAX, ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
NAME, NAME_OFF, NAME_MAX	Real array output	The array holds requested system information. See “UTSNAME structure” in Section 2 for a definition of this data.
ERRNO	ERRNO	This returned value indicates error status. See “ERRNO” in Section 2.
<result>	Integer result	A function-specific result is returned. See “Result (Integer or Real)” in Section 2.

POSIX_WAITPID

The POSIX_WAITPID procedure suspends the execution of the calling process until a specified child process returns status information.

Supported Functions

POSIX_WAITPID provides functions equivalent to the C language wait( ) and waitpid( ) functions. The following conditions force a function equivalent to wait( ):

- PID parameter = -1
- OPTION parameter = 0

Procedure Declaration

You declare this procedure as follows:

```
LIBRARY MCPSUPPORT (LIBACCESS = BYFUNCTION);
INTEGER PROCEDURE POSIX_WAITPID (SELECTOR, PID, STAT_LOC, OPTION,
                                 STRUCT, STRUCT_OFF, STRUCT_MAX, ERRNO;
  VALUE        SELECTOR, PID, OPTION, STRUCT_OFF, STRUCT_MAX;
  REFERENCE    STRUCT, STAT_LOC, ERRNO;
  REAL ARRAY   STRUCT [0];
  INTEGER      SELECTOR, PID, STAT_LOC, OPTION, STRUCT_OFF, STRUCT_MAX,
               ERRNO;
LIBRARY MCPSUPPORT;
```

Parameter Summary

Parameter	Rule	Description
SELECTOR	Call-by-value integer	This input parameter must contain a value of 1.
PID	Call-by-value integer	This input parameter indicates the child process for which status is to be received. See "PID parameter (WAITPID)" in Section 2 for further information.
STAT_LOC	Call-by-reference integer	This output parameter receives the termination status for the specified process. See "STATUS parameter" in Section 2 for an analysis of the returned termination status.
OPTION	Call-by-value integer	This input parameter defines function options. See "Option parameter (WAITPID)" in Section 2 for further information.
STRUCT, STRUCT_OFF, STRUCT_MAX	Structure array output	Not used.
ERRNO	ERRNO	This returned value indicates error status. See "ERRNO" in Section 2.
<result>	Integer result	A function-specific result is returned. See "Result (Integer or Real)" in Section 2.

Section 6
Programming Examples

About this Section

This section contains sample programs that illustrate the use of POSIX interface based functions in ALGOL programs. Additional examples will be provided in future releases.

Example 1 illustrates a simple ALGOL program that uses many of the functions described in Section 3. Note that this program includes the SYMBOL/POSIX/ALGOL/PROPERTIES file. Briefly, this program:

- Executes a DIRSETUP procedure. This procedure:
  - Displays the starting (old) current working directory, changes this directory to "tempdir", and then displays the new working directory. The GETCWD and CHDIR functions are used.
  - Determines if the process has read permission for an existing file ("tempfile"). If not, the program sets read/write/execute permissions for all users of that file. The ACCESS and CHMOD functions are used.
- Executes the SIGSETUP procedure. This procedure:
  - Creates a signal environment for this process. The SIGPUSH function is used.
  - Defines a signal-catching function for the SIGALRM signal. The SIGACTION function is used.
  - Pauses for a period of 10 seconds. The ALARM and PAUSE functions are used.
  - Suspends execution for 100 seconds. The PAUSE function is used.
- Uses the SEMGET function to create a semaphore set containing 2 semaphores; read and alter access permissions are set for all users.
- Uses the SEMCTL function to set the semaphore's SEMVAL value to 1.
- Uses the SETSID function to establish the calling process as a process group and session leader.
- Creates a child process with the FORK function.

- The child process performs the following operations:
  - It executes the CHILD_INFO and PERSONAL_INFO procedures to print a variety of information (user, group, and process group IDs) common to both the parent and child process. Various GET... functions provide this information.
  - It prints its process ID (obtained with the GETPID function).
- The parent process performs the following operations:
  - It executes the PARENT_INFO and PERSONAL_INFO procedures to print a variety of information (user, group, and process group IDs) common to both the parent and child process. Various GET... functions provide this information.
  - It prints its process ID (obtained with the GETPID function).
  - It suspends execution with the WAITP function.

Example 2 is a C language program that provides functions equivalent to those in Example 1.

```
BEGIN
$INCLUDE "SYMBOL/POSIX/ALGOL/PROPERTIES."

  FILE RMT (KIND=REMOTE, UNITS=CHARACTERS, MAXRECSIZE=80);
  ARRAY MSG [0:20]; POINTER MSGP;  REAL MSGLN;
  EBCDIC ARRAY EMSG [0:99];

  DEFINE
    SEND_MSG (L, P) = BEGIN
                      REPLACE POINTER (P) + L BY 0 FOR 1;
                      IF MYSELF.SW1
                        THEN
                          DISPLAY (P)
                        ELSE
                          WRITE (RMT, L, P);
                      END #,
    MSG_INIT      =  MSGP := POINTER (MSG) #,
    PTXT (X)      =  REPLACE MSGP:MSGP BY X  #,
    PNUM (X)      =  BEGIN
                     IF (X) < 0 THEN PTXT ("-");
                     PTXT ((X) FOR * DIGITS);
                     END #,
    PSTR (X)      =  PTXT ((X) UNTIL = 0) #,
    PERR (N)      =  BEGIN
                     REPLACE EMSG BY 0 FOR 100;
                     STRERROR(EMSG,0,100,N);
                     PTXT("@LINE "); PTXT(LINENUMBER FOR 8 DIGITS);
                     PTXT(" ERROR= "); PNUM(N);
                     PTXT(" "); PSTR(EMSG);
                     FLUSH;
                     N := 0;
                     END #,
    FLUSH         =  BEGIN
                     SEND_MSG (OFFSET (MSGP), MSG);
                     MSGP := POINTER (MSG);
                     END #;

  DEFINE GET_CONTROL = SEM_OPERATION(-1) #,
         REL_CONTROL = SEM_OPERATION(1)  #;

  DEFINE PATH_MAX = 256 #,
         NAME_MAX = 3  #,
         NGROUPS_MAX = 16 #;

  INTEGER GLOB, FLAG, SEMID;
  INTEGER I, ERR, STATUS, VAR, PID;
```

Example 6–1. Using POSIX Functions in an ALGOL Program (cont.)

```
LABEL XIT;
EBCDIC ARRAY STR[0:99];
REAL ARRAY SOPS[0:2], ARG[0:SEMID_DS_SIZE-1];

PROCEDURE PRINTGROUP(GROUPID);                   % PRINTGROUP procedure
INTEGER GROUPID;
BEGIN
   ARRAY GR [0:99];
   INTEGER ERR;

   PNUM(GROUPID);PTXT("(");
   IF GETGRGID(GROUPID, GR, 0, 100, ERR) < 0 THEN
      BEGIN
      PTXT("?????)");FLUSH;
      STRERROR(STR, 0, 100, ERR);
      PTXT("GETGRGID ERROR: ");PSTR(STR);FLUSH;
      END
   ELSE
      BEGIN
      PSTR(POINTER(GR[GR_NAME]));PTXT(")");FLUSH;
      END;

END OF PRINTGROUP;

PROCEDURE PRINTUSER(USERID);                     % PRINTUSER procedure
INTEGER USERID;
BEGIN     ARRAY PW [0:PW_SIZE-1];
   INTEGER ERR;

   PNUM(USERID);PTXT("(");
   IF GETPWUID(USERID, PW, 0, PW_SIZE, ERR) < 0 THEN
      BEGIN
      PTXT("?????)");FLUSH;
      STRERROR(STR, 0, 100, ERR);
      PTXT("GETPWUID ERROR: ");PSTR(STR);FLUSH;
      END
   ELSE
      BEGIN
      PSTR(POINTER(PW[PW_NAME]));PTXT(")");FLUSH;
      END;

END OF PRINTUSER;

PROCEDURE PRINTALLGROUPS;                     % PRINTALLGROUPS procedure
BEGIN
   INTEGER NGROUPS, I, GID, ERR;
   INTEGER ARRAY LISTX[0:NGROUPS_MAX*NAME_MAX-1];
   LABEL XIT;
```

Example 6–1. Using POSIX Functions in an ALGOL Program (cont.)

```
   NGROUPS := GETGROUPS(LISTX, 0, 0, ERR);
   IF NGROUPS < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("GETGROUPS ERROR: ");PSTR(STR);FLUSH;
      GO XIT;
      END
   ELSE IF NGROUPS = 0 THEN
      BEGIN
      PTXT("NO SUPPLEMENTARY GROUPS ARE AVAILABLE");FLUSH;
      GO XIT;
      END;

   IF GETGROUPS(LISTX, 0, NGROUPS_MAX*NAME_MAX, ERR) < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("GETGROUPS ERROR: ");PSTR(STR);FLUSH;
      GO XIT;
      END;

   PTXT("THE FOLLOWING SUPPLEMENTARY GROUPS ARE AVAILABLE");FLUSH;
   FOR I := 0 STEP 1 UNTIL NGROUPS-1 DO
      BEGIN
      PTXT(" ");PNUM(I);PTXT(" ");PNUM(LISTX[I]);FLUSH;
      END;

XIT:

END OF PRINTALLGROUPS;

PROCEDURE PERSONAL_INFO;                         % PERSONAL_INFO procedure
BEGIN
   INTEGER UID, GID, ERR;
   EBCDIC ARRAY BUF[0:99];

   IF GETLOGIN(BUF, 0, 100, ERR) < 0 THEN
      BEGIN
      PTXT("LOGIN NAME IS NOT KNOWN");FLUSH;
      END
   ELSE
      BEGIN
      PTXT("LOGIN NAME IS ");PSTR(BUF);FLUSH;
      END;

   PRINTUSER(GETUID(ERR));
   PRINTUSER(GETEUID(ERR));
   PRINTGROUP(GETGID(ERR));
```

Example 6–1. Using POSIX Functions in an ALGOL Program (cont.)

```
   PRINTGROUP(GETEGID(ERR));
   PRINTALLGROUPS;
END OF PERSONAL_INFO;

PROCEDURE PARENT_INFO;                          % PARENT_INFO procedure
BEGIN
   INTEGER ERR;

   PTXT("<<<< PARENT PROCESS >>>>");FLUSH;
   PTXT("REAL USER ID FOR PARENT ");PNUM(GETUID(ERR));FLUSH;
   PTXT("REAL GROUP ID FOR PARENT ");PNUM(GETGID(ERR));FLUSH;
   PTXT("EFFECTIVE USER ID FOR PARENT ");PNUM(GETEUID(ERR));FLUSH;
   PTXT("EFFECTIVE GROUP ID FOR PARENT ");PNUM(GETEGID(ERR));FLUSH;
   PTXT("PROCESS GROUP ID FOR PARENT ");PNUM(GETPGRP(ERR));FLUSH;
   PERSONAL_INFO;

END OF PARENT_INFO;

PROCEDURE CHILD_INFO;                            % CHILD_INFO procedure
BEGIN
   INTEGER ERR;

   PTXT("<<<< CHILD PROCESS >>>>");FLUSH;
   PTXT("REAL USER ID FOR CHILD ");PNUM(GETUID(ERR));FLUSH;
   PTXT("REAL GROUP ID FOR CHILD ");PNUM(GETGID(ERR));FLUSH;
   PTXT("EFFECTIVE USER ID FOR CHILD ");PNUM(GETEUID(ERR));FLUSH;
   PTXT("EFFECTIVE GROUP ID FOR CHILD ");PNUM(GETEGID(ERR));FLUSH;
   PTXT("PROCESS GROUP ID FOR CHILD ");PNUM(GETPGRP(ERR));FLUSH;
   PERSONAL_INFO;

END OF CHILD_INFO;

INTEGER PROCEDURE DIRSETUP;                        % DIRSETUP procedure
BEGIN
   EBCDIC ARRAY BUF[0:PATH_MAX-1];
   EBCDIC ARRAY NEWDIR[0:255], NEWFILE[0:255];
   INTEGER ERR;
   LABEL XIT;

   IF GETCWD(BUF, 0, PATH_MAX, ERR) < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("GETCWD ERROR: ");PSTR(STR);FLUSH;
      DIRSETUP := -1;
      GO XIT;
      END;
```

Example 6–1. Using POSIX Functions in an ALGOL Program (cont.)

```
      PTXT("OLD WORKING DIRECTORY IS ");PSTR(BUF);FLUSH;
      REPLACE NEWDIR BY "tempdir", 0;
      IF CHDIR(NEWDIR, 0, PATH_MAX, PATH_TYPE_PATHNAME,
                                  SEARCHRULE_POSIX, ERR) < 0 THEN
         BEGIN
         STRERROR(STR, 0, 100, ERR);
         PTXT("CHDIR ERROR: ");PSTR(STR);FLUSH;
         DIRSETUP := -1;
         GO XIT;
         END;

      IF GETCWD(BUF, 0, PATH_MAX, ERR) < 0 THEN
         BEGIN
         STRERROR(STR, 0, 100, ERR);
         PTXT("GETCWD ERROR: ");PSTR(STR);FLUSH;
         DIRSETUP := -1;
         GO XIT;
         END;

      PTXT("NEW WORKING DIRECTORY IS ");PSTR(BUF);FLUSH;
      REPLACE NEWFILE BY "tempfile", 0;

      IF ACCESS(NEWFILE, 0, PATH_MAX, PATH_TYPE_PATHNAME,
              SEARCHRULE_POSIX, R_OK, ERR) < 0 THEN
         BEGIN
         IF CHMOD(NEWFILE, 0, PATH_MAX, PATH_TYPE_PATHNAME,
                 SEARCHRULE_POSIX, 1"111111111", ERR) < 0 THEN
            BEGIN
            STRERROR(STR, 0, 100, ERR);
            PTXT("CHMOD ERROR: ");PSTR(STR);FLUSH;
            DIRSETUP := -1;
            END;
         END;

   XIT:

   END OF DIRSETUP;

 % SIGNAL HANDLING PROCEDURE.                         % SIGHANDLER procedure
 INTEGER PROCEDURE SIG_HANDLER (SIG,INFO1,INFO2,INFO3,INFO4,INFO5,
           INFO6,INFO7,INFO8,INFO9);
   VALUE SIG,INFO1,INFO2,INFO3,INFO4,INFO5,INFO6,INFO7,INFO8,
           INFO9;
   INTEGER SIG,INFO1,INFO2,INFO3,INFO4,INFO5,INFO6,INFO7,INFO8,
           INFO9;
   BEGIN
```

Example 6–1. Using POSIX Functions in an ALGOL Program (cont.)

```
   PTXT("INTERCEPTED A SIGALRM SIGNAL");FLUSH;
   FLAG := 0;
END OF SIG_HANDLER;

PROCEDURE SIGSETUP;                                   % SIGSETUP procedure
BEGIN
   INTEGER R, ERR;
   ARRAY ACT[0:2], OACT[0:2], NULL[0:0];
   LABEL XIT;

   % FIRST DO A SIGPUSH TO MAKE THE STACK SIGNAL CAPABLE
   R := SIGPUSH(ERR);
   IF R < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("SIGPUSH ERROR: ");PSTR(STR);FLUSH;
      GO XIT;
      END;
   % ESTABLISH THE SIGNAL CATCHING FUNCTION
   % ASSOCIATE SIGALRM TO THE SIG_HANDLER PROCEDURE
   R := SIGACTION(SIGALRM, SIG_HANDLER, ACT, 0, 3, NULL, 0, 0, ERR);
   IF R < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("SIGACTION ERROR: ");PSTR(STR);FLUSH;
      GO XIT;
      END;

   % TEST SIGNAL CATCHING FUNCTION
   FLAG := 1;
   IF ALARM(10, ERR) < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("ALARM ERROR: ");PSTR(STR);FLUSH;
      GO XIT;
      END;
```

Example 6–1. Using POSIX Functions in an ALGOL Program (cont.)

```
   PAUSE(ERR);
   PTXT("DONE WITH PAUSE");FLUSH;
   FLAG := 1;
   IF ALARM(10, ERR) < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("ALARM ERROR: ");PSTR(STR);FLUSH;
      GO XIT;
      END;

   SLEEP(100, ERR);
   PTXT("DONE WITH SLEEP");FLUSH;

XIT:

END OF SIGSETUP;

INTEGER PROCEDURE SEM_OPERATION(OP);            % SEM_OPERATION procedure
INTEGER OP;
BEGIN
   INTEGER R;

   SOPS[0] := 0; % SEMAPHORE NUMBER IN A SET
   SOPS[1] := OP; % OPERATION TO BE PERFORMED
   SOPS[2] := 0; % MODIFIER FLAG FOR THE OPERATION
   R := SEMOP(SEMID, SOPS, 0, 3, 1, ERR);
   IF R < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("SEMOP ERROR: ");PSTR(STR);FLUSH;
      SEM_OPERATION := -1;
      END;

END OF SEM_OPERATION;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  START OF THE TEST PROGRAM       %                     % Main program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MSG_INIT;
FLUSH;

I := LINKLIBRARY(MCPSUPPORT);
PTXT("RESULT OF LINKLIBRARY IS ");PNUM(I);
FLUSH;

DIRSETUP; % SET UP CURRENT DIRECTORY
SIGSETUP; % SET UP SIGNAL CATCHING FUNCTION
```

Example 6–1. Using POSIX Functions in an ALGOL Program (cont.)

```
   % CREATE A SEMAPHORE
   SEMID := SEMGET(1000, 2, 1"111111111" & 1 IPC_CREAT, ERR);
   IF SEMID < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("SEMGET ERROR: ");PSTR(STR);FLUSH;
      GO XIT;
      END;

   ARG[0] := ARG[1] := 1;
   SEMCTL(SEMID, 2, SEM_SETALL, 0, ARG, 0, SEMID_DS_SIZE, ERR);

   VAR := 88;
   SETSID(ERR); % ESTABLISH THE CALLING PROCESS AS SESSION AND
                % PROCESS GROUP LEADER

   PID := FORK(ERR);
   IF PID < 0 THEN
      BEGIN
      STRERROR(STR, 0, 100, ERR);
      PTXT("FORK ERROR: ");PSTR(STR);FLUSH;
      GO XIT;
      END
   ELSE IF PID = 0 THEN % CHILD PROCESS
      BEGIN
      GET_CONTROL; % GET CONTROL
      CHILD_INFO;  % PRINT CHILD INFORMATION
      REL_CONTROL; % RELEASE CONTROL
      GLOB := GLOB + 1;
      VAR := VAR + 1;
      PTXT("CHILD PID=");PNUM(GETPID(ERR));PTXT(" GLOB=");PNUM(GLOB);
      PTXT(" VAR=");PNUM(VAR);FLUSH;
      GO XIT;
      END;

   % PARENT PROCESS
   GET_CONTROL; % OBTAIN CONTROL
   PARENT_INFO; % PRINT USER RELATED INFORMATION
   REL_CONTROL; % RELEASE CONTROL

   PTXT("PARENT PID=");PNUM(GETPID(ERR));PTXT(" GLOB=");PNUM(GLOB);
   PTXT(" VAR=");PNUM(VAR);FLUSH;

   WAITP(STATUS, ERR);

XIT:

END.
```

Example 6–1. Using POSIX Functions in an ALGOL Program

```
#include <sys/types.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdio.h>
#include <signal.h>
#include <limits.h>
#include <grp.h>
#include <pwd.h>

#define get_control sem_operation(-1)
#define rel_control sem_operation(1)

#define TRUE 1
#define FALSE 0

void sigsetup();
void parent_info();
void child_info();
void ding(int);
void sem_operation();
void personal_info();
void printallgroups();
void printuser(uid_t);
void printgroup(gid_t);
void dirsetup();

int glob = 6;
short semary[2];
int flag, semid;
struct sembuf psembuf;

int main(int argc, char *argv[])
{
        int status, var;
        pid_t pid;

        dirsetup(); /* set up current directory */
        sigsetup(); /* set up signal catching */

        /* test signal catching function */
        flag = TRUE;
        (void)alarm(10);
        pause();
        (void)printf("Done with pause\n");
        flag = TRUE;
```

Example 6–2. An Equivalent C Program (cont.)

```
        (void)alarm(10);
        sleep(100);
        (void)printf("Done with sleep\n");

        /* create a semaphore */
        semid = semget(1000, 2, 0777|IPC_CREAT);
        if(semid < 0) {
                perror("semget failed");
                exit(-1);
        }
        semary[0]=semary[1]=1;
        semctl(semid, 2, SETALL, semary);

        var=88;
        setsid(); /* establish the calling process as session and
                     process group leader */

        if((pid=fork()) < 0) {
                perror("fork error");
                exit(-1);
        }
        else if(pid ==  0) {   /* child */
                get_control; /* wait for control */
                child_info();
                rel_control; /* release control */
                glob++;
                var++;
                printf("Child pid = %d, glob = %d, var = %d\n",
                        getpid(), glob, var);
                printf("end of child\n");
                exit(0);
        }
        /* parent */
        get_control;  /* got control? */
        parent_info(); /* print user related information */
        /* wake up child process */
        rel_control; /* release control */
        printf("Parent pid = %d, glob = %d, var = %d\n",
                getpid(), glob, var);
        printf("end of parent\n");
        wait(&status);

        exit(0);
}
void dirsetup()
{
```

Example 6–2. An Equivalent C Program (cont.)

```
char buf[PATH_MAX];
char newdir[] = "tempdir";

char newfile[] = "tempfile";

        if(getcwd(buf,PATH_MAX) == NULL) {
                perror("getcwd failed");
                exit(-1);
        }
        printf("Old Working Directory is %s\n",buf);
        if(chdir(newdir) < 0) {
                perror("chdir failed");
                exit(-1);
        }
        if(getcwd(buf,PATH_MAX) == NULL) {
                perror("getcwd failed");
                exit(-1);
        }
        printf("Current Working Directory is %s\n",buf);
        if(access(newfile, R_OK|W_OK|X_OK) < 0) {
                if(chmod(newfile,0777) < 0) {
                        perror("chmod failed");
                        exit(-1);
                }
        }
}

void parent_info()
{
        printf("<<<< PARENT PROCESS >>>>\n");
        printf("real user id for parent = %d\n", getuid());
        printf("real group id for parent = %d\n", getgid());
        printf("effective id for parent = %d\n", geteuid());
        printf("effective  group id for parent = %d\n", getegid());
        printf("process group id for parent = %d\n", getpgrp());
        personal_info();
}

void child_info()
{
        printf("<<<< CHILD PROCESS >>>>\n");
        printf("real user id for child = %d\n", getuid());
        printf("real group id for child = %d\n", getgid());
        printf("effective id for child = %d\n", geteuid());
        printf("effective  group id for child = %d\n", getegid());
        printf("process group id for child = %d\n", getpgrp());
        personal_info();
}
```

Example 6–2. An Equivalent C Program (cont.)

```
void sem_operation(int op)
{
        psembuf.sem_op = op;
#if 0
        psembuf.sem_flg = SEM_UNDO;
#endif
        psembuf.sem_num = 0;
        semop(semid, &psembuf, 1);
}

void ding(int sig)
{
        printf("In signal handling routine!\n");
        flag = FALSE;
        return;
}

void sigsetup()
{
int i;
struct sigaction act,oact;

        /* ignore interrupt from the controlling terminal */
        act.sa_handler = SIG_IGN;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        if(sigaction(SIGINT,&act,&oact) != 0) {
                perror("sigaction failed");
                exit(-1);
        }
        /* establish the signal catching function */
        act.sa_handler = ding;
        if(sigaction(SIGALRM,&act,&oact) != 0) {
                perror("sigaction failed");
                exit(-1);
        }
}

/* print out the group number in decimal followed by
   groupname. */

void printgroup(gid_t groupid)
{
unsigned long lt;

struct group *grpptr;
        lt = (unsigned long)groupid;
        (void)printf(" %lu(",lt);
```

Example 6–2. An Equivalent C Program (cont.)

```
        grpptr = getgrgid(groupid);
        if(grpptr == NULL)
        {
                (void)printf("?????)");
                return;
        }
        (void)printf("%s)",grpptr->gr_name);
        return;
}

/* print out the user id in decimal followed by (username) */
void printuser(uid_t userid)
{
unsigned long lt;
struct passwd *pwptr;
        lt = (unsigned long)userid;
        (void)printf(" %lt(",lt);
        pwptr = getpwuid(userid);
        if(pwptr == NULL)
        {
                (void)printf("?????)");
                return;
        }
        (void)printf("%s)",pwptr->pw_name);
        return;
}

void printallgroups()
{
int ngroups;
gid_t *grpptr;

int i;
gid_t gid;

#ifndef NGROUPS_MAX
#define NGROUPS_MAX 0
#endif

#if NGROUPS_MAX < 1
        (void)printf("Supplementary group IDs are "
                        "not supported\n");
#else
        ngroups = getgroups(0,(gid_t *)NULL);
        if(ngroups == -1)
        {
```

Example 6–2. An Equivalent C Program (cont.)

```
                (void)perror("getgroups() failed");
                return;
        }
        if(ngroups == 0)
        {
                (void)printf("No supplementary groups are "
                                "available\n");
                return;
        }
        grpptr = calloc(ngroups,sizeof(gid_t));
        if(getgroups(ngroups,grpptr) == -1)
        {
                (void)perror("getgroups() failed");
                return;
        }
        (void)printf("The following supplementary groups are "
                        "available\n");
        for(i=1; i <= ngroups; i++)
        {
                gid = *grpptr++;
                (void)printf("\t");
                printgroup(gid);
                (void)printf("\n");
        }
#endif
        return;
}

void personal_info()
{
uid_t uid;
gid_t gid;
char *login;

        login = getlogin();
        if(login == NULL)
                (void)printf("Login name is not known\n");
        else
                (void)printf("Login name is '%s'\n",login);
        printuser(getuid());
        printuser(geteuid());
        printgroup(getgid());
        printgroup(getegid());
        (void)printf("\n");
        printallgroups();
}
```

Example 6–2. An Equivalent C Program

Glossary

A

absolute pathname
In the POSIX interface, a pathname that begins with a slash character (/). The absolute pathname locates a file or a directory starting at the file system root.

array
An ordered collection of a fixed number of common elements under one name, each element having the same data type. Access for each element is through an index to the common name.

B

blocked signal
A type of signal that the operating system is not delivering to a receiving process as a result of a request by that receiving process. The operating system indefinitely postpones delivery of blocked signals; they remain pending until the process conditions change.

by reference
Pertaining to one method of passing a parameter to a procedure. The system evaluates the location of the actual parameter and replaces the formal parameter with a reference to that location. Any change made to the formal parameter affects the actual parameter, and vice versa. *Synonym for* call-by-reference.

by value
Pertaining to one method of passing a parameter to a procedure. A copy of the value of the actual parameter is assigned to the formal parameter, which is thereafter handled as a variable that is local to the procedure body. Any change made to the value of a by-value formal parameter has no effect outside the procedure body. *Synonym for* call-by-value.

byte-file
The type of disk file normally created and accessed by a strictly conforming POSIX.1 application. Many traditional system applications do not process byte-files. The following file attributes define a byte-file: FILESTRUCTURE=STREAM, FRAMESIZE=8, and MAXRECSIZE=1. *See also* record-file.

C

call-by-reference
Pertaining to one method of passing a parameter to a procedure. The system evaluates the location of the actual parameter and replaces the formal parameter with a reference to that location. Any change made to the formal parameter affects the actual parameter, and vice versa. *Synonym for* by reference.

call-by-value
Pertaining to one method of passing a parameter to a procedure. A copy of the value of the actual parameter is assigned to the formal parameter, which is thereafter handled as a variable that is local to the procedure body. Any change made to the value of a call-by-value formal parameter has no effect outside the procedure body. *Synonym for* by value.

catch a signal
To call a signal-catching function by a process after it is interrupted by delivery of a specific signal type. *See also* signal catcher.

child process
In the POSIX interface, a process created with the fork function. It starts as a copy of the calling (parent) process, but it has its own unique process ID.

code segment dictionary
A memory structure that is associated with a process and that indexes the memory addresses of the various segments of program code used by that process. The same code segment dictionary can be shared by more than one process, provided that each process is an instance of the same procedure. A code segment dictionary is also referred to as a D1 stack.

compiler control option
An individual compiler directive that appears in a compiler control record (CCR). Compiler control options were previously referred to as compiler dollar options or dollar options.

current working directory
See working directory.

D

dot
In the POSIX interface, a filename node that consists solely of a single dot character (.). When part of a pathname, such a node refers to the preceding pathname component.

dot-dot
In the POSIX interface, a filename node that consists of two dot characters (..). When part of a pathname, such a node refers to the parent of the preceding pathname component.

D1 stack
See code segment dictionary.

D2 stack (D{2} stack)

(1) A stack initiated for each executing program that is used for the storage of items allocated at lexical level 2. The D2 stack is also referred to as the working stack. (2) In the transaction processing system (TPS), data and procedures that are global to a particular transaction base reside in the D2 stack of the transaction library, which is also referred to as the *<transaction base name>/CODE/HOSTLIB stack.*

E

EBCDIC

Extended Binary Coded Decimal Interchange Code. An 8-bit code representing 256 graphic and control characters that are the native character set of most mainframe systems.

EBCDIC array

In ALGOL, an array whose elements are EBCDIC characters.

effective group ID

In the POSIX interface, the group ID currently in effect for a process. The effective group ID is used to validate file access, establish ownership, and check permissions. This value is subject to change over the lifetime of the process. It is a numeric value that corresponds to the GROUPCODE task attribute. *See also* group ID, real group ID, GROUPCODE.

effective user ID

In the POSIX interface, the user ID currently in effect for a process. The effective user ID is used to validate file access, establish ownership, and check permissions. This value is subject to change over the lifetime of the process. It is a numeric value that corresponds to the USERCODE task attribute. *See also* user ID, real user ID, USERCODE.

entry point

A procedure or function that is a library object.

environment

(1) In the Editor, the set of conditions in the area of the object code in which a particular line of a program is found. This information is stored in the cross-reference files. (2) In the POSIX interface, an array of string variables of the form "*name=value*" that specify various operating characteristics of a process. Individual environment values can be set and accessed by a program. In addition, a new environment can be established when a program is executed.

environment variables

(1) In a workstation environment, such as OS/2 , names that specify global values. LIB and INCLUDE are examples of environment variables. (2) In the POSIX interface, any string variable contained in the environment.

ERRNO

In the POSIX interface, an external variable for returning error identification information to the program.

F

family name

(1) The name, consisting of up to 17 alphanumeric characters, assigned by an installation to identify a family of disks. (2) The name (label) of the disk or disk pack on which a physical file is located. The family name of a file is determined by the value of the FAMILYNAME file attribute. (3) The name of the logical group of disk packs on which a physical file is located. A family name consists of from 1 to 17 alphanumeric characters and is assigned by the installation.

FIB

See file information block.

FIFO special file

In the POSIX interface, a file with the property that data written into it is read on a first-in-first-out basis. In practice, a FIFO special file is similar to a pipe. However, unrelated processes can exchange data through a FIFO special file. In addition, a FIFO filename exists in the system file hierarchy. *See also* pipe.

file description

See open file description.

file descriptor

In the POSIX interface, a per-process unique, non-negative integer used to identify an open file for the purpose of file access.

file group class

In the POSIX interface, the property of a file indicating access permissions for a process related to the process's group identification. A process is in the file group class if it is not in the file owner class and if its effective group ID or one of its supplementary group IDs matches the group ID associated with the file.

file mode

In the POSIX interface, a word containing file permission bits and other characteristics of a file. File mode is specified in the SECURITYMODE file attribute.

file name

(1) A name or word that designates a set of data items. (2) A unique identifier for a file, consisting of name constants separated by slashes. Each name constant consists of letters, digits, and selected special characters. A file name can be optionally preceded by an asterisk (*) or usercode, and optionally followed by ON and a family name. (3) In RPG, a name that designates a set of data items. (4) In COBOL, a user-defined word that names a file described in a file description entry or a sort-merge file description entry within the FILE SECTION of the DATA DIVISION. (5) In the POSIX interface, a name node within a pathname.

file offset

In the POSIX interface, the byte position in the file where the next I/O operation begins.

file other class

In the POSIX interface, the property of a file indicating access permissions for a process related to the process's user and group identification. A process is in the file other class if it is not in the file owner class or the file group class.

file owner class

In the POSIX interface, the property of a file indicating access permissions for a process related to the process's user identification. A process is in the file owner class if its effective user ID matches the user ID of the file.

file permission bits

In the POSIX interface, information about a file that is used (along with other information) to determine if a process has read, write, or execute/search permission for that file. File permission bits are provided in the file mode. These bits are divided into three parts: owner, group, and other. Each part is used with the corresponding file class of processes. *See also* file group class, file mode, file other class, and file owner class.

file title

The complete identifier for a file that consists of the file name, and, for disk files, the word ON, and the family name.

G

generate a signal

To recognize a signal event and create an appropriate signal.

GID

See group ID.

group

(1) A collection of devices, such as processors, memory modules, and I/O devices, under the control of a single master control program (MCP). A group is referred to as a partition. (2) A collection of related data items that can be viewed as a single data item. A group can also refer to a collection of groups. (3) In the POSIX interface, an association of users who share a specific group ID as the identifier associated with their GROUPCODE or one of their SUPPLEMENTARYGRPs.

group class

See file group class.

group ID (GID)

In the POSIX interface, a unique number corresponding to the GROUPCODE task attribute. The operating system associates the group ID value with an instance of a user group. A value of 1 indicates that no group is assigned. *See also*, group, effective group ID, real group ID.

guard file

A disk file created by the GUARDFILE utility program that describes the access rights of various users and programs to a program, data file, or database.

I

I/O

Input/output. An operation in which the system reads data from or writes data to a file on a peripheral device such as a disk drive.

include file

An external file that is included as part of a compilation by writing the INCLUDE preprocessor directive as part of the source text.

integer

(1) A whole number. (2) In COBOL, a numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point.

intrinsic

A system-supplied program routine for common mathematical and other operations that is loaded onto the system separately. An intrinsic can be invoked by the operating system or user programs.

L

library

(1) A collection of objects grouped together to be exported to another process, imported from another process, or both. There are three types of libraries: client libraries, server libraries, and connection libraries. (2) *Synonym for* server library.(3) (VDP) A collection of related files.

library directory

A library template associated with a server library.

library object

An object that is exported by a server library or connection library and imported by a client library or connection library.

library process

An instance of the execution of a server library program or connection library program.

M

master control program (MCP)

The central program of the enterprise server operating system.

MCP

See master control program.

mix

The set of processes that currently exist on a particular computer. The mix can include active, scheduled, and suspended processes.

mix number

A 4-digit number that identifies a process while it is executing. This number is stored in the MIXNUMBER task attribute.

N

named semaphore

In the POSIX interface, a semaphore that a process references by name (character string). Note that only POSIX.4-defined semaphores can be referenced in this way; X/Open-defined semaphores must be referenced by identifier. *See* semaphore, unnamed semaphore.

null character

A character whose binary value is zero.

null string

An empty or zero-length string.

O

open file

In the POSIX interface, a file that is currently associated with a file descriptor.

open file description

In the POSIX interface, a record of how a process or group of processes is accessing a file. Open file description information includes file offset, file status, and file access modes.

other class

See file other class.

owner class

Synonym for file owner class.

owner of a file

A file owner is normally the creator of that file. Typically, the owner has certain privileges (such as deletion rights) that are not available to other users. The usercode portion of the file title indicates the owner of most A Series files.

P

parent process

In the POSIX interface, the process that created a child process with the fork function. *See also* parent process ID.

parent process ID (PPID)

In the POSIX interface, the process ID of a process's parent process. When the parent process's lifetime is ended, the parent process ID is the process ID of a specified system process. *See also* parent process.

path

(1) The route that must be traced from a directory to a subdirectory, or through a series of subdirectories, to find a file. (2) In the I/O subsystem, a set of addresses that uniquely describes the data flow between the host and any peripheral device. (3) In Network Definition Language II (NDLII) and X.25, a route between two nodes. (4) In Data Management System II (DMSII), a specific location within the logical ordering of a data set, set, subset, or access. (5) In Extended Retrieval with Graphic Output (ERGO), an ordered list of data sets used in generating a report. (6) In the MS-DOS operating system, a specification of all the directories that must be searched to find a file.

pathname

In the POSIX interface, the ordered list of directory filenames that locates a directory or a file. A slash (/) is used to separate each filename from its predecessor. POSIX pathnames are case-sensitive; therefore, /home/adam is not the same as /home/ADAM.

PCW

See program control word.

pending signal

In the POSIX interface, a signal that has been generated but not yet delivered. *See* blocked signal.

PGID

See process group ID.

PIB

See program information block, process information block.

PID

See process ID.

pipe

(1) A connection between two processes through which the output of the first process becomes the input to the second process. (2) In the POSIX interface, a logical connection between processes that have a common ancestor. Pipes are half-duplex—data flows in one direction only. They are accessed by a pair of file descriptors created by the pipe( ) function. Pipes do not have a name in the POSIX file hierarchy. *See also* FIFO special file.

Portable Operating System Interface (POSIX)

One of a number of interfaces defined by an Institute of Electrical and Electronic Engineers (IEEE) standard. An individual interface is referred to as POSIX.*n*, where *n* is a numeric suffix derived from the standard (for example, POSIX.1 and POSIX.2).

POSIX

See Portable Operating System Interface.

POSIX.1

An abbreviation for the *Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API) [C Language]* standard (ISBN 1-55937-061-0). This standard is published by the Institute of Electrical and Electronics Engineers, Inc. (IEEE). It defines a portable interface between C language application programs and the operating system.

POSIX.2

An abbreviation for the *Portable Operating System Interface (POSIX) – Part 2: Shell and Utilities* standard (ISBN 1-55937-255-9). This standard is published by the Institute of Electrical and Electronics Engineers, Inc. (IEEE). It defines a shell command language and a set of system utilities that are largely based on the system services defined In the POSIX interface.1.

POSIX.4

An abbreviation for the *Part 1: System Application Program Interface (API) – Amendment 1: Real-time Extension [C Language]* of the POSIX family of standards. This amendment defines optional facilities such as semaphores, messages, and shared memory.

PPID

See parent process ID.

process

(1) The execution of a program or of a procedure that was initiated. The process has its own process stack and process information block (PIB). It also has a code segment dictionary, which can be shared with other processes that are executions of the same program or procedure. (2) A software application; that is, any activity or systematic sequence of operations that produces a specified result. (3) In the Advanced Data Dictionary System (ADDS), a structure that models a logical view of relationships between different parts of a system.

process group

In the POSIX interface, a collection of processes that permits the signaling of related processes.

process group ID (PGID)

In the POSIX interface, a unique positive integer that represents a process group during its lifetime.

process group leader

In the POSIX interface, a process whose process ID is the same as its process group ID.

process ID (PID)

In the POSIX interface, a unique positive integer the operating system associates with each process. This is equivalent to the process's MIXNUMBER task attribute.

process information block (PIB)

A memory structure that is associated with each process stack and code segment dictionary. The PIB contains control information that is visible only to the operating system. The PIB for a process stack also contains a reference to a task attribute block (TAB).

program control word (PCW)

(1) A word that is used to transmit processing information from a control program to the operational programs, or between operational programs. (2) A word containing the initial code-stream pointer and execution state values associated with an activation record in a program. A PCW is the means by which the execution state is established for an activation record when the activation record is created by procedure entry.

Q

queue

(1) A data structure used for storing objects; the objects are removed in the same order they are stored. (2) In Data Communications ALGOL (DCALGOL), a linked list of messages. (3) *See also* job queue, ready queue.

R

real group ID

In the POSIX interface, a process characteristic established when the process is created. The real group ID identifies the group associated with the user who created the process. The real group ID does not change for the lifetime of a process. *See also* group ID, effective group ID.

real number

Any number, including fractions and whole numbers.

real user ID

In the POSIX interface, a process characteristic established when the process is created. The real user ID identifies the user who created the process. The real user ID does not change for the lifetime of a process. *See* user ID, effective user ID.

record-file

A term used to describe the type of disk file normally created and accessed by CANDE, WFL, MARC, and traditional system applications. Conforming POSIX.1 applications only create a record-file if an explicit request is made. The following file attributes define a record-file: FILESTRUCTURE=ALIGNED180, FRAMESIZE=48. *See also* byte-file.

relative pathname

In the POSIX interface, a partial pathname used to locate a file or directory relative to the current working directory. The system concatenates the current working directory and the relative pathname to form an absolute pathname. *See also* absolute pathname, current working directory.

root

(1) The origin of all directories and files in a file system structure. (2) In the UNIX system, the user name for a superuser. (3) In the POSIX interface, the base directory of the file system. All other directories and files are located under the root directory and can be found by providing a full pathname from the root directory. The root directory is represented with the slash character (/).

S

saved set-group-ID

In the POSIX interface, a process characteristic that allows flexibility in assigning the effective group ID while executing certain code files. If the code file to be executed has its SETGROUPCODE flag set, saved set-group-ID is set to the effective group ID of the calling process when the code file is executed.

saved set-user-ID
In the POSIX interface, a process characteristic that allows flexibility in assigning the effective user ID while executing certain code files. If the code file to be executed has its SETUSERCODE flag set, saved set-user-ID is set to the effective user ID of the calling process when the code file is executed.

semaphores
(1) A method used by NetWare for A Series to synchronize the association of resources among both programs and processes. One use of semaphores is to provide a system of file sharing and file locking. (2) In the POSIX interface, a structure used to synchronize concurrent processes. Two types of semaphores can be used—those defined by X/Open and those defined by POSIX.4. *See* named semaphore, unnamed semaphore.

signal
In the POSIX interface, a mechanism by which a process can be notified of or affected by an event occurring in the system. Possible events include the expiration of a timer, a hardware fault, or a task termination request. The term signal also refers to the event itself.

signal catcher
A callable function that a process associates with a signal type. When the operating system delivers a signal of this signal type, it interrupts the receiving process and causes the process to execute the associated signal-catching function. *See also* catch a signal.

signal delivery
The operating system's action of creating a signal in response to a specific event.

signal generation
Pertaining to the operating system's action of creating a signal in response to a specific event.

signal mask
A set of signals the process wants to block if they occur.

signal type
In the POSIX interface, a signal characteristic that determines its meaning and how it is handled by the system.

stack
(1) A region of memory used to store data items in a particular order on a last-in, first-out basis. (2) A nonpreferred synonym for process stack.

stuffed indirect reference word (SIRW)
A special control word used by the CPU to reference a location in an addressing environment. The form of the reference is such that the SIRW always points to the same location, no matter what the state of the current addressing environment.

supplementary group ID
In the POSIX interface, a process characteristic that is used to determine file access permissions. A process may have up to 16 supplementary group IDs in addition to the effective group ID. These supplementary group IDs are set to the supplementary group IDs of the parent process when the process is created. *See* group ID.

system command

Any of a set of commands used to communicate with the operating system. System commands can be entered at an operator display terminal (ODT), in a Menu-Assisted Resource Control (MARC) session, or by way of the DCKEYIN function in a privileged Data Communications ALGOL (DCALGOL) program.

system library

A library that is part of the system software and is accorded special privileges by the operating system. Two examples of system libraries are GENERALSUPPORT and PRINTSUPPORT.

T

TAB

See task attribute block.

task

(1) A dependent process. (2) Any process, whether dependent or independent. *See also* process.

task attribute

Any of a number of items that describe and control various aspects of process execution such as the usercode, priority, and the default family specification. Task attributes can be assigned interactively through task equations, or programmatically through statements that use task variables.

task attribute block (TAB)

A memory structure that stores the values of task attributes associated with a given task variable. Before the Mark 3.9 release, this information was part of the process information block (PIB).

timestamp

An encoded, 48-bit numerical value for the time and date. Various timestamps are maintained by the system for each disk file. Timestamps note the time and date a file was created, last altered, and last accessed.

U

unnamed semaphore

In the POSIX interface, a semaphore that a process refers to only by an integer identifier. *See* semaphore, named semaphore.

user ID (UID)

In the POSIX interface, the unique number the operating system associates with a user who logs on to the system. *See also* effective user ID, real user ID, usercode.

usercode

An identification code used to establish user identity and control security, and to provide for segregation of files. Usercodes can be applied to every task, job, session, and file on the system. A valid usercode is identified by an entry in the USERDATAFILE. In the POSIX interface, the usercode is mapped to an equivalent effective user ID.

USERDATAFILE

A system database that defines valid usercodes and contains various data about each user (such as accesscodes, passwords, and chargecodes) and the population of users for a particular installation.

V

volume

The medium of a mass storage device such as a disk, disk pack, or tape reel. The term *volume* is not restricted to the volume library on a cataloging system or the volume directory on a system with tape volume security. For example, on the BTOS family of workstations, the hard disk is a volume, and each floppy disk is a volume. When a volume is initialized, it is assigned a volume name and an optional password.

W

working directory

In the POSIX interface, a directory associated with a process that is used in pathname resolution for pathnames that do not begin with a slash (/). Synonymous with current working directory.

X

X/Open

A UNIX-based common applications environment defined by the X/Open Company. X/Open includes functional descriptions that are not defined by POSIX.1. Some of these functions are considered extensions to POSIX.1.

Index