Data General

Customer Documentation

POSIX.1 Conformance Document for the DG/UX™ System

AViiON®
PRODUCT LINE

POSIX.1 Conformance Document for the DG/UX™ System

069-701078-02

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 069-701078

Revision 02, August 1991

NOTICE

POSIX.1 Conformance Document for the DG/UX™ System

069-701078-02

Revision History:	Effective with:
Original Release – November 1989	DG/UX Release 4.20
Revision 1 – May 1991	DG/UX Release 4.32
Revision 2 – August 1991	DG/UX Release 5.4

Preface

This document describes how the DG/UX™ system meets the requirements necessary for an operating system to conform to the standard defined in IEEE 1003.1-1988 (POSIX.1), approved on August 22, 1988.

This document fulfills the requirements for documentation on conformance indicated in Section 2.2.1.2 of the standard.

Who Should Read This Manual?

This document is for people who are evaluating operating systems for POSIX.1 conformance. The reader should be familiar with the IEEE 1003.1-1988 (POSIX.1) standard approved on August 22, 1988.

Document Organization

This document describes the behavior of the DG/UX implementation for all implementation-defined features identified in the POSIX.1 standard. The document specifies the behavior of the implementation in those sections of the standard where it is stated that implementations may vary. Section numbers in this document correspond to section numbers in the standard.

Related Documents

When using this document, you will need a DG/UX system documentation set for specific information on the DG/UX system. For more information on the POSIX.1 standard, you should refer to the *IEEE Portable Operating Systems Interface for Computer Environments, Std 1003.1-1988*. You can obtain this standard at the following address:

Publication Sales
IEEE Service Center
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08854 -1331
(201) 981-0060

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please use the enclosed TIPS order form (United States only) or contact your local Data General sales representative. A list of related documents appears at the end of this manual with the TIPS order form.

For a complete list of AViiON® and DG/UX™ manuals, see the *Guide to AViiON® and DG/UX™ System Documentation* (069-701085). The on-line version of this manual found in **/usr/release/doc_guide** contains the most current list.

Telephone Assistance

If you are unable to solve a problem using any manual you received with your system, free telephone assistance is available with your hardware warranty and with most Data General software service options. If you are within the United States or Canada, contact the Data General Service Center by calling 1-800-DG-HELPS. Lines are open from 8:00 a.m. to 5:00 p.m., your time, Monday through Friday. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

For telephone assistance outside the United States or Canada, ask your Data General sales representative for the appropriate telephone number.

Joining Our Users Group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive FOCUS monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual Member Directory, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1-800-877-4787 or 1-512-345-5316.

End of Preface

Contents

2. Definitions and General Requirements

2.2 Conformance

Release 5.4 of the DG/UX operating system conforms to IEEE Std. 1003.1-1988, also known as the IEEE Standard Portable Operating System Interface for Computer Environments (POSIX.1), with C Standard Language-Dependent System Support.

2.2.1.3 Conforming Implementation Options

2.2.1 Implementation Conformance

The following are the environment requirements for DG/UX 5.4 to conform to the standard:

The application should be compiled with the switches -ansi and -D_POSIX_SOURCE. (Other switches needed by the application may also be used.)

2.2.1.1 Requirements

Release 5.4 of the DG/UX operating system conforms to the POSIX.1 standard. Nonstandard extensions to the standard are identified as such in the DG/UX system documentation. No modifications to the default DG/UX system environment are required in order to run conforming POSIX.1 applications.

2.2.1.2 Documentation

This document is the Conformance Document for DG/UX 5.4 and meets all the requirements listed in section 2.2.1.2 of the POSIX.1 standard.

Release 5.4 of the DG/UX operating system conforms to IEEE Std. 1003.1-1988, also known as the IEEE Standard Portable Operating System Interface for Computer Environments, or simply as POSIX.1.

2.2.1.3 Conforming Implementation Options

The value of the {NGROUPS_MAX} option on DG/UX systems is 0. Processes that need to use supplementary groups must use the **sysconf**() function in order to determine how many such groups are available.

The {_POSIX_JOB_CONTROL} option is supported by the DG/UX system, but applications should use the **sysconf**() function to verify that it is available.

Processes needing to know whether or not the {_POSIX_CHOWN_RESTRICTED} option is supported for a particular file on the DG/UX system must use the **pathconf**() or **fpathconf**() system calls in order to find out.

2.2.3 Language-Dependent Services for the C Programming Language

The DG/UX system provides all the facilities referenced in Chapter 8 of the POSIX.1 standard.

2.2.3.1 Types of Conformance

The default C language compiler on DG/UX Systems conforms to the IEEE Std. 1003.1-1988 C Language Binding (C Standard Language-Dependent System Support).

2.2.3.2 C Standard Language-Dependent System Support

The default C language compiler on DG/UX systems conforms to draft X3J11/88-002 (13 May 1988) of the proposed ANSI C Programming Language Standard.

2.3 General Terms

The following terms are those whose meanings are partly or fully implementation-defined. This section completes their definitions for the DG/UX system implementation.

appropriate privileges: The only additional way to associate DG/UX system privileges with a process is for that process to have an effective user ID of 0 (the superuser). There are no DG/UX system privileges that cannot be obtained by the superuser.

character special file: A DG/UX system will include in its file system hierarchy some character special files that are not terminal device files. For information about such files, see chapter 7 of the *Programmer's Reference for the DG/UX™ System*.

file: The following additional file types are defined: socket and symbolic link. See the **stat**(2) manual page for details.

file group class: There are no additional ways for a DG/UX system process to belong to a file group class other than those described in the standard.

parent process ID: The original parent process ID of a process on the DG/UX system is the process ID of the process that created it via the **fork**() function. After the lifetime of the original parent process has ended, the child process's parent process ID is set to 1, which is the process ID of the special system initialization process.

pathname: A pathname that begins with two successive slashes is treated by the DG/UX system as if it began with a single slash.

read-only file system: Any file system that can be mounted by the DG/UX system can be mounted read-only if desired. When a file system is mounted read-only, none of the attributes of the file system may be altered, and none of the files on that file system may be written, deleted, created, or have any of their attributes (including the time of last access) altered. Operations that try to write or update any part of a read only file system or any of its files will fail with the error [EROFS].

2.4 General Concepts

The following terms are those whose meanings are partly or fully implementation-defined. This section completes their definitions for the DG/UX system implementation.

extended security controls: The DG/UX system provides no extended security control mechanisms beyond the access control mechanisms that are required by the standard.

file access permissions: The DG/UX system provides no additional or alternate file access control mechanisms beyond the file access control mechanism defined by the standard.

file times update: When file access times are marked for update, they are updated periodically, not necessarily immediately.

2.5 Error Numbers

In addition to the error numbers described in the standard, other error numbers are defined by the DG/UX system. See the file **<errno.h>** for the actual values of the error numbers.

On DG/UX systems, any attempt to use an invalid address as an argument to a system call will fail with the error [EFAULT]. However, attempts by library routines to access invalid addresses will result in segmentation violations.

On DG/UX systems, the maximum size of a file in bytes can be altered on a per-process basis by using the **ulimit**() or **setrlimit**() system calls. The initial ceiling for a process is 2147483647 bytes. Any process that tries to extend a file's size beyond that process's file size limit will fail with the error [EFBIG].

2.6 Primitive System Data Types

On DG/UX systems, the following POSIX.1 system data types are defined in the header **<sys/types.h>**:

```
typedef unsigned int  size_t;
typedef int           ssize_t;
typedef long          time_t;
typedef int           dev_t;
typedef long          off_t;
typedef unsigned long ino_t;
typedef int           uid_t;
typedef int           gid_t;
typedef unsigned long mode_t;
typedef unsigned long nlink_t;
typedef int           pid_t;
```

This file also contains other types that are not visible when only the _POSIX_SOURCE symbol is defined; they require that various other feature-test macros be defined in order to be visible. See **<sys/types.h>** for details.

2.7 Environment Description

Names of environment variables on DG/UX systems are not restricted to just the portable filename character set. Any 8-bit characters may be used, with the following exceptions: ASCII NUL ('\0') and the equal sign ('=').

2.8 C Language Definitions

2.8.2 POSIX Symbols

In addition to _POSIX_SOURCE, the following additional feature-test macros are supported on the DG/UX system:

- _SYSV3_SOURCE: If this symbol is defined, then features from AT&T's System V Release 3 become visible.
- _SYSV4_SOURCE: If this symbol is defined, then features from AT&T's System V Release 4 become visible.
- _BSD_SOURCE: If this symbol is defined, then features from the Berkeley Software Distribution (BSD) become visible.
- _M88KBCS_SOURCE: If this symbol is defined, then features from 88open's M88000 Binary Compatibility Standard become visible.
- _DGUX_SOURCE: If this symbol is defined, then all features from the DG/UX system become visible.

- _XOPEN_SOURCE: If this symbol is defined, then features from X/OPEN's Portability Guide, version 3, become visible.

For information on additional members to structures or unions that are defined by the standard, see the individual DG/UX system header files in and below the directory **/usr/include**.

2.8.2.2 Common Usage-Dependent Support

If the feature test macro _POSIX_SOURCE is not defined in a program using the Common Usage C Language-Dependent environment, then only the symbols made visible in the headers by use of one or more of the other feature test macros listed above will be visible. If no such feature test macros at all are defined, the effect will be equivalent to defining the symbol _DGUX_SOURCE, and all symbols in the header will become visible.

2.9 Numerical Limits

2.9.2 Minimum Values

The constants listed in Table 2-2 of the standard are defined in the file **<limits.h>** with the same values as in the standard.

2.9.3 Run-Time Increasable Values

The constant {NGROUPS_MAX} is defined in **<limits.h>** with value 0. Applications wishing to use supplemental groups must use the **sysconf**() function in order to see if they are available.

2.9.4 Run-Time Invariant Values (Possibly Indeterminate)

The following symbols are not defined in **<limits.h>** if the feature test macro _POSIX_SOURCE has been defined. The **sysconf**() function must be used to determine the actual values of these parameters:

- {ARG_MAX}
- {CHILD_MAX}
- {OPEN_MAX}

2.9.5 Pathname Variable Values

The following symbols are not defined in **<limits.h>** if the feature test macro _POSIX_SOURCE has been defined. The **pathconf**() or **fpathconf**() functions must be used to determine the actual values of these parameters:

- {LINK_MAX}
- {MAX_CANON}
- {MAX_INPUT}
- {NAME_MAX}
- {PATH_MAX}
- {PIPE_BUF}

2.10 Symbolic Constants

2.10.1 Symbolic Constants for the access() Function

The following **access**() function symbols are defined in **<unistd.h>**:

```
#define F_OK    0
#define X_OK    1
#define W_OK    2
#define R_OK    4
```

2.10.2 Symbolic Constants for the lseek() Function

The following **lseek**() function symbols are defined in **<unistd.h>**:

```
#define SEEK_SET    0
#define SEEK_CUR    1
#define SEEK_END    2
```

2.10.3 Compile-Time Symbolic Constants for Portability Specifications

The following portability symbols are defined in **<unistd.h>**:

```
#define _POSIX_JOB_CONTROL    1
#define _POSIX_SAVED_IDS      1
#define _POSIX_VERSION        198808L
```

2.10.4 Execution-Time Symbolic Constants for Portability Specifications

The following symbols are not defined in **<unistd.h>**. Applications that need to determine DG/UX system behavior with respect to these options must use the **pathconf**() or **fpathconf**() functions to query the system about a specific file.

- {_POSIX_CHOWN_RESTRICTED}
- {_POSIX_NO_TRUNC}
- {_POSIX_VDISABLE}

3. Process Primitives

3.1 Process Creation and Execution

3.1.1 Process Creation

3.1.1.2 Description

The DG/UX system includes several process characteristics not defined in the POSIX.1 standard. When a new process is created, all such extended characteristics are inherited unchanged from the parent process, with the following exceptions:

- All semaphore adjustment values (see documentation for the **semop**() system call) are cleared.
- Process locks, text locks and data locks (see documentation for the **plock**() system call) are not inherited.
- The new process's resource-consumption counts for both itself and for its children (see documentation for the **setrlimit**() system call) are initialized to zero.
- Any pending SIGALRM signals are discarded.
- The new process is not traced, regardless of whether the parent process is traced, unless explicitly set up for tracing by **ptrace**() (see documentation for the **ptrace**() and **dg_xtrace**() system calls).

3.1.1.4 Errors

fork() will fail with the error [ENOMEM] and no child process will be created if the process requires more memory than the system is able to supply.

3.1.2 Execute a File

3.1.2.2 Description

The number of bytes available for the process's combined argument and environment lists is {ARG_MAX}. Null terminators, pointers, and alignment bytes are included in this total.

If the **PATH** variable is not present in the environment, then only the current working directory is searched in order to locate the image specified by the *file* argument.

The DG/UX system includes several process characteristics not defined in the POSIX.1 standard. When a process's image is replaced with another, all such extended characteristics are kept unchanged, with the following exceptions:

- If the calling process is a child created by **vfork**(), all shared memory segments are returned to its parent. Otherwise, all shared memory segments attached to the calling process will be detached.
- If the calling process is a child created by **vfork**(), any text or data segment locks are returned to its parent (see documentation for the **plock**() system call). Otherwise, all text and data segment locks will be released.
- Profiling is disabled for the new process; see documentation for the **profil**() system call.
- If the process is being traced, it is sent a SIGTRAP signal just before the new program image begins to execute (see documentation for the **ptrace**() and **dg_xtrace**() system calls).

3.1.2.4 Errors

Only regular files may be executed; attempts to execute files of other types will fail with the error [EACCES].

The **exec**() functions will fail with the error [ENOMEM] if the new process requires more memory than is allowed by the system-imposed maximum MAXMEM.

3.2 Process Termination

3.2.1 Wait for Process Termination

3.2.1.2 Description

The **wait**() and **waitpid**() functions report status if a child process that is being traced (by the **ptrace**() or **dg_xtrace**() system calls) stops. In this case, the 8 low order bits of the wait status will contain the octal value 0177, and the 8 high order bits will contain the value of the signal that stopped the child process.

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes will be assigned a new parent process ID of 1, which is the process ID of the special system initialization process.

3.2.2 Terminate a Process

3.2.2.2 Description

Children of terminated process will be assigned a new parent process ID of 1, which is the process ID of the special system initialization process.

The SIGCHLD signal is supported, so a SIGCHLD signal will be sent to the parent process.

Job control is supported, so a SIGHUP signal followed by a SIGCONT signal will be sent to every process in a newly orphaned process group.

3.3 Signals

3.3.1 Signal Concepts

3.3.1.1 Signal Names

The following additional signals are defined in the DG/UX system. See the system documentation for the **signal**() system call for details.

- SIGTRAP – trace trap
- SIGBUS – bus error
- SIGSYS – bad argument to system call
- SIGPWR – power-fail restart
- SIGWINCH – window size changed
- SIGPOLL – pollable event occurred
- SIGURG – urgent socket data has arrived
- SIGIO – I/O is now possible
- SIGXCPU – exceeded CPU time limit
- SIGFXSZ – exceeded file size limit
- SIGVTALRM – virtual time alarm
- SIGPROF – profiling time alarm
- SIGLOST – file record locks revoked

- SIGDGNOTIFY – notification message has been placed in the process's message queue

3.3.1.2 Signal Generation and Delivery

If a subsequent occurrence of a pending signal is generated, the signal is, in most cases, not delivered more than once. The exception to this rule is due to properties of the MC88100 CPU: if concurrently executing instructions generate imprecise exceptions which then generate different instantiations of the same signal, then the signal handler for that signal will be entered once for each instantiation of the signal.

In addition to the conditions specified in the standard, the DG/UX system will generate the following signals due to the conditions specified below:

- SIGTRAP – trace trap
- SIGBUS – bus error
- SIGSYS – bad argument to system call
- SIGPWR – power-fail restart
- SIGWINCH – window size changed
- SIGPOLL – pollable event occurred
- SIGURG – urgent socket data has arrived
- SIGIO – I/O is now possible
- SIGXCPU – exceeded CPU time limit
- SIGFXSZ – exceeded file size limit
- SIGVTALRM – virtual time alarm
- SIGPROF – profiling time alarm
- SIGLOST – file record locks revoked
- SIGDGNOTIFY – notification message has been placed in the process's message queue

3.3.2 Send a Signal to a Process

3.3.2.2 Description

If a receiving process's effective user ID has been altered through the use of the S_ISUID mode bit, the DG/UX system permits the application to receive a signal sent by the parent process or by a process with the same real user ID.

The SIGCONT signal is supported, so the user ID tests described in the standard will not be applied when sending SIGCONT to a process that is a member of the same session as the sending process.

The following set of special system processes are unaffected by all calls of the form **kill(0,** *sig*):

- The system initialization process (PID 1).

The DG/UX system does not provide extended security controls, so no further restrictions are placed on the sending of signals.

If the argument *pid* is negative, but not -1, *sig* shall be sent to all processes whose process group ID is equal to the absolute value of *pid*, and for which the process has permission to send a signal.

3.3.4 Manipulate Signal Sets

When the value of the *signo* argument is an invalid or unsupported signal number, the **sigaddset()**, **sigdelset()**, and **sigismember()** functions return −1 and set **errno** to [EINVAL].

3.3.6 Examine Pending Signals

3.3.6.4 Errors

The **sigpending()** function will generate the following errors under the conditions noted:

- [EFAULT]: The argument *set* specifies an invalid area of the calling process's address space or an address area which does not have write access.

4. Process Environment

4.2 User Identification

4.2.4 Get User Name

4.2.4.3 Returns

The implementation of the **cuserid**() function does not use the **getpwnam**() function, so the result of a user's call to either routine will not be overwritten by a subsequent call to the other routine. However, **cuserid**() and **getpwuid**() do use the same static buffer, so the result of a user's call to either of these routines will be overwritten by a subsequent call to the other.

4.2.4.4 Errors

There are no implementation-defined conditions under which the **getlogin**() or **cuserid**() functions will detect errors.

4.3 Process Groups

4.3.3 Set Process Group ID for Job Control

4.3.3.2 Description

The {_POSIX_JOB_CONTROL} option is supported by the DG/UX system, so the **setpgid**() function is fully supported.

4.4 System Identification

4.4.1 System Name

4.4.1.2 Description

The members of the **utsname** structure are all character strings at most 255 characters in length (not counting the terminating NUL character) with arbitrary contents that have been configured into the system's kernel. No particular semantics are attached to the values of any of the members. The default values of the structure members are

as follows:

```
char        sysname  [256] = "DG/UX";
char        nodename [256] = "no_node";
char        release  [256] = "5.4";
char        version  [256] = "00";
char        machine  [256] = "AViiON";
```

4.4.1.4 Errors

The **uname**() function will generate the following errors under the conditions noted:

- [EFAULT]: The argument *name* points to an invalid address.

4.5 Time

4.5.1 Get System Time

4.5.1.4 Errors

The **time**() function will generate the following errors under the conditions noted:

- [EFAULT]: The argument *tloc* points to an invalid address.

4.5.2 Process Times

4.5.2.4 Errors

The **times**() function will generate the following errors under the conditions noted:

- [EFAULT]: The argument *buffer* points to an invalid address.

4.6 Environment Variables

4.6.1 Environment Access

4.6.1.4 Errors

There are no implementation-defined conditions under which the **getenv**() function will detect errors.

4.7 Terminal Identification

4.7.1 Generate Terminal Pathname

4.7.1.4 Errors

There are no implementation-defined conditions under which the **ctermid**() function will detect errors.

4.7.2 Determine Terminal Device Name

4.7.2.4 Errors

There are no implementation-defined conditions under which the **ttyname**() or **isatty**() functions will detect errors.

4.8 Configurable System Variables

4.8.1 Get Configurable System Variables

4.8.1.2 Description

In addition to the variables listed in the standard, the following configurable system variables are supported by the DG/UX system:

- _SC_BCS_VERSION: get version number of 88open BCS to which the system conforms
- _SC_BCS_VENDOR_STAMP: get BCS vendor stamp of the system
- _SC_BCS_SYS_ID: get system hardware's unique system ID number
- _SC_MAXUMEMV: get maximum user process size, in Kbytes
- _SC_MAXUPROC: get maximum number of simultaneous processes per real user ID
- _SC_MAXMSGSZ: get maximum number of bytes in a message
- _SC_NMSGHDRS: get maximum number of message headers in system
- _SC_SHMMAXSZ: get maximum size of a shared memory segment

- _SC_SHMMINSZ: get minimum size of a shared memory segment
- _SC_SHMSEGS: get maximum number of attached shared memory segments per process
- _SC_NMSYSSEM: get total number of semaphores in system
- _SC_MAXSEMVL: get maximum semaphore value
- _SC_NSEMMAP: get number of semaphore sets
- _SC_NSEMMSL: get number of semaphores per set
- _SC_NSHMMNI: get number of shared memory segments in the system
- _SC_ITIMER_VIRT: determine whether or not system supports a virtual timer
- _SC_ITIMER_PROF: determine whether or not system supports a profiling timer
- _SC_TIMER_GRAN: get granularity (in microseconds) of system's real-time clock
- _SC_PHYSMEM: get system's physical memory size, in Kbytes
- _SC_AVAILMEM: get amount of physical memory available to user processes, in Kbytes
- _SC_NICE: determine whether or not **nice**() process prioritization is supported on system
- _SC_MEMCTL_UNIT: get number of bytes in a **memctl**() memory unit
- _SC_SHMLBA: get number of bytes used as rounding factor on memory addresses by **shmsys**()
- _SC_SVSTREAMS: determine whether or not system supports System V style STREAMS
- _SC_CPUID: get the value stored in the M88100 Processor Identification Register

5. Files and Directories

5.1 Directories

5.1.1 Format of Directory Entries

Directories are implemented as a special type of file on the DG/UX system. Each 512-byte directory block has 8 bytes of self-identification information, followed by 504 bytes of information that are used to store variable-length directory entries. The first 4 bytes of a directory entry are used to store the file serial number of the referenced file; the special serial number 0 indicates that the entry is unused. The next 2 bytes of the directory entry are used to store the length in bytes of the entry. The seventh byte of the directory entry is used to store a special sequence number for use in generating invariant file offset values for directory entries (both local and remote). The eighth byte of the directory entry is used to store the length in bytes of the filename for the entry. The remainder of the entry is used to store that name as a null-terminated C-style string, padded if necessary to a 4-byte alignment.

5.1.2 Directory Operations

5.1.2.4 Errors

opendir will fail with the error [EMFILE] when the maximum number of file descriptors are currently open.

readdir will fail with the error [EBADF] when the file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

closedir will fail with the error [EBADF] when the file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

5.2 Working Directory Pathname

5.2.2.4 Errors

getcwd() returns NULL with the error [EACCESS] if read or search permission was denied for a component of the pathname.

5.3 General File Creation

5.3.1 Open a File

5.3.1.2 Description

Any bits in the *mode* argument other than the file permission bits are ignored, regardless of whether they are set.

If the O_CREAT flag is not set, the O_EXCL flag is treated as if it were not set, regardless of whether it was actually set.

Only regular files and FIFOs can be truncated with the O_TRUNC flag; the flag is ignored for all other types of files.

5.3.3 Set File Creation Mask

5.3.3.2 Description

Any bits in the *cmask* argument other than the file permission bits are ignored, regardless of whether they are set.

5.3.4 Link to a File

5.3.4.2 Description

The DG/UX system's implementation of the **link**() function does not support the linking of files across file systems, or the linking of directories; only the **symlink**() system call may be used for these operations.

The calling process is required to have permission to access the existing file.

5.4 Special File Creation

5.4.1 Make a Directory

5.4.1.2 Description

Any bits in the *mode* argument other than the file permission bits and the set UID and set GID bits are ignored, regardless of whether they are set. (However, the set UID bit has no meaning for directories.)

5.4.2 Make a FIFO Special File

5.4.2.2 Description

Any bits in the *mode* argument other than the file permission bits are ignored, regardless of whether they are set.

5.5 File Removal

5.5.1 Remove Directory Entries

5.5.1.2 Description

The DG/UX system prohibits the use of the **unlink**() function on directories; any such attempts will fail with the error [EPERM].

5.5.2 Remove a Directory

5.5.2.2 Description

The DG/UX system prohibits the removal of the current working directory of the calling process; any such attempts will fail with the error [EINVAL].

If the directory to be removed is currently in use by the system as a mount point for a file system (either local or remote), any attempts to remove it with **rmdir**() will fail with the error [EBUSY]. Note that this includes the root directory, since it is the mount point for the root file system.

5.5.3 Rename a File

5.5.3.4 Errors

The **rename**() function will fail with the error [EBUSY] if an attempt is made to rename a directory that is the mount point of a file system.

The DG/UX system's implementation of the **rename**() function does not support the renaming of files across file systems; any such attempts will fail with the error [EXDEV].

5.6 File Characteristics

5.6.1 File Characteristics: Header and Data Structure

5.6.1.2 <sys/stat.h> File Modes

The DG/UX system does not OR any other implementation-defined bits into the S_IRWXU, S_IRWXG or S_IRWXO bitmasks.

5.6.1.3 <sys/stat.h> Time Entries

In addition to the functions defined by the standard, the implementation-defined function **utimes**() can be used to directly change the **st_atime**, **st_mtime**, or **st_ctime** fields of a file.

5.6.2 Get File Status

5.6.2.2 Description

The DG/UX system has no additional or alternate file access control mechanisms, so there are no additional conditions under which the **stat**() or **fstat**() functions will fail.

5.6.3 File Accessibility

5.6.3.2 Description

If a process has a real user ID of 0 (the superuser), then the **access**() function will indicate success for the X_OK test even if none of the file's execute file permission bits are set.

5.6.4 Change File Modes

5.6.4.2 Description

The S_ISUID and S_ISGID bits will be ignored if the **chown**() function is executed on any file that resides on a file system that was mounted with the **nosuid** option.

Upon successful completion of the **chmod**() function, any other file descriptors in the system that refer to the changed file will continue to provide the same access to the file as they previously did.

5.6.5 Change Owner and Group of a File

5.6.5.2 Description

If a process has appropriate privileges, the S_ISUID and S_ISGID bits will remain undisturbed by the **chown**() function.

5.7 Configurable Pathname Variables

5.7.1 Get Configurable Pathname Variables

5.7.1.2 Description

In addition to the variables listed in the standard, the following configurable pathname variables are supported by the DG/UX system:

- _PC_BLKSIZE: get optimum buffer size (in bytes) for I/O to the file, or 0 if such a size is not available

5.7.1.4 Errors

When the value of *name* is invalid, the **pathconf**() function returns −1 and sets **errno** to [EINVAL].

When search permission is denied for a component of the path prefix, the pathconf() and fpathconf() functions return −1 and set **errno** to [EACCES].

When the implementation does not support an association of the variable *name* with the specified file, the **pathconf**() and **fpathconf**() functions return −1 and set **errno** to [EINVAL].

When the length of the *path* argument exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX} while {_POSIX_NO_TRUNC} is in effect, the pathconf() and fpathconf() functions return −1 and set **errno** to [ENAMETOOLONG].

When the named file does not exist or the path argument points to an empty string, the **pathconf**() and **fpathconf**() functions return −1 and set **errno** to [ENOENT].

When a component of the path prefix is not a directory, the **pathconf**() and **fpathconf**() functions return −1 and set **errno** to [ENOTDIR].

When the fildes argument is not a valid file descriptor, the **fpathconf**() function shall return −1 and set **errno** to [EBADF].

6. Input and Output Primitives

6.4 Input and Output

6.4.1 Read from a File

6.4.1.2 Description

Calls to the **read**() function past end-of-file on device special files are treated the same as for ordinary files; but the result of doing this on a device that is incapable of seeking (such as a terminal) may vary from one type of device to another.

Attempting to read more than {INT_MAX} bytes will not by itself result in an error.

6.4.1.4 Errors

If a **read**() operation is interrupted by a signal and some of the data is transferred successfully, the function will return the number of bytes actually read.

The [EIO] error will also occur if there is a physical I/O error when reading the contents of the file from the file system.

If the value of *nbytes* is greater than {INT_MAX}, up to {UINT_MAX} will be read. If the value is greater than {UINT_MAX}, the high order bits of *nbytes* will be truncated and that number of bytes will be read.

6.4.2 Write to a File

6.4.2.2 Description

If the number of bytes to be written is zero, and the file is a FIFO special file or a block special file, the **write**() function shall return zero and have no other results. If the number of bytes to be written is zero, and the file is a terminal device file, the **write**() function shall return zero and have no other results, assuming that a write of one or more bytes would also have been successful under the same conditions and with enough space in the output buffer.

If the number of bytes to be written exceeds {INT_MAX}, the **write**() operation will fail with the error [EFBIG], since the maximum valid file size is {INT_MAX}.

6.4.2.4 Errors

If the number of bytes to be written exceeds {INT_MAX}, the **write**() operation will fail with the error [EFBIG], since the maximum valid file size is {INT_MAX}.

If a **write**() operation is interrupted by a signal, and some of the data is transferred successfully, the function will return the number of bytes actually written.

The [EIO] error will also occur if there is a physical I/O error when writing the contents of the file to the file system.

6.5 Control Operations on Files

6.5.2 File Control

6.5.2.2 Description

In addition to the F_SETFL flag bits defined in the standard, the DG/UX system allows the following flag bits to be set via the **fcntl**() function:

- FASYNC: signal process group when data is ready
- O_ASYNC: physically flush data on writes before returning
- O_NDELAY: no delay
- FDGNRETRY: all I/O errors are treated as hard errors and are not retried

All other bits in the flag are ignored.

Advisory locking is supported for all files that can be opened for writing.

If the *l_len* field of an **flock** structure is negative, the **fcntl**() operation will fail with the error [EINVAL].

6.5.2.4 Errors

If the *command* is F_SETLKW and a deadlock would exist if the lock were granted, the **fcntl**() operation will fail with the error [EDEADLK].

6.5.3 Reposition Read/Write File Offset

6.5.3.2 Description

The **lseek**() function has no effect on devices that are incapable of seeking (such as terminal devices).

7. Device and Class-Specific Functions

7.1 General Terminal Interface

The DG/UX system supports asynchronous terminal connections and virtual terminal connections, including network virtual terminals and local terminal emulators. Synchronous terminal connections are not supported.

7.1.1 Interface Characteristics

7.1.1.3 The Controlling Terminal

In the DG/UX system, the controlling terminal for a session is allocated in the following manner: If a session leader has no controlling terminal, and it opens a terminal device that is not already associated with a session without using the O_NOCTTY option to **open()**, then that terminal becomes the controlling terminal of the session leader.

7.1.1.5 Input Processing and Reading Data

When the number of bytes stored in the input queue exceeds MAX_INPUT, the entire contents of the input queue are flushed.

7.1.1.6 Canonical Mode Input Processing

When the number of bytes stored in the input line exceeds MAX_CANON, the entire contents of the input line are flushed.

{MAX_CANON} is supported by all terminal interfaces supported by the system.

7.1.1.7 Non-Canonical Mode Input Processing

The value of the {MAX_INPUT} parameter on the DG/UX system will never be less than 255, so it is impossible to set the value of MIN (which is stored in an 8-bit character) to anything greater than {MAX_INPUT}.

7.1.1.8 Writing Data and Output Processing

The DG/UX system provides a buffering mechanism for terminals, so the completion of a **write()** call does not mean that all of the bytes scheduled for transmission to the device have necessarily been completed yet.

7.1.1.9 Special Characters

The START and STOP characters can be changed via the **tcsetattr**() function.

There are no multibyte sequences that have a meaning different from the meaning of the bytes when considered individually.

If IEXTEN is set, the following additional single-byte characters are recognized: EOL2 (additional line input line delimiter) and SWTCH (shell layers switch character).

7.1.2 Settable Parameters

7.1.2.1 termios Structure

The members of the **termios** structure are as follows:

```
struct termios
      {
      tcflag_t        c_iflag;
      tcflag_t        c_oflag;
      tcflag_t        c_cflag;
      tcflag_t        c_lflag;
      char            c_line;
      cc_t            cc [NCCS];
      };
```

The `c_line` field indicates which line discipline is in use for the terminal. The total size of the `termios` structure is 36 bytes.

7.1.2.2 Input Modes

Since the only terminal devices supported by the DG/UX system use asynchronous serial data transmission, the break condition is always interpreted according to the standard.

When the number of characters in the input queue rises above seven-eighths the value of MAX_INPUT, a STOP character is sent. The terminal is prevented from sending more data until the number of characters in the input queue falls below one-fourth the value of MAX_INPUT, when a START character is sent.

The initial input control value after **open**() is 0 (no mode bits set).

7.1.2.3 Output Modes

If OPOST is set, output data is processed according to the values of the rest of the output control modes. See the **tty**(7) manual page for details about what modes are available.

The initial output control value after **open**() is 0 (no mode bits set).

7.1.2.4 Control Modes

The initial hardware control value after **open**() is that the following attributes are set: B300, CS8, CREAD, HUPCL.

7.1.2.5 Local Modes

If ECHOE and ICANON are set, the ERASE character will cause the terminal to echo the erase character, even if the display is currently at the beginning of the line.

If ECHOK and ICANON are set, the KILL character causes the terminal to echo the '0 (newline) character after the KILL character.

If IEXTEN is set, the DG/UX system extensions of VEOL2 (additional line input line delimiter) and VSWTCH (shell layers switch character) will be recognized, providing that ICANON is also set. IEXTEN has no interaction with ISIG, IXON, or IXOFF.

The initial local control value after **open**() is 0 (no mode bits set).

7.1.2.6 Special Control Characters

Since job control is supported, the SUSP character is not ignored.

The number of elements in the *c_cc* array is 19.

The START and the STOP characters can both be changed with the **tcsetattr**() function.

The initial octal values of all control characters are as follows:

VINTR: 0177
VQUIT: 0034
VERASE: 0043
VKILL: 0100
VEOF: 0004
VEOL: 0000
VMIN: 0004
VTIME: 0000
VSTART: 0021

VSTOP: 0023
VSUSP: 0377
VEOL2: 0000
VSWTCH: 0032

7.1.2.7 Baud Rate Function

7.1.2.7.2 Description

Attempts to set unsupported baud rates will be ignored and will not cause any error to be returned by the functions **cfsetispeed**(), **cfsetospeed**(), or **tcsetattr**().

7.1.2.7.4 Errors

There are no implementation-defined conditions under which the **cfgetispeed**(), **cfgetospeed**(), **cfsetispeed**(), or **cfsetospeed**() functions will detect additional errors.

7.2 General Terminal Interface Control Functions

There are no specific commands that allow the use of terminal interface control functions from background processes.

7.2.2 Line Control Functions

7.2.2.2 Description

If the duration is not zero, the **tcsendbreak**() function shall send zero-valued bits for *duration* microseconds.

Except for certain pseudo-terminal device files, the **tcsendbreak**() function will always send some data (zero-valued bits) to generate a break condition. For details on the behavior of pseudo-terminals, see the manual page for the **pty**(7) device.

7.2.3 Get Foreground Process Group ID

7.2.3.2 Description

The {_POSIX_JOB_CONTROL} option is supported by the DG/UX system, so the **tcgetpgrp**() function is fully supported.

7.2.4 Set Foreground Process Group ID

7.2.4.2 Description

The {_POSIX_JOB_CONTROL} option is supported by the DG/UX system, so the **tcsetpgrp**() function is fully supported.

8. Language-Specific Services for the C Programming Language

8.1 Referenced C Language Routines

8.1.1 Extensions to Time Functions

If the first character of the **TZ** environment variable is a colon, then the characters after the colon are interpreted as the name of a rule set describing how time zone information is to be presented. See the **localtime**(3C) manual page for details.

8.1.2 Extensions to setlocale() Function

8.1.2.2 Description

The implementation-defined native environment for the **setlocale**() function on the DG/UX system is the locale named "C", as specified in the C standard.

There are no implementation-defined additional categories for use by the **setlocale**() function.

If the environment variable corresponding to the specified category is not set or is set to the empty string and the **LANG** environment variable is not set or is set to the empty string, the behavior of **setlocale**() is to return the value of the specified category in the current locale, without changing anything in the locale. This holds true even if the *locale* parameter to the function is the empty string.

8.2 FILE-Type C Language Functions

8.2.1 Map a Stream Pointer to a File Descriptor

8.2.1.4 Errors

There are no implementation-defined conditions under which the **fileno**() function will detect additional errors.

8.2.2 Open a Stream on a File Descriptor

8.2.2.2 Description

The DG/UX system does not support any additional values for the *type* argument to the **fdopen**() function.

8.2.2.4 Errors

There are no implementation-defined conditions under which the **fdopen**() function will detect additional errors.

8.3 Other C Language Functions

8.3.2 Set Time Zone

8.3.2.2 Description

If **TZ** is absent from the environment, then calling the **tzset**() function will have no effect on the time zone information used by the **localtime**(), **ctime**(), **strftime**(), and **mktime**() functions; they will use Coordinated Universal Time (also known as GMT).

9. System Databases

9.1 System Databases

If the initial user program field is null, the program **/bin/sh** is used by default.

If the initial working directory field is null, the root directory (/) is used by default.

The following additional field is contained in the **group** database:

- An optional encrypted password

The following additional fields are contained in the *user* database:

- An optional encrypted password
- An optional string of arbitrary text used to hold additional information about the user; the content of this string is not interpreted by any of the functions defined in this standard

9.2 Database Access

9.2.1 Group Database Access

9.2.1.3 Returns

The return values of the **getgrgid**() and **getgrnam**() functions point to a static data area that will be overwritten on each call to either function.

9.2.1.4 Errors

There are no implementation-defined conditions under which the **getgrgid**() and **getgrnam**() functions will detect additional errors.

9.2.2 User Database Access

9.2.2.2 Description

The implementation of the **cuserid**() function does not use the **getpwnam**() function, so the results of either routine will not be overwritten by a subsequent call to the other routine. However, **cuserid**() and **getpwuid**() do use the same static buffer, so the result of a user's call to either of these routines will be overwritten by a

subsequent call to the other.

9.2.2.3 Returns

The return values of the **getpwuid**() and **getpwnam**() functions point to a static data area that will be overwritten on each call to either function.

9.2.2.4 Errors

There are no implementation-defined conditions under which the **getpwuid**() and **getpwnam**() functions will detect additional errors.

10. Data Interchange Format

10.1 Archive/Interchange File Format

Both the extended **tar** format and the extended **cpio** format are supported by the DG/UX system. For information on the format-creating utilities, see the manual pages for the **tar(1)** and **cpio(1)** commands.

10.1.1 Extended tar Format

The blocks in a **tar** archive may or may not be grouped for physical I/O operations; see the manual page for **tar**(1) for details on how many blocks can be written with a single **write**() operation.

All possible eight-bit characters are valid in DG/UX system pathnames, so pathnames are not restricted to the portable filename character set. Since there are no possible file names in a **tar** archive that could create invalid file names or pathnames on the DG/UX system, there are no files that are ignored by the format-reading **tar** utility. All extracted files are thus stored in the file system hierarchy with their names unchanged.

The set UID, set GID, and TSVTX bits are all supported by the DG/UX system.

If the *typeflag* field is set to CHARTYPE or BLKTYPE, the *size* field is ignored. If the *typeflag* field is set to FIFOTYPE, the *size* field is set to zero.

The *devmajor* and *devminor* fields contain, respectively, the major and minor device numbers of the device file. These numbers will be extracted unchanged from the archive, but the resulting device node will not necessarily refer to the same device as it did on the system on which the **tar** archive was created.

No high-performance attributes are available on the DG/UX system, so files whose *typeflag* field have the value '7' are treated as regular files.

The reserved *typeflag* values 'A' through 'Z' are not used or recognized.

10.1.2 Extended cpio Format

The **cpio** archive format on the DG/UX system is always recorded as a series of blocks, the size of which must be even multiples of 512 bytes.

10.1.2.1 Header

The *c_dev* field is used to store the octal representation of the device number of the device on which the file system holding the file being archived resides. The *c_ino* field is used to store the octal representation of the file serial number of the file being archived.

The *c_rdev* field is used to store the octal representation of the device number of the file being archived. This value has meaning only for character or block special files.

10.1.2.2 File Name

All possible eight-bit characters are valid in DG/UX system pathnames, so pathnames are not restricted to the portable filename character set. Since there are no possible filenames in a **cpio** archive that could create invalid filenames or pathnames on the DG/UX system, there are no files that are ignored by the format-reading **tar** utility. All extracted files are thus stored in the file system hierarchy with their names unchanged.

10.1.2.4 Special Entries

For character and block special files, *c_filesize* is defined as 0.

10.1.2.5 cpio Values

Some of the additional file types reserved by the standard are supported by the DG/UX system's **cpio** utility: C_ISBLK, C_ISCHR, C_ISLNK and C_ISSOCK.

All of the mode flags listed in this section are supported by the DG/UX system.

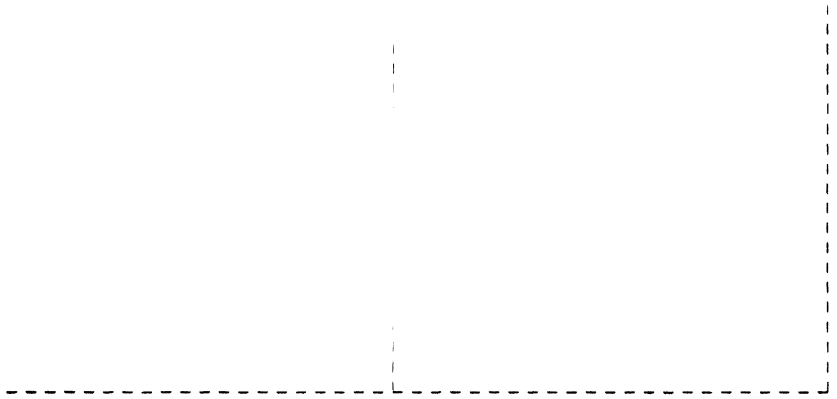
10.1.3 Multiple Volumes

When the **cpio** format-reading utility encounters an end-of-file condition in the format's logical byte stream, it will issue a message to its standard output and then read from its standard input the filename of the next volume in the archive.

When the **tar** format-writing utility encounters an end-of-file condition on the output device, it will issue a message to its standard error, it will read a newline from its standard input, and then continue writing the next volume of the archive.

When the **tar** format-reading utility encounters an end-of-file condition on the input device, the program terminates. The utility may be invoked again to read the next volume of the archive.

End of Document



Cut here and insert in binder spine pocket

Data General

Data General Corporation, Westboro, Massachusetts 01580