GEMDOS Function Reference

Cauxin()

WORD Cauxin(VOID)

Cauxin() waits for the next available data byte from **GEMDOS** handle 2 (normally device 'aux:') and when available, returns it in the low byte of the returned **WORD**.

OPCODE 3 (0x03)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w   #$3,-(sp)
trap     #1
addq.l   #2,sp
```

RETURN VALUE The **WORD** value contains the retrieved byte in the lower eight bits. The contents of the upper 8 bits are currently undefined.

CAVEATS This function can cause flow control problems.

When using this function while its handle is redirected, an end-of-file condition will hang the system. **GEMDOS** version 0.30 and all **MiNT** versions correct this bug. **MiNT** returns **MINT_EOF** (0xFF1A) when the end-of-file is reached.

In addition, if this handle is redirected to something other than 'aux:', an end-of-file will hang the system. Besides these known bugs, this function is used by many 'C' compilers to redirect standard error messages. It is therefore advisable to use **Bconin()** instead.

SEE ALSO **Cauxis()**, **Cauxout()**, **Bconin()**

Cauxis()

WORD Cauxis(VOID)

Cauxis() indicates whether **GEMDOS** handle 2 (normally device 'aux:') has at least one character waiting.

OPCODE 18 (0x12)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$12,-(sp)
```

```
trap        #1
addq.l      #2,sp
```

RETURN VALUE The return value will be **DEV_READY** (-1) if at least one character is available for reading or **DEV_BUSY** (0) if not.

CAVEATS When using this function while its handle is redirected, an end-of-file condition will hang the system. **GEMDOS** version 0.30 and all **MiNT** versions correct this bug. **MiNT** returns **MINT_EOF** (0xFF1A) when the end-of-file is reached.

In addition, some 'C' compilers use this handle as a standard error device. It is therefore advisable to use **Bconstat()**.

SEE ALSO **Cauxin()**, **Cauxout()**, **Cauxos()**, **Bconstat()**

Cauxos()

WORD Cauxos(VOID)

Cauxos() indicated whether **GEMDOS** handle 2 (normally device 'aux:') is ready to receive characters.

OPCODE 19 (0x13)

AVAILABILITY All **GEMDOS** versions

BINDING

```
move.w      #$13,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE A value of **DEV_READY** (-1) is returned if the output device is ready to receive characters or **DEV_BUSY** (0) if it is not.

CAVEATS This function actually returns the status of whatever device **GEMDOS** handle 2 is redirected to. In addition, some 'C' compilers use this handle as a standard error device. It is therefore recommended that **Bcostat()** be used instead.

SEE ALSO **Cauxin()**, **Cauxis()**, **Cauxout()**, **Bcostat()**.

Cauxout()

VOID Cauxout(*ch*)
WORD *ch*;

Cauxout() outputs a character to **GEMDOS** handle 2, normally the 'aux:' device.

OPCODE 4 (0x04)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *ch* is a **WORD** value, however, only the lower eight bits are sent. The upper eight bits must be 0.

BINDING

```
move.w      #ch,-(sp)
move.w      #4,-(sp)
trap        #1
addq.l      #4,sp
```

CAVEATS This function can cause flow control to fail when **GEMDOS** handle 2 is directed to 'aux:'.

In addition, some 'C' compilers use this function as a standard error device. It is therefore recommended that **Bconout()** be used in place of this function.

SEE ALSO **Cauxin()**, **Cauxis()**, **Cauxos()**, **Bconout()**

Cconin()

LONG Cconin(VOID)

Cconin() reads a character (waiting until one is available) from **GEMDOS** handle 0 (normally 'con:').

OPCODE 1 (0x01)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #1,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE The **LONG** value returned is a bit array arranged as follows:

Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
Shift key status (see below)	Keyboard scancode	Unused (0)	ASCII code of character

The ASCII code of the character will be 0 if a non-ascii keyboard key is struck.

CAVEATS When using this function while its handle is redirected, an end-of-file condition will hang the system. **GEMDOS** version 0.30 and all **MiNT** versions correct this bug. **MiNT** returns **MINT_EOF** (0xFF1A) when the end-of-file is reached.

COMMENTS The shift key status will only be returned when bit 3 of the system variable *conterm* (char *(0x484)) is set. This is normally not enabled.

If the handle has been redirected, the inputted character will appear in the lower 8 bits of the return value.

SEE ALSO **Cconis()**, **Cconout()**, **Cconrs()**, **Cnecin()**, **Crawin(), Bconin()**

Cconis()

WORD Cconis(VOID)

Cconis() verifies that a character is waiting to be read from **GEMDOS** handle 0 (normally 'con:').

OPCODE 11 (0xB)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$0B,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Cconis()** returns a **DEV_READY** (-1) if a character is available or **DEV_BUSY** (0) if not.

SEE ALSO **Cconin()**, **Bconstat()**

Cconos()

WORD Cconos(VOID)

Cconos() checks to see whether a character may be output to **GEMDOS** handle 1 (normally 'con:').

OPCODE 16 (0x10)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$10,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE This function returns **DEV_READY** (-1) if at least one character may be sent or **DEV_BUSY** (0) if not.

SEE ALSO **Cconout(), Bcostat()**

Cconout()

VOID Cconout(*ch*)
WORD *ch*;

Cconout() outputs one character via **GEMDOS** handle 1 (normally 'con:').

OPCODE 2 (0x02)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *ch* is a **WORD** value, however, only the lower eight bits are sent through the output stream. The upper eight bits must be 0.

BINDING

```
move.w      ch,-(sp)
move.w      #2,-(sp)
trap        #1
addq.l      #4,sp
```

CAVEATS With **GEMDOS** versions below 0.15, this handle should not be redirected to a write-only device as the call attempts to read from the output stream to process special keys.

COMMENTS No line feed translation is done at the time of output. To start a new line, ASCII 13

and ASCII 10 must both be sent.

SEE ALSO **Cconin()**, **Bconout()**

Cconrs()

VOID Cconrs(*str*)
char **str*;

Cconrs() reads a string from the standard input stream (**GEMDOS** handle 0) and echoes it to the standard output stream (**GEMDOS** handle 1).

OPCODE 10 (0x0A)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *str* should be a character pointer large enough to hold the inputted string. On function entry, *str[0]* should be equal to the maximum number of characters to read.

BINDING

```
pea         str
move.w      #$0A,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE On return, the string buffer passed as a parameter will be filled in with the inputted characters. *str[1]* will contain the actual number of characters in the buffer. (char *) &*str[2]* is the pointer to the start of the actual string in memory.

Cconrs() will not terminate unless CTRL-C is pressed, the buffer is full or either RETURN or CTRL-J is pressed.

CAVEATS **GEMDOS** versions below 0.15 echoes the input to the console even if output has been redirected elsewhere.

COMMENTS The string **Cconrs()** creates is not null-terminated. The following keys *are* processed by the function:

Key	Translation
RETURN	End of input. Do not place RETURN in in buffer.
CTRL-J	End of line. Do not place CTRL-J in buffer.
CTRL-H	Kill last character.
DELETE	Kill last character.
CTRL-U	Echo input line and start over.

CTRL-X	Kill input line and start over.
CTRL-R	Echo input line and continue.
CTRL-C	Exit program.

When the input stream is redirected, **Cconrs()** returns 0 in *str[1]* when the end-of-file marker is reached.

SEE ALSO **Cconin()**, **Cconws()**

Cconws()

VOID Cconws(*str*)
char ***str*;

Cconws() writes a string to **GEMDOS** handle 1 (normally 'con:').

OPCODE 9 (0x09)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *str* is a pointer to a null-terminated character string to be written to the output stream.

BINDING

```
pea         str
move.w      #$09,-(sp)
trap        #1
addq.l      #6,sp
```

CAVEATS With **GEMDOS** versions below 0.15, this handle should not be redirected to a write-only device as the call attempts to read from the output stream to process special keys.

COMMENTS No line feed translation is performed on outputted characters so both an ASCII 13 and ASCII 10 must be sent to force a new line. In addition, the system checks for special keys so a CTRL-C embedded in the string will terminate the process.

SEE ALSO **Cconout()**, **Cconrs()**

Cnecin()

WORD Cnecin(VOID)

Cnecin() is exactly the same as **Cconin()** except that the character fetched from the input stream is not echoed.

OPCODE 8 (0x08)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS None.

BINDING

```
move.w      #8,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE The **LONG** value returned is a bit array arranged as follows:

Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
Shift key status (see below)	Keyboard scancode	Unused (0)	ASCII code of character

The ASCII code of the character will be 0 if a non-ascii keyboard key is struck.

CAVEATS When using this function while its handle is redirected, an end-of-file condition will hang the system. **GEMDOS** version 0.30 and all **MiNT** versions correct this bug. **MiNT** returns **MINT_EOF** (0xFF1A) when the end-of-file is reached.

COMMENTS The shift key status will only be returned when bit 3 of the system variable *conterm* (char *(0x484)) is set. This is normally not enabled.

If the handle has been redirected, the inputted character will appear in the lower 8 bits of the return value.

SEE ALSO **Cconin()**, **Bconin()**

Cprnos()

WORD Cprnos(VOID)

Cprnos() returns the status of **GEMDOS** handle 3 (normally ‘prn:’).

OPCODE 17 (0x11)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS None.

BINDING

```
move.w      #$11,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Cprnos()** returns a **DEV_READY** (-1) if the output stream is ready to receive a character or **DEV_BUSY** (0) if not.

SEE ALSO **Cprnout()**, **Bcostat()**

Cprnout()

WORD Cprnout(*ch*)
WORD *ch*;

Cprnout() sends one character to **GEMDOS** handle 3 (normally 'prn:').

OPCODE 5 (0x05)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *ch* is a **WORD** value, however, only the lower 8 bits are sent to the output stream. The upper eight bits should be 0.

BINDING

```
move.w      ch,-(sp)
move.w      #$5,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Cprnout()** returns a non-zero value if the function successfully wrote the character to the printer or 0 otherwise.

COMMENTS No input translation is performed with this call. Therefore, you must send an ASCII 13 and ASCII 10 to force a new line.

SEE ALSO **Bconout()**

Crawcin()

LONG Crawcin(VOID)

Crawcin() is similar to **Cconout()**, however it does not process any special keys and does not echo the inputted character.

OPCODE 7 (0x07)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$07,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE The **LONG** value returned is a bit array arranged as follows:

Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
Shift key status (see below)	Keyboard scancode	Unused (0)	ASCII code of character

The ASCII code of the character will be 0 if a non-ascii keyboard key is struck.

CAVEATS When using this function while its handle is redirected, an end-of-file condition will hang the system. **GEMDOS** version 0.30 and all **MiNT** versions correct this bug. **MiNT** returns **MINT_EOF** (0xFF1A) when the end-of-file is reached.

COMMENTS The shift key status will only be returned when bit 3 of the system variable *conterm* (char *(0x484)) is set. This is normally not enabled.

If the handle has been redirected, the inputted character will appear in the lower 8 bits of the return value.

Under normal circumstances, when **GEMDOS** handle 0 is being read from, no special system keys, including CTRL-C, are checked.

SEE ALSO **Cconin()**, **Crawio()**, **Bconin()**

Crawio()

LONG Crawio(*ch*)
WORD *ch*;

Crawio() combines console input and output in one function.

OPCODE 6 (0x06)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *ch* is a **WORD** value, however, only the lower eight bits are meaningful and the upper eight bits should be set to 0. If *ch* is 0x00FF on input, **Crawio()** returns the character read from **GEMDOS** handle 0 (normally 'con:').

BINDING

```
move.w      ch,-(sp)
move.w      #6,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE If *ch* is 0x00FF upon entry, **Crawio()** returns a bit array arranged as follows:

Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
Shift key status (see below)	Keyboard scancode	Unused (0)	ASCII code of character

The ASCII code of the character will be 0 if a non-ascii keyboard key is struck.

If no character was waiting in the input stream, **Crawio()** returns a 0.

CAVEATS When using this function while its handle is redirected, an end-of-file condition will hang the system. **GEMDOS** version 0.30 and all **MiNT** versions correct this bug. **MiNT** returns **MINT_EOF** (0xFF1A) when the end-of-file is reached.

Due to the definition of this call it is impossible to write 0x00FF to the output stream or read a zero from this call.

COMMENTS The shift key status will only be returned when bit 3 of the system variable *conterm* (char *(0x484)) is set. This is normally not enabled.

If the handle has been redirected, the inputted character will appear in the lower 8 bits of the return value.

Under normal circumstances, when **GEMDOS** handle 0 is being read from, no special system keys, including CTRL-C, are checked.

SEE ALSO **Cconout()**, **Cconin()**, **Bconout()**, **Bconin()**

Dclosedir()

LONG Dclosedir(***dirhandle*** **)**
LONG ***dirhandle*****;**

Dclosedir() closes the specified directory.

OPCODE 299 (0x12B)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *dirhandle* is a valid directory handle which specifies the directory to close.

BINDING

```
move.l      dirhandle,-(sp)
move.w      #$12B,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Dclosedir()** returns **E_OK** (0) if successful or **EIHNDL** (-37) if the directory handle was invalid.

SEE ALSO **Dopendir(), Dreaddir(), Drewinddir()**

Dcntl()

LONG Dcntl(***cmd, name, arg*** **)**
WORD ***cmd*****;**
char ****name***;**
LONG ***arg*****;**

Dcntl() performs file system specific operations on directories or files.

OPCODE 304 (0x130)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS The only two built-in file systems that support **Dcntl()** calls are 'U:\' and 'U:\DEV.' *cmd* specifies what operation to perform and affects the meaning of *name* and *arg*. Valid *cmd* arguments for 'U:\' are

cmd	Meaning

FS_INSTALL (0xF001)	This mode installs a new file system. *name* must be 'U:\' and *arg* should point to a *fs_descr* structure as follows:
	
	`struct fs_descr`
	`{`
	`  FILESYS  *file_system;`
	`  WORD  dev_no;`
	`  LONG  flags;`
	`  LONG  reserved[4];`
	`};`
	
	If this call is successful, a pointer to a **kerinfo** structure is returned, otherwise the return value is **NULL**. The file system itself is not accessible until this call is made and it is mounted with **FS_MOUNT**.

FS_MOUNT (0xF002)	This mode mounts an instance of an installed file system. *name* should be in the format 'U:\???' where '???' is the name which the file system will be accessed by. *arg* should point to the *fs_descr* structure as above. If the file system is mounted correctly, the *dev_no* field will be updated to reflect the instance number of the mount (file systems may be mounted multiple times).
FS_UNMOUNT (0xF003)	This mode unmounts an instance of a file system. *name* is the name of the file system in the form 'U:\???' where '???' is the name of the file system instance. *arg* should point to the file system *fs_descr* structure.
FS_UNINSTALL (0xF004)	This mode uninstalls a file system identified by the *fs_descr* structure passed in *arg*. A file system can only be sucessfully uninstalled after all instances of it have been unmounted. *name* should be 'U:\'.

Valid *cmd* arguments for 'U:\DEV' are:

cmd	Meaning

DEV_INSTALL (0xDE02)	This command attempts to install a device driver. *name* should be in the format 'U:\DEV\???' where '???' is the name of the device to install. *arg* is a pointer to a *dev_descr* structure as follows:
	
	`struct dev_descr`
	`{`
	`  /* Pointer to a device driver structure */`
	`  DEVDRV *driver;`
	`  /* Placed in aux field of file cookies */`
	`  WORD dinfo;`
	`  /* 0 or O_TTY (0x2000) for TTY */`
	`  WORD flags;`
	`  /* If O_TTY is set, points to tty struct */`
	`  struct tty *tty;`
	`  /* Reserved for future expansion */`
	`  LONG reserved[4];`
	`}`
	
	If the device is successfully installed, **Dcntl()** will return a pointer to a **kerinfo** structure which contains information about the kernel. On failure, **Dcntl()** will return **NULL**. See the section on loadable file systems earlier in this chapter for more information.

DEV_NEWTTY (0xDE00)	This command identifies a **BIOS** terminal device whose name is *name* (in the form 'U:\DEV\DEVNAME' and whose device number is *arg*. This call simply makes the **MiNT** kernel aware of the device. It should have been previously installed by **Bconmap()**. Any attempt to access the device prior to installing it with the **BIOS** will result in an **EUNDEV** (-15) unknown device error. If the device is installed, **Dcntl()** returns a 0 or positive value. A negative return code signifies failure.
DEV_NEWBIOS (0xDE01)	This command is the same as **DEV_NEWTTY** except that it is designed for devices which must have their data transmitted raw (SCSI devices, for example).

BINDING

```
move.l      arg,-(sp)
pea         name
move.w      cmd,-(sp)
move.w      #$130,-(sp)
trap        #1
lea         12(sp),sp
```

VERSION NOTES The **FS_** group of *cmd* arguments are only available as of **MiNT** version 1.08.

Due to a bug in **MiNT** versions 1.08 and below, calling this function with a parameter of **DEV_NEWBIOS** will not have any effect.

RETURN VALUE See above.

SEE ALSO **Bconmap(), Fcntl()**

Dcreate()

LONG Dcreate(***path*** **)**
char ****path*;**

Dcreate() creates a **GEMDOS** directory on the specified drive.

OPCODE 57 (0x39)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *path* is a pointer to a string containing the directory specification of the directory to create. *path* should *not* contain a trailing backslash. Below are some examples and their results.

path	Result
C:\ONE\ATARI	Creates a folder named “ATARI” as a subdirectory of “ONE” on drive ‘C:’.
\ONE\ATARI	Creates a folder named “ATARI” as a subdirectory of “ONE” on the current **GEMDOS** drive.
ATARI	Creates a folder named “ATARI” as a subdirectory of the current **GEMDOS** path on the current **GEMDOS** drive.

BINDING

```
pea         path
move.w      #$39,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE Upon return one of three codes may result:

E_OK (0) :	Operation successful
EPTHNF (-34):	Path not found
EACCDN (-36):	Access denied

CAVEATS Prior to **GEMDOS** version 0.15 **GEMDOS** did not detect if the creation of a subdirectory failed and could therefore leave partially created directories on disk.

SEE ALSO **Ddelete()**

Ddelete()

LONG Ddelete(***path*** **)**
char ****path*;**

Ddelete() removes a directory on the specified drive.

OPCODE 58 (0x3A)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *path* contains the directory specification of the directory you wish to remove. *path* should *not* contain a trailing backslash. For valid examples of *path*, see the entry for **Dcreate()**.

BINDING

```
pea         path
move.w      #$3A,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE Upon return one of four codes may result:

E_OK (0) :	Operation successful
EPTHNF (-34):	Path not found
EACCDN (-36):	Access denied
EINTRN (-65):	Internal error

CAVEATS Prior to **GEMDOS** version 0.15 a **Ddelete()** on a directory recently created will fail but a second attempt will not.

COMMENTS The directory being deleted must be empty or the call will fail.

SEE ALSO **Dcreate()**

Dfree()

LONG Dfree(***buf, drive*** **)**
DISKINFO ****buf*;**
WORD ***drive***;

Dfree() returns information regarding the storage capacity/current usage of the specified drive.

OPCODE 54 (0x36)

AVAILABILITY

All **GEMDOS** versions.

PARAMETERS

buf is a **DISKINFO** pointer which will be filled in on function exit. **DISKINFO** is defined as:

```
typedef struct
{
       /* No. of Free Clusters */
       ULONG b_free;

       /* Clusters per Drive */
       ULONG b_total;

       /* Bytes per Sector */
       ULONG b_secsize;

       /* Sectors per Cluster */
       ULONG b_clsize;
} DISKINFO;
```

drive is a **WORD** which indicates the drive to perform the operation on. A value of **DEFAULT_DRIVE** (0) indicates the current **GEMDOS** drive. A value of 1 indicates drive 'A:', a 2 indicates 'B:', etc...

BINDING

```
move.w      drive,-(sp)
pea         info
move.w      #$36,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE

Upon return, a value of 0 indicates success. Otherwise, a negative **GEMDOS** error code is returned.

CAVEATS

Prior to **GEMDOS** version 0.15 this function is very slow when used on a hard disk.

COMMENTS

To obtain the free number of bytes on a disk, use the formula (*info.b_free* * *info.b_secsize* * *info.b_clsize*). To obtain the total number of bytes available on a disk, use the formula (*info.b_total* * *info.b_secsize* * *info.b_clsize*).

Dgetcwd()

LONG Dgetcwd(***path, drv, size*** **)**
char ****path*;**
WORD ***drv, size***;

Dgetcwd() returns the processes' current working directory for the specified drive.

OPCODE 315 (0x13B)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.96 exists.

PARAMETERS *path* is a pointer to a buffer with room for at least *size* characters into which will be copied the complete working path of drive *drv*.

BINDING

```
pea         path
move.w      size,-(sp)
move.w      drv,-(sp)
move.w      #$13B,-(sp)
trap        #1
add.l       #10,sp
```

RETURN VALUE **Dgetcwd()** returns 0 if successful or a **GEMDOS** error code otherwise.

SEE ALSO **Dgetpath(), Dgetdrv()**

Dgetdrv()

WORD Dgetdrv(VOID)

Dgetdrv() returns the current **GEMDOS** drive code.

OPCODE 25 (0x19)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$19,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Dgetdrv()** returns the current **GEMDOS** drive code. Drive 'A:' is represented by a return value of 0, 'B:' by a return value of 1, and so on.

SEE ALSO **Dsetdrv()**

Dgetpath()

LONG Dgetpath(***buf, drive*** **)**
char ****buf*;**
WORD ***drive***;

Dgetpath() returns the current **GEMDOS** path specification.

OPCODE 71 (0x47)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *buf* is a pointer to a character buffer which will contain the current **GEMDOS** path specification on function exit. *drive* is the number of the drive whose path you want returned. *drive* should be **DEFAULT_DRIVE** (0) for the current **GEMDOS** drive, 1 for drive 'A:', 2 for drive 'B:', and so on.

BINDING

```
move.w      drive,-(sp)
pea         buf
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Dgetpath()** will return one of two errors on function exit:

E_OK (0):	Operation successful
EDRIVE (-49):	Invalid drive specification

COMMENTS As there is no way to specify the buffer size to this function you should allow at least 128 bytes of buffer space. This will allow for up to 8 folders deep. Newer file systems (CD-ROM drives) may demand up to 200 bytes.

SEE ALSO **Dsetpath()**

Dlock()

LONG Dlock(***mode, drv*** **)**
WORD ***mode, drv***;

Dlock() locks a **BIOS** disk device against **GEMDOS** usage.

OPCODE 309 (0x135)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.93 exists.

PARAMETERS Setting *mode* to **DRV_LOCK** (1) places a lock on **BIOS** device *drv* whereas a *mode* setting of **DRV_UNLOCK** (0) unlocks *drv*.

BINDING

```
move.w      drv,-(sp)
move.w      move,-(sp)
move.w      #$135,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Dlock()** returns 0 if successful or a negative **GEMDOS** error code otherwise.

COMMENTS Locking a device provides a method for device formatters to prevent other processes from simultaneously attempting to access a drive. If a process which locked a device terminates, that device is automatically unlocked.

BIOS device numbers and **GEMDOS** drive letters do not necessarily have a one to one correspondence. To lock a **GEMDOS** drive use **Fxattr()** to determine the device number of the drive you wish to lock.

SEE ALSO **Fxattr()**

Dopendir()

LONG Dopendir(***name, flag*** **)**
char ****name*;**
WORD ***flag***;

Dopendir() opens the specified directory for reading.

OPCODE 296 (0x128)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *name* is a pointer to a null-terminated directory specification of the directory to open. *name* should not be contain a trailing backslash.

flag determines whether to open the file in normal or compatibility mode. A value of **MODE_NORMAL** (0) for *flag* signifies normal mode whereas a value of **MODE_COMPAT** (1) signifies compatibility mode.

Compatibility mode forces directory searches to be performed much like **Fsfirst()** and **Fsnext()** (restricting filenames to the **DOS** 8 + 3 standard in uppercase). In normal mode, filenames returned by **Dreaddir()** will be in the format native to the

file system and a **UNIX** style file index will be returned.

BINDING

```
move.w      flag,-(sp)
pea         name
move.w      #$128,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Dopendir()** returns a **LONG** directory handle (which may be positive or negative) if successful. A negative **GEMDOS** error code will be returned if the call fails.

CAVEATS Failure to properly close directory handles may cause the system to eventually run out of handles which will cause the **OS** to fail.

COMMENTS Negative directory handles and negative **GEMDOS** error codes may be differentiated by checking for 0xFF in the high byte. Returned values with 0xFF in the high byte are errors.

SEE ALSO **Dclosedir(), Dreaddir(), Drewinddir()**

Dpathconf()

LONG Dpathconf(***name, mode*** **)**
char ****name*;**
WORD ***mode***;

Dpathconf() returns information regarding limits and capabilities of an installed file system.

OPCODE 292 (0x124)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *name* specifies the file system you wish information about. *mode* dictates the return value as follows:

Name	*mode*	Return Value
DP_INQUIRE	-1	Returns the maximum legal value for the mode parameter in **Dpathconf()**.
DP_IOPEN	0	Retuns the possible maximum number of open files at one time. If **UNLIMITED** (0x7FFFFFFF) is returned, then the number of open files is limited only by available memory.
DP_MAXLINKS	1	Returns the maximum number of links to a file. If **UNLIMITED** (0x7FFFFFFF) is returned, then the number of links to a file is limited only by available memory.

DP_PATHMAX	2	Returns the maximum length of a full path name in bytes. If **UNLIMITED** (0x7FFFFFFF) is returned, then the maximum size of a pathname is unlimited.
DP_NAMEMAX	3	Returns the maximum length of a file name in bytes. If **UNLIMITED** (0x7FFFFFFF) is returned, then the maximum length of a filename is unlimited.
DP_ATOMIC	4	Returns the number of bytes that can be written per write operation. If **UNLIMITED** (0x7FFFFFFF) is returned, then the number of bytes that can be written at once is limited only by available memory.
DP_TRUNC	5	Returns a code indicating the type of filename truncation as follows: **DP_NOTRUNC (0)** File names are not truncated. If a file name in any system call exceeds the filename size limit then an **ERANGE** (-64) range error is returned. **DP_AUTOTRUNC (1)** File names are truncated automatically to the maximum allowable length. **DP_DOSTRUNC (2)** File names are truncated to the **DOS** standard (maximum 8 character node with 3 character extension).
DP_CASE	6	Returns a code which indicates case sensitivity as follows: **DP_SENSITIVE (0)** File system is case-sensitive. **DP_NOSENSITIVE (1)** File system is not case-sensitive (file and path names are always converted to upper-case). **DP_SAVEONLY (2)** File system is not case-sensitive, however, file and path names are saved in their original case. Ex: A file called 'Compendi.um' will appear as 'Compendi.um' but may be referenced as 'compendi.um' or 'COMPENDI.UM'.

BINDING

```
move.w      mode,-(sp)
pea         name
move.w      #$124,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE See above.

SEE ALSO **Sysconf()**

Dreaddir()

LONG Dreaddir(*len, dirhandle, buf*)
WORD *len*;
LONG *dirhandle*;
char **buf*;

Dreaddir() enumerates the contents of the specified directory.

OPCODE

297 (0x129)

AVAILABILITY

Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS

Dreaddir() fetches information about the next file contained in the directory specified by *dirhandle*. *len* specifies the length of the buffer pointed to by *buf* which should be enough to hold the size of the filename, **NULL** byte, and index (if in normal mode).

BINDING

```
pea         buf
move.l      dirhandle
move.w      len
move.w      #$129,-(sp)
trap        #1
lea         12(sp),sp
```

RETURN VALUE

Dreaddir() returns a 0 if the operation was successful, **ERANGE** (-64) if the buffer was not large enough to hold the index and name, or **ENMFIL** (-47) if there were no more files to read.

COMMENTS

In normal mode, **Dreaddir()** returns a 4-byte file index in the first four bytes of *buf*. The filename then follows starting at the fifth byte of *buf*. The file index is present to prevent confusion under some file systems when two files of the same name exist. In some file systems this is legal, however, in all file systems, the 4-byte index will be unique.

When in compatibility mode, the filename begins at &*buf[0]*.

SEE ALSO

Dopendir(), Dclosedir(), Drewinddir()

Drewinddir()

LONG Drewinddir(*handle*)
LONG *handle*;

Drewinddir() rewinds the specified directory pointer to its first file.

OPCODE 298 (0x12A)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *handle* specifies the directory handle of the directory to rewind.

BINDING

```
move.l      handle,-(sp)
move.w      #$12A,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Drewinddir()** returns a 0 if successful or a negative **GEMDOS** error code otherwise.

SEE ALSO **Dopendir(), Dreaddir(), Drewinddir()**

Dsetdrv()

LONG Dsetdrv(*drive*)
WORD *drive*;

Dsetdrv() sets the current **GEMDOS** drive and returns a bitmap of mounted drives.

OPCODE 14 (0x0E)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *drive* is the code of the drive to set as the default **GEMDOS** disk drive. Calling the function as:

```
bmap = Dsetdrv(Dgetdrv());
```

will return the bitmap of mounted drives without changing the current **GEMDOS** drive.

BINDING

```
move.w      drive,-(sp)
```

```
move.w      #$0E,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Dsetdrv()** returns a **LONG** bit array that indicates which drives are mounted on the system. Bit 0 indicates drive 'A:', bit 1 drive 'B:', etc.

SEE ALSO **Dgetdrv()**

Dsetpath()

LONG Dsetpath(***path*** **)**
char ****path*;**

Dsetpath() sets the path of the current **GEMDOS** drive.

OPCODE 59 (0x3B)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *path* is a pointer to a character buffer containing the new path specification for the current **GEMDOS** drive.

BINDING

```
pea         path
move.w      #$3B,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Dsetpath()** returns one of two return codes on function exit:

E_OK (0):	Operation successful
EPTHNF (-34):	Path not found

CAVEATS You may specify a drive letter and colon in the input path specification to set the path of a particular drive but this feature is unstable in all versions of **GEMDOS** and may confuse drive assignments. It is therefore advised that this feature be avoided.

SEE ALSO **Dgetpath()**

Fattrib()

LONG Fattrib(*fname, flag, attr*)
char ****fname*;**
WORD *flag, attr*;

Fattrib() reads or modifies the attribute bits of a **GEMDOS** file.

OPCODE 67 (0x43)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *fname* is a pointer to a null-terminated string which contains the **GEMDOS** filename of the file to manipulate. *flag* should be set to **FA_INQUIRE** (0) to read the file's attributes and **FA_SET** (1) to set them. If you are setting attributes, *attr* contains the file's new attributes.

BINDING

```
move.w      attr,-(sp)
move.w      flag,-(sp)
pea         fname
move.w      #$43,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE If reading the attributes, **Fattrib()** returns a bit array of attributes as defined below. If setting the attributes, **Fattrib()** returns the file's old attributes. In any case, a negative return code indicates that a **GEMDOS** error occurred.

Name	Bit	Meaning
FA_READONLY	0	Read-only flag
FA_HIDDEN	1	Hidden file flag
FA_SYSTEM	2	System file flag
FA_VOLUME	3	Volume label flag
FA_DIR	4	Subdirectory
FA_ARCHIVE	5	Archive flag
—	6...	Currently reserved

CAVEATS **GEMDOS** versions below 0.15 did not set the archive bit correctly. The archive bit is now correctly set by **TOS** when a file is created or written to.

Fchmod()

LONG Fchmod(***name*****,** ***mode*** **)**
char ****name***;**
WORD ***mode*****;**

Fchmod() alters file access permissions of the named file.

OPCODE 306 (0x132)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *name* specifies a valid **GEMDOS** file specification of the file whose access permissions you wish to modify. *mode* is a bit mask composed by OR'ing together values defined as follows:

Name	Mask	Meaning
S_IRUSR	0x100	Read permission for the owner of the file.
S_IWUSR	0x80	Write permission for the owner of the file.
S_IXUSR	0x40	Execute permission for the owner of the file.
S_IRGRP	0x20	Read permission for members of the same group the file belongs to.
S_IWGRP	0x10	Write permission for members of the same group the file belongs to.
S_IXGRP	0x08	Execute permission for members of the same group the file belongs to.
S_IROTH	0x04	Read permission for all others.
S_IWOTH	0x02	Write permission for all others.
S_IXOTH	0x01	Execute permission for all others.

BINDING

```
move.w      mode,-(sp)
pea         name
move.w      #$132,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Fchmod()** returns **E_OK** (0) if successful or a negative **GEMDOS** error code otherwise.

CAVEATS Not all file systems support all bits. Unrecognized bits will be ignored.

COMMENTS Only the owner of a file may change a file's permission status.

'Execute' status refers to the permission to search the named directory for a file name or component.

SEE ALSO **Fattrib(), Fxattr()**

Fchown()

LONG Fchown(*name, uid, gid*)
char ****name*;**
WORD *uid, gid*;

Fchown() changes a file's ownership.

OPCODE 305 (0x131)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *name* specifies the file whose ownership status you wish to change. *uid* sets the new owner and *gid* sets the new group.

BINDING

```
move.w      gid,-(sp)
move.w      uid,-(sp)
pea         name
move.w      #$131,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE **Fchown()** returns 0 if the operation was successful or a negative **GEMDOS** error code otherwise.

CAVEATS Most file systems don't understand the concept of file ownership (including **TOS**).

COMMENTS *uid* may only be modifies if the caller's uid is 0. *gid* may only be changed to the group id of a group the caller belongs to.

SEE ALSO **Fchmod(), Fxattr()**

Fclose()

LONG Fclose(*handle*)
WORD *handle*;

Fclose() closes the file specified.

OPCODE 62 (0x3E)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *handle* is a valid **WORD** file handle which will be closed as a result of this call.

BINDING

```
move.w      handle,-(sp)
move.w      #$3E,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Fclose()** returns **E_OK** (0) if the file was closed successfully or **EIHNDL** (-37) if the handle given was invalid.

CAVEATS Calling this function with an invalid file handle will crash the system on **GEMDOS** versions below 0.15. In addition, **GEMDOS** versions below 0.15 will become confused if you close a standard **GEMDOS** handle (0-5).

COMMENTS As of **GEMDOS** version 0.15, closing a standard **GEMDOS** handle (0-5) will simply reset it to its default **BIOS** state.

SEE ALSO **Fcreate()**, **Fopen()**

Fcntl()

LONG Fcntl(***handle, arg, cmd*** **)**
WORD ***handle*****;**
LONG ***arg*****;**
WORD ***cmd*****;**

Fcntl() performs a command on the specified file.

OPCODE 260 (0x104)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *handle* specifies the **GEMDOS** file handle of the file on which the operation specified by *cmd* will affect. *arg* varies with each command. Valid commands are:

cmd	Meaning
F_DUPFD (0x0000)	Duplicate the given file handle. **Fcntl()** will return a file handle in the range *arg* – 32. If no file handles exist within that range, an error will be returned.
F_GETFD (0x0001)	Return the inheritance flag for the specified file. A value of 1 indicates that child processes started with **Pexec()** will inherit this file handle, otherwise a value of 0 is returned. *arg* is ignored.

F_SETFD (0x0002)	Set the inheritance flag for the named file. *arg* specifies if child processes started with **Pexec()** will inherit the file handle. A value of 0 indicates that they will not. A value of 1 indicates that they will. **GEMDOS** handles 0-5 default to a value of 1whereas other handles default to a value of 0.
F_GETFL (0x0003)	Return the file descriptor flags for the specified file. These are the same flags passed to **Fopen()**. *arg* is ignored.
F_SETFL (0x0004)	Set the file decriptor flags for the specified file to *arg*. Only user-modifyable bits are considered. All others should be 0. It is not possible to change a file's read/write mode or sharing modes with this call. Attempts to do this will fail without returning an error code.

F_GETLK (0x0004)	Test for the presence of a lock on the specified file. arg is a pointer to a **FLOCK** structure defined as follows:
	
	`typedef struct flock`
	`{`
	`  /* Type of lock`
	`  0 = Read-only lock`
	`  1 = Write-only lock`
	`  2 = Read/Write lock */`
	`  WORD l_type;`
	`  /* 0 = offset from beginning of file`
	`  1 = offset from current position`
	`  2 = offset from end of file */`
	`  WORD l_whence;`
	`  /* Offset to start of lock */`
	`  LONG l_start;`
	`  /* Length of lock (0 for rest of file) */`
	`  LONG l_len;`
	`  /* Process ID maybe filled in by call */`
	`  WORD l_pid;`
	`} FLOCK;`
	
	If a prior lock exists which would prevent the specified lock from being applied, the interfering lock is copied into the structure with the process ID of the locking process. Otherwise, **Fcntl()** returns **F_UNLCK** (3).

F_SETLK (0x0005)	Set or remove an advisory lock on the specified file. *arg* points to a **FLOCK** structure as defined above. Setting *l_type* to **F_RDLOCK** or **F_WRLCK** will cause a lock to be set. Setting *l_type* to **F_UNLCK** wil attempt to remove the specified lock. When locking and unlocking FIFO's, *l_whence*, *l_start*, and *l_len* should be 0. The command returns 0 if successful or a negative **GEMDOS** error code otherwise.
F_SETLKW (0x0007)	The calling procedure is the same as above, however, if other processes already have a conflicting lock set, it will suspend the calling process until the lock is freed.
FSTAT (0x4600)	Get the extended attributes for a file. *arg* points to a **XATTR** structure which is filled in with the file's extended attributes. If successful, the function returns 0, otherwise a negative **GEMDOS** error code is returned. See **Fxattr()** for an explanation of the **XATTR** structure.

FIONREAD (0x4601)	Return an estimate of the number of bytes available for reading from the specified file without causing the process to block (wait for more input) in the **LONG** pointed to by *arg*.
FIONWRITE (0x4602)	Return an estimate of the number of bytes that may be written from the specified file without causing the process to block in the **LONG** pointed to by *arg*.
SHMGETBLK (0x4D00)	Returns the address of a memory block associated with the file. arg should be **NULL** for future compatibility. Note: Different processes may receive different addresses for a shared block.
SHMSETBLK (0x4D01)	*arg* points to a block of memory (previously allocated) which is to be associated with the file. The file must have been created at 'U:\SHM\' or the call will fail.
PPROCADDR (0x5001)	Return the address of the specified processes' control structure (opened as a file) in *arg*. See the discussion of **MiNT** processes for information about this structure.
PBASEADDR (0x5002)	Return the address of the specified processes' **GEMDOS** basepage (opened as a file) in *arg*,
PCTXTSIZE (0x5003)	Return the length of the specified processes' context structure (opened as a file) in *arg*. Seeking to the offset returned by **PPROCADDR** minus this number and reading this many bytes will yield the current user context of the process. Seeking back this many bytes more and reading will yield the last system context of the process. This structure is volatile and is likely to change from one **MiNT** version to the next.
PSETFLAGS (0x5004)	*arg* is a pointer to a **LONG** from which the processes' memory allocation flags (***PRGFLAGS***) will be set.
PGETFLAGS (0x5005)	*arg* is a pointer to a **LONG** into which the processes' memory allocation flags (***PRGFLAGS***) will be placed.
PTRACEGFLAGS (0x5006)	*arg* points to a **WORD** which will be filled in with the trace flags of a process. Setting bit #0 of *arg* causes the parent process to receive signals destined for the child. See the discussion on program debugging for more information.
PTRACESFLAGS (0x5007)	*arg* points to a **WORD** which will be used to set the trace flags of a process. See the discussion on program debugging for more information.
PTRACEGO (0x5008)	This call restarts a process which was stopped because of a signal. *arg* points to a **WORD** which contains 0 to clear all of the child processes' pending signals or the signal value to send to the process.
PTRACEFLOW (0x5009)	This call restarts a process in a special tracing mode in which the process is stopped and a **SIGTRACE** signal is generated whenever program flow changes (ex: JSR/BSR/JMP/BEQ). *arg* should be set to 0 to clear all of the pending signals of the process being traced or a signal value which is to be sent to the child.
PTRACESTEP (0x500A)	This call restarts a process and allows it to execute one instruction before a **SIGTRAP** instruction is generated.

PLOADINFO (0x500C)

arg points to a structure as follows:

```
struct ploadinfo
{
       WORD      fnamelen;
       char *    cmdlin;
       char *    fname;
};
```

cmdlin should point to a 128 byte character buffer into which the processes' command line will be copied.

fname should point to a buffer *fnamelen* bytes long into which the complete path and filename of the process' parent will be copied. If the buffer is too short the call will return **ENAMETOOLONG**.

TIOCGETP (0x5400)

Get terminal parameters from the TTY device with the specified file handle. *arg* is a pointer to an **sgttyb** structure which is filled in by this command.

```
struct sgttyb
{
   /* Reserved */
   char sg_ispeed;
   /* Reserved */
   char sg_ospeed;
   /* Erase character */
   char sg_erase;
   /* Line kill character */
   char sg_kill;
   /* Terminal control flags */
   WORD sg_flags;
};
```

TIOCSETP (0x5401)

Set the terminal parameters of the TTY device specified. *arg* is a pointer to an **sgyttb** structure as defined above. You should first get the terminal control parameters, modify what you wish to change, and then set them with this call.

TIOCGETC (0x5402)

Get the terminal control characters of the TTY device specified. *arg* is a pointer to a **tchars** structure filled in by this call which is defined as follows:

```
struct tchars
{
   /* Raises SIGINT */
   char t_intrc;
   /* Raises SIGKILL */
   char t_quitc;
   /* Starts terminal output */
   char t_startc;
   /* Stops terminal output */
   char t_stopc;
   /* Marks end of file */
   char t_eofc;
   /* Marks end of line */
   char t_brkc;
};
```

TIOCSETC (0x5403)

Set the terminal control characters of the TTY device specified. *arg* is a pointer to a **tchars** structure as defined above. Setting any structure element to 0 disables that feature.

TIOCGLTC (0x5404)	Get the extended terminal control characters from the TTY device specified. *arg* is a pointer to a **ltchars** structure which is filled in by this call defined as follows:
	
	`struct ltchars`
	`{`
	`  /* Raise SIGTSTP now */`
	`  char t_suspc;`
	`  /* Raise SIGTSTP when read */`
	`  char t_dsuspc;`
	`  /* Redraws the input line */`
	`  char t_rprntc;`
	`  /* Flushes output */`
	`  char t_flushc;`
	`  /* Erases a word */`
	`  char t_werasc;`
	`  /* Quotes a character */`
	`  char t_lnextc;`
	`};`

TIOCSLTC (0x5405)	Set the extended terminal control characters for the TTY device specified from the **ltchars** structure pointed to by *arg*.
TIOCGPGRP (0x5406)	Return the process group ID for the TTY specified in the **LONG** pointed to by *arg*.
TIOCSPGRP (0x5407)	Set the process group ID of the TTY specified in the **LONG** pointed to by *arg*.
TIOCSTOP (0x5409)	Stop terminal output (as if the user had pressed CTRL-S). *arg* is ignored.
TIOCSTART (0x540A)	Restart output to the terminal (as if the user had pressed CTRL-Q) if it had been previously stopped with CTRL-S or a **TIOCSTOP** command. *arg* is ignored.

TIOCGWINSZ (0x540B)	Get information regarding the window for this terminal. *arg* points to a **winsize** structure which is filled in by this command.
	
	`struct winsize`
	`{`
	`  /* # of Text Rows */`
	`  WORD ws_row;`
	`  /* # of Text Columns */`
	`  WORD ws_column;`
	`  /* Width of window in pixels */`
	`  WORD ws_xpixel;`
	`  /* Height of window in pixels */`
	`}`

TIOCSWINSZ (0x540C)	Change the extents of the terminal window for the specified TTY. *arg* points to a **winsize** structure which contains the new window information. It is up to the window manager to modify the window extents and raise the **SIGWINCH** signal if necessary.

<table>
<tr><td>TIOCGXKEY
(0x540D)</td><td>Return the current definition of a system key. arg points to a structure xkey as follows:

<pre>struct xkey
{
        WORD xk_num;
        char xk_def[8];
};</pre>
xk_def will be filled in with the NULL terminated name associated with the key specified in xk_num as follows:

<table><tr><th>xk_num</th><th>Key</th></tr><tr><td>0-9</td><td>F1-F10</td></tr><tr><td>10-19</td><td>F11-F20</td></tr><tr><td>20</td><td>Cursor up</td></tr><tr><td>21</td><td>Cursor down</td></tr><tr><td>22</td><td>Cursor right</td></tr><tr><td>23</td><td>Cursor left</td></tr><tr><td>24</td><td>Help</td></tr><tr><td>25</td><td>Undo</td></tr><tr><td>26</td><td>Insert</td></tr><tr><td>27</td><td>Clr/Home</td></tr><tr><td>28</td><td>Shift+Cursor up</td></tr><tr><td>29</td><td>Shift+Cursor down</td></tr><tr><td>30</td><td>Shift+Cursor right</td></tr><tr><td>31</td><td>Shift+Cursor left</td></tr></table></td></tr>
<tr><td>TIOCSXKEY
(0x540E)</td><td>Set the current definition of a system key. arg must point to an xkey structure (as defined above). xk_num and xk_def are used to set the text associated with a system key.

If a terminal recognizes special keys (by having its XKEY bit set in the sg_flags field of its sgttyb structure) then setting a system key will cause the text specified by xk_def to be sent to a process whenever the key is struck. Note: this works only if the terminal is reading characters using Fread().</td></tr>
<tr><td>TIOCIBAUD
(0x5412)</td><td>Read/Write the input baud rate for the specified terminal device. If arg points to a LONG then the input baud rate will be set to that value. If arg is 0, the DTR on the terminal will be dropped (if this feature is supported). If arg is negative, the baud rate will not be changed. The old baud rate is returned in the value pointed to by arg.

If the terminal does not support separate input and output baud rates then this call will set both rates.</td></tr>
<tr><td>TIOCOBAUD
(0x5413)</td><td>Read/Write the output baud rate for the specified terminal device. If arg points to a LONG then the output baud rate will be set to that value. If arg is 0, the DTR on the terminal will be dropped (if this feature is supported). If arg is negative, the baud rate will not be changed. The old baud rate is returned in the value pointed to by arg.

If the terminal does not support separate input and output baud rates then this call will set both rates.</td></tr>
<tr><td>TIOCCBRK
(0x5414)</td><td>Clear the break condition on the specified device. arg is ignored.</td></tr>
<tr><td>TIOCSBRK
(0x5415)</td><td>Set the break condition on the specified device. arg is ignored.</td></tr>
</table>

TIOCGFLAGS (0x5416)	Return the current stop bit/data bit configuration for the terminal device in the lower 16 bits of the **LONG** pointed to by *arg*. See the entry for **TIOCSFLAGS** for the flags required to parse *arg*.
TIOCSFLAGS (0x5417)	Set the current stop bit/data bit configuration for the terminal device. The new configuration is contained in *arg*. Valid mask values for *arg* are as follows: **Name** **Mask** **Meaning** **TF_1STOP** 0x0001 1 stop bit **TF_15STOP** 0x0002 1.5 stop bits **TF_2STOP** 0x0003 2 stop bits **TF_8BIT** 0x0000 8 data bits **TF_7BIT** 0x0004 7 data bits **TF_6BIT** 0x0008 6 data bits **TF_5BIT** 0x000C 5 data bits
TCURSOFF (0x6300)	Hide the cursor on the selected terminal device. *arg* is ignored.
TCURSON (0x6301)	Show the cursor on the selected terminal device. *arg* is ignored.
TCURSBLINK (0x6302)	Enable cursor blinking on the selected terminal device. *arg* is ignored.
TCURSSTEADY (0x6303)	Disable cursor blinking on the selected terminal device. *arg* is ignored.
TCURSSRATE (0x6304)	Set the cursor blink rate to the **WORD** pointed to by *arg*.
TCURSGRATE (0x6305)	Return the current cursor blink rate in the **WORD** pointed to by *arg*.

BINDING

```
move.w      cmd,-(sp)
move.l      arg,-(sp)
move.w      handle,-(sp)
move.w      #$260,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE Unless otherwise noted, **Fcntl()** returns a 0 if the operation was successful or a negative **GEMDOS** error code otherwise.

SEE ALSO **Flock(), Fopen(), Fxattr(), Pgetpgrp(), Psetpgrp()**

Fcreate()

LONG Fcreate(*fname*, *attr*)
char **fname*;
WORD *attr*;

Fcreate() creates a new file (or truncates an existing one) with the specified name and attributes.

OPCODE 60 (0x3C)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *fname* is a character pointer to the **GEMDOS** file specification of the file to create or truncate. *attr* is a bit array which specifies the attributes of the new file. Valid mask values are given below:

Name	Bit	Meaning
FA_READONLY	0	Read-only file
FA_HIDDEN	1	Hidden file
FA_SYSTEM	2	System file
FA_VOLUME	3	Volume label
—	4	Reserved
FA_ARCHIVE	5	Archive bit

BINDING

```
move.w      attr,-(sp)
pea         fname,-(sp)
move.w      #$3C,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Fcreate()** returns a **LONG** value. If the **LONG** is negative, it should be interpreted as a **GEMDOS** error. Possible errors are **EPTHNF** (-34), **ENHNDL** (-35) , or **EACCDN** (-36).

If positive, the **WORD** portion of the returned **LONG** should be regarded as the file handle.

CAVEATS With **GEMDOS** version 0.13, creating a read-only file returns a read-only file handle which is of little use. **GEMDOS** versions below 0.15 incorrectly allow more than one volume label per disk.

COMMENTS **GEMDOS** versions 0.15 and above automatically set the archive bit. You may set it yourself on versions below 0.15.

SEE ALSO **Fopen()**, **Fclose()**

Fdatime()

LONG Fdatime(***timeptr, handle, flag*** **)**
DATETIME ****timeptr*;**
WORD ***handle, flag*****;**

Fdatime() reads or modifies a file's time and date stamp.

OPCODE 87 (0x57)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *timeptr* is a pointer to a **DATETIME** structure which is represented below. *handle* is a valid **GEMDOS** file handle to the file to modify. *flag* is **FD_INQUIRE** (0) to fill *timeptr* with the file's date/timestamp and **FD_SET** (1) to change the file's date/timestamp to the contents of *timeptr*.

```
typedef struct
{
        unsigned hour:5;
        unsigned minute:6;
        unsigned second:5;
        unsigned year:7;
        unsigned month:4;
        unsigned day:5;
} DATETIME;
```

BINDING

```
move.w      flag,-(sp)
move.w      handle,-(sp)
pea         timeptr
move.w      #$57,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE **Fdatime()** returns a 0 if the date/time was successfully read/modified. Otherwise, it returns a negative **GEMDOS** error code.

CAVEATS **GEMDOS** versions below 0.15 yielded very unpredictable results with this call and should therefore be avoided.

COMMENTS *timeptr.second* should be multiplied times two to obtain the actual value. *timeptr.year* is expressed as an offset from 1980.

Fdelete()

LONG Fdelete(*fname*)
char **fname*;

Fdelete() deletes the specified file.

OPCODE 65 (0x41)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *fname* is the **GEMDOS** file specification of the file to be deleted.

BINDING

```
pea         fname
move.w      #$41,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Fdelete()** returns **E_OK** (0) if the operation was successful or a negative **GEMDOS** error code if it fails.

CAVEATS Do not attempt to delete a file that is currently open or unpredictable results will occur.

COMMENTS **Ddelete()** must be used to delete subdirectories.

SEE ALSO **Ddelete()**

Fdup()

LONG Fdup(*shandle*)
WORD *shandle*;

Fdup() duplicates a standard file handle (0-5) and assigns it a new handle (>6).

OPCODE 69 (0x45)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *shandle* is the standard **GEMDOS** handle to be duplicated.

BINDING

```
move.w      shandle,-(sp)
move.w      #$45,-(sp)
trap        #1
```

```
addq.l      #4,sp
```

RETURN VALUE **Fdup()** returns a normal **GEMDOS** file handle in the lower **WORD** of the returned **LONG**. If the **LONG** return value is negative then it should be treated as a **GEMDOS** error code.

COMMENTS This function is generally used to save a standard file handle so that an **Fforce()** operation may be undone.

SEE ALSO **Fforce()**

Fforce()

LONG Fforce(***shandle, nhandle*** **)**
WORD ***shandle, nhandle*****;**

Fforce() is used to redirect the standard input or output from a **GEMDOS** standard handle to a specific handle created by the application.

OPCODE 70 (0x46)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *shandle* is a standard **GEMDOS** handle to be redirected. *nhandle* is the new handle you wish to direct it to. Valid values for *shandle* and *nhandle* are as follows:

Name	Handle	GEMDOS Filename	Meaning
GSH_CONIN	0	con:	Standard input (defaults to whichever **BIOS** device is mapped to **GEMDOS** handle -1)
GSH_CONOUT	1	con:	Standard output (defaults to whichever **BIOS** device is mapped to **GEMDOS** handle -1)
GSH_AUX	2	aux:	Currently mapped serial device (defaults to whichever **BIOS** device is mapped to **GEMDOS** handle -2)
GSH_PRN	3	prn:	Printer port (defaults to whichever **BIOS** device is currently mapped to **GEMDOS** handle -3).
—	4	None	Reserved
—	5	None	Reserved
GSH_BIOSCON	-1	None	Refers to **BIOS** handle 2. This handle may only be redirected under the presence of **MiNT**. Doing so redirects output of the **BIOS**.

GSH_BIOSAUX	-2	None	Refers to **BIOS** handle 1. This handle may only be redirected under the presence of **MiNT**. Doing so redirects output of the **BIOS**.
GSH_BIOSPRN	-3	None	Refers to **BIOS** handle 0. This handle may only be redirected under the presence of **MiNT**. Doing so redirects output of the **BIOS**.
GSH_MIDIIN **GSH_MIDIOUT**	-4 -5	None	**GEMDOS** handles -4 and -5 refer to **MIDI** input and output respectively. Redirecting these handles will affect **BIOS** handle 3. These special handles exist only with the presence of **MiNT**.

BINDING

```
move.w      nhandle,-(sp)
move.w      shandle,-(sp)
move.w      #$46,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Fforce()** returns **E_OK** (0) if no error occurred or **EIHNDL** (-37) if a bad handle is given.

CAVEATS Prior to **GEMDOS** versions 0.15, handles forced to the printer would not work properly.

COMMENTS This function is often used to redirect the input or output of a child process. It should be used in conjunction with **Fdup()** to restore the standard handle before process termination. In addition, you should be aware that any file handle redirected to a standard handle ('con:' for example) will be closed when the child exits and should not be closed by the parent.

Standard **GEMDOS** file handles which have been redirected will revert to their original mapping upon **Fclose()**.

SEE ALSO **Fdup()**

Fgetchar()

LONG Fgetchar(***handle, mode*** **)**
WORD ***handle, mode*****;**

Fgetchar() reads a character from the specified handle.

OPCODE 263 (0x107)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *handle* is a valid **GEMDOS** handle to read from. If *handle* is a TTY then *mode* (a bit mask) has meaning as follows:

Name	*mode*	Meaning
TTY_COOKED	0x01	Cooked mode. Special control characters such as CTRL-C and CTRL-Z are checked and acted upon. In addition, flow control with CTRL-S and CTRL-Q is activated.
TTY_ECHO	0x02	Echo mode. Characters read are echoed back to the TTY.

BINDING

```
move.w      mode,-(sp)
move.w      handle,-(sp)
move.w      #$107,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Fgetchar()** returns the character read in the low byte of the returned **LONG**. If the device is a terminal where scan codes are available, the **LONG** will be mapped in the same manner as **Bconin()**. If an end-of-file is reached, the value 0xFF1A will be returned.

SEE ALSO **Bconin(), Fputchar(), Fread()**

Fgetdta()

DTA *Fgetdta(VOID)

Fgetdta() returns current **DTA** (Disk Transfer Address)

OPCODE 47 (0x2F)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS None.

BINDING

```
move.w      #$2F,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Fgetdta()** returns a pointer to the current Disk Transfer Address. The structure **DTA** is defined as:

```
typedef struct
{
        BYTE          d_reserved[21];
        BYTE          d_attrib;
        UWORD     d_time;
```

```
        UWORD    d_date;
        LONG         d_length;
        char         d_fname[14];
} DTA;
```

COMMENTS When an application starts, its **DTA** overlaps the command line string in the processes' basepage. Any use of the **Fsfirst()** or **Fsnext()** call without first reallocating a new **DTA** will cause the processes' command line to be corrupted.

To prevent this, you should use **Fsetdta()** to define a new **DTA** structure for your process prior to using **Fsfirst()** or **Fsnext()**. Be careful to avoid assigning your **DTA** to a local or automatic variable without setting it to its original value before the variable goes out of scope.

SEE ALSO **Fsetdta(), Fsfirst(), Fsnext()**

Finstat()

LONG Finstat(*handle*)
WORD *handle*;

Finstat() determines the input status of a file.

OPCODE 261 (0x105)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *handle* specifies the **GEMDOS** file handle of the file to return information about.

BINDING

```
move.w      handle,-(sp)
move.w      #$105,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Finstat()** returns 0 or a positive number of characters waiting to be read if successful. A negative **GEMDOS** error code is returned otherwise.

CAVEATS Currently **Finstat()** always returns 0 for disk files.

SEE ALSO **Cauxis(), Cconis(), Fcntl(), Foutstat()**

Flink()

LONG Flink(*oldname, newname*)
char **oldname, *newname*;

Flink() creates a new name for the specified file. After the call the file may be referred to by either name. An **Fdelete()** call on one filename will not affect the other.

OPCODE 301 (0x12D)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *oldname* points to the **GEMDOS** path specification of the currently existing file and *newname* specifies the name of the alias to create.

BINDING

```
pea         newname
pea         oldname
move.w      #$12D,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE **Flink()** returns a 0 if successful or a negative **GEMDOS** error code otherwise.

CAVEATS Not all file systems support 'hard links'.

COMMENTS The filenames given must reside on the same physical device.

SEE ALSO **Frename(), Fsymlink()**

Flock()

LONG Flock(*handle, mode, start, length*)
WORD *handle,mode*;
LONG *start,length*;

Flock() sets or removes a lock on a portion of a file which prevents other processes from accessing it.

OPCODE 92 ($5C)

AVAILABILITY Only present when '**_FLK**' cookie exists.

PARAMETERS *handle* specifies the **GEMDOS** handle of the file. *mode* is **FLK_LOCK** (0) to create a lock and **FLK_UNLOCK** (1) to remove it. *start* specifies the byte offset from the beginning of the file which indicates where the lock starts. *length* specifies the length of the lock in bytes.

BINDING

```
move.l      length,-(sp)
move.l      start,-(sp)
move.w      mode,-(sp)
move.w      handle,-(sp)
trap        #1
lea         12(sp),sp
```

RETURN VALUE **Flock()** returns **E_OK** (0) if the call was successful, **ELOCKED** (-58) if an overlapping section of the file was already locked, **ENSLOCK** (-59) if a matching lock was not found for removal, or another **GEMDOS** error code as appropriate.

COMMENTS To remove a lock, you must specify identical *start* and *length* parameters as you originally set.

MiNT allows locks to be set on devices by locking their entry in 'U:\DEV\' as shown in the example below:

```
handle = Fopen( "U:\DEV\MODEM1", 3 );
if( handle < 0)
      return ERR_CODE;      /* Unable to open. */

retcode = Flock( (WORD)handle, 0, 0, 0 );        /* Lock
*/
if( retcode != E_OK )
      return FILE_IN_USE;   /* File is already locked */

/*
 * Now do device input/output.
 */

Flock( (WORD)handle, 1, 0, 0 );     /* Unlock */
Fclose( (WORD)handle );
```

SEE ALSO **Fopen(), Fwrite(), Fread()**

Fmidipipe()

LONG Fmidipipe(***pid, in, out*** **)**
WORD ***pid, in, out*****;**

Fmidipipe() is used to change the file handles used for MIDI input and output.

OPCODE 294 (0x126)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *pid* is the process id of the process whose MIDI devices you wish to alter. If *pid* is 0, then the current process will be modified. *in* specifies the **GEMDOS** file handle of the device to handle MIDI input. *out* specifies the **GEMDOS** file handle of the device to handle MIDI output.

BINDING

```
move.w      out,-(sp)
move.w      in,-(sp)
move.w      pid,-(sp)
move.w      #$126,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Fmidipipe()** returns a 0 if successful or a negative **GEMDOS** error code otherwise.

COMMENTS An **Fmidipipe(** 0, *in*, *out* **)** call is essentially the same as:

```
Fforce( -4, in);
Fforce( -5, out);
```

After this call, any **Bconin()** calls to MIDI device 5 will translate to a one character read from handle *in*. Likewise any **Bconout()** calls to MIDI device 5 will translate to a one character write to handle *out*.

SEE ALSO **Fdup(), Fforce()**

Fopen()

LONG Fopen(***fname*, *mode*** **)**
char ****fname*;**
WORD ***mode***;

Fopen() opens the **GEMDOS** file specified.

OPCODE 61 ($3D)

AVAILABILITY All **GEMDOS** versions. *mode* bits pertaining to file sharing/record locking are only valid when the '**_FLK**' cookie is present.

PARAMETERS *fname* is the **GEMDOS** file specification of the file to be opened. *mode* specifies the mode the file is to be placed into once opened. *mode* is a bit array which may be formed by using the bit masks given as follows:

Bit 7	Bits 6-4	Bit 3	Bits 2-0
Inheritance flag	Sharing mode	Reserved	Access code

Bits 0-2 specify the file access code as follows:

Bit 2	Bit 1	Bit 0	File Access Codes
0	0	0	Read only access (**S_READ**)
0	0	1	Write only access (**S_WRITE**)
0	1	0	Read/Write access (**S_READWRITE**)

Bit 3 is reserved and should always be 0. Bits 4-6 specify the file sharing mode of the file to be opened as follows:

Bit 6	Bit 5	Bit 4	File Sharing Codes
0	0	0	Compatibility Mode (**S_COMPAT**). If the file's read-only bit is set, then this is the same as Deny Writes, otherwise it is the same as Deny Read/Writes.
0	0	1	Deny Read/Writes (**S_DENYREADWRITE**)
0	1	0	Deny Writes (**S_DENYWRITE**)
0	1	1	Deny Reads (**S_DENYREAD**)
1	0	0	Deny None (**S_DENYNONE**)

Bit 7 (**S_INHERIT**) is the file's inheritance flag. If this flag is not set, a child process will inherit any open file handles and has the same access as the parent. If this flag is set, a child must re-open any files it wishes to use and must face the same sharing restrictions other processes must share.

BINDING

```
move.w      mode,-(sp)
pea         fname
move.w      #$3D,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE

Upon return, if the longword is positive, the lower **WORD** contains the new handle of the open file, otherwise the negative **LONG** should be regarded as a **GEMDOS** error code.

COMMENTS

Bits 7-3 of *mode* should be set to 0 unless the '**_FLK**' cookie is present indicating the presence of the file sharing/record locking extensions to **GEMDOS**.

SEE ALSO

Fclose(), Fcreate()

Foutstat()

LONG Foutstat(*handle*)
WORD *handle*;

Foutstat() determines the output status of a file.

OPCODE 262 (0x106)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *handle* specifies the **GEMDOS** file handle of the file to return information about.

BINDING

```
move.w      handle,-(sp)
move.w      #$106,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Foutstat()** returns a 0 or positive number indicating the number of characters which may be written to the specified file without blocking. If an error occurred, **Foutstat()** returns a negative **GEMDOS** error code.

CAVEATS Currently this function always returns 1 for disk files.

SEE ALSO **Cconos(), Cauxos(), Cprnos(), Fcntl(), Finstat()**

Fpipe()

LONG Fpipe(*fhandle*)
WORD *fhandle[2]*;

Fpipe() creates a pipe named 'SYS$PIPE.xxx' (where 'xxx' is a three digit integer) on 'U:\PIPE\' and returns two file handles to it, one for reading and one for writing.

OPCODE 256 (0x100)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *fhandle* is a pointer to an array of two **WORD**s. If the functions is successful, *fhandle[0]* will contain an open **GEMDOS** file handle to the pipe which may be used for reading only. *fhandle[1]* will contain an open **GEMDOS** file handle to the pipe which may be used for writing only.

BINDING

```
pea         fhandle
move.w      #$100,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE

Fpipe() returns **E_OK** (0) if successful or a negative **GEMDOS** error code otherwise.

CAVEATS

No more than 999 pipes created with **Fpipe()** may be in use at once.

COMMENTS

This function is normally used by shells who wish to redirect the input and output of their child processes. Prior to lauching a child process, the shell redirects its input and output (as necessary) to the read and write ends of the newly created pipe.

Fputchar()

LONG Fputchar(***handle, lchar, mode*** **)**
WORD ***handle*****;**
LONG ***lchar*****;**
WORD ***mode*****;**

Fputchar() writes a character to the specified file.

OPCODE

264 (0x108)

AVAILABILITY

This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS

handle specifies the handle of the file to write a character to.

If the file specified by *handle* is a pseudo-terminal then all four bytes of *lchar* are written (it should be formatted as a character read from **Bconin()**), otherwise only the low byte of *lchar* is transmitted.

mode is only valid if *handle* refers to a terminal device. If *mode* is **TTY_COOKED** (0x0001) then control characters (which could cause **SIGINT** or **SIGTSTP** signals to be raised) passed through this function will be interpreted and acted upon. Setting *mode* to 0 will cause control characters to have no special effect.

BINDING

```
move.w      mode,-(sp)
move.l      lchar,-(sp)
move.w      handle,-(sp)
move.w      #$108,-(sp)
trap        #1
```

```
lea         10(sp),sp
```

RETURN VALUE **Fputchar()** returns 4L if the character was output to a terminal, 1L if the character was output to a non-terminal, 0L if the character could not be written (possibly because of flow control), **EIHNDL** (-37) if the handle was invalid, or a negative **BIOS** error code if an error occurred during I/O.

SEE ALSO **Cconout(), Cauxout(), Crawio(), Cprnout(), Bconout(), Fgetchar(), Fwrite()**

Fread()

LONG Fread(***handle, length, buf*** **)**
WORD ***handle*****;**
LONG ***length*****;**
VOIDP ***buf*****;**

Fread() reads binary data from a specified file from the current file pointer.

OPCODE 63 (0x3F)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *handle* is the **GEMDOS** file handle of the file to read from. *length* specifies the number of bytes of data to read. *buf* is a pointer to a buffer (at least *length* bytes long) where the read data will be stored.

BINDING

```
pea         buf
move.l      length,-(sp)
move.w      handle,-(sp)
move.w      #$3F,-(sp)
trap        #1
lea         12(sp),sp
```

RETURN VALUE **Fread()** returns either a positive amount indicating the number of bytes actually read (this number may be smaller than length if an **EOF** is hit) or a negative **GEMDOS** error code.

CAVEATS **Fread()** will crash the system if given a parameter of 0 for *length* on **GEMDOS** versions lower than 0.15.

SEE ALSO **Fwrite(), Fopen(), Fclose()**

Freadlink()

LONG Freadlink(*bufsiz, buf, name*)
WORD *bufsiz*;
char **buf, *name*;

Freadlink() determines what file the specified symbolic link refers to.

OPCODE 303 (0x12F)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *bufsiz* specifies the length of buffer *buf* into which the original file pointed to by the symbolic link *name* is written.

BINDING

```
pea         name
pea         buf
move.w      bufsiz,-(sp)
move.w      #$12F,-(sp)
trap        #1
lea         12(sp),sp
```

RETURN VALUE **Freadlink()** returns 0 if successful or a negative **GEMDOS** error code otherwise.

SEE ALSO **Fsymlink()**

Frename()

LONG Frename(*reserved, oldname, newname*)
WORD *reserved*;
char **oldname,*newname*;

Frename() renames a standard **GEMDOS** file. It may also be used to move a file in the tree structure of a physical drive.

OPCODE 86 (0x56)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *reserved* is not currently used and should be 0. *oldname* is the **GEMDOS** file specification of the file's current name/location. *newname* is the **GEMDOS** file specification of the new name/location of the file.

BINDING

```
pea         newname
```

```
pea         oldname
move.w      #0,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE **Frename()** returns **E_OK** (0) if the operation was successful or a negative **GEMDOS** error code if not.

CAVEATS Prior to **GEMDOS** version 0.15, this command may not be used to rename folders. Also, do not attempt to rename a file that is currently open under any version of **GEMDOS**.

Fseek()

LONG Fseek(***offset, handle, mode*** **)**
LONG ***offset*****;**
WORD ***handle,mode*****;**

Fseek() moves the file position pointer within a **GEMDOS** file.

OPCODE 66 (0x42)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *handle* specifies the **GEMDOS** file handle of the file pointer to modify. The meaning of *offset* varies with *mode* as follows:

Name	*mode*	Meaning
SEEK_SET	0	*offset* specifies the positive number of bytes from the beginning of the file.
SEEK_CUR	1	*offset* specifies the negative or positive number of bytes from the current file position.
SEEK_END	2	*offset* specifies the positive number of bytes from the end of the file.

BINDING

```
move.w      mode,-(sp)
move.w      handle,-(sp)
move.l      offset,-(sp)
move.w      #$42,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE **Fseek()** returns a positive value representing the new absolute location of the file pointer from the beginning of the file or a negative **GEMDOS** error code.

Fselect()

WORD Fselect(*timeout, rfds, wfds, reserved*)
WORD *timeout*;
LONG **rfds, *wfds*;
LONG *reserved*;

Fselect() enumerates file descriptors which are ready for reading and/or writing.

OPCODE 285 (0x11D)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *timeout* specifies the maximum amount of time (in milliseconds) to wait for at least one of the specified file descriptors to become unblocked. If *timeout* is 0 then the process will wait indefinitely.

rfds and *wfds* each point to a **LONG** bitmap describing the read and write file descriptors to wait for. Setting bit #10 of the **LONG** pointed to by *rfds*, for example, will cause **Fselect()** to return when **GEMDOS** handle 10 is available for reading.

As many read or write file descriptors can be specified per call as desired. Specifying **NULL** for either *rfds* or *wfds* is the same as passing a pointer to a **LONG** with no bits set.

Upon return the **LONG**s pointed to by *rfds* and *wfds* will be filled in with a similar bitmap indicating which handles are ready to be read/written. *reserved* should always be set to 0L.

BINDING

```
move.l      reserved,-(sp)
pea         wfds
pea         rfds
move.w      timeout,-(sp)
move.w      #$11D,-(sp)
trap        #1
lea         16(sp),sp
```

RETURN VALUE **Fselect()** returns the sum of bits set in both *rfds* and *wfds*. A return value of 0 indicates that the function timed out before any of the specified file handles became available. A negative **GEMDOS** error code is returned if the function failed.

CAVEATS **Fselect()** does not currently work on any **BIOS** device except the keyboard.

COMMENTS **Fselect(** 0L, 0L, 0L, 0L**)** will block the calling process forever.

SEE ALSO **Finstat(), Foutstat()**

Fsetdta()

VOID Fsetdta(***ndta*** **)**
DTA ****ndta***;**

Fsetdta() sets the location of a new **DTA** (Disk Transfer Address) in memory.

OPCODE 26 (0x1A)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *ndta* is a pointer to a valid memory area which will be used as the new **DTA**. The **DTA** structure is defined under the entry for **Fgetdta()**.

BINDING

```
pea         ndta
move.w      #$1A,-(sp)
trap        #1
addq.l      #6,sp
```

COMMENTS When an application starts, its **DTA** overlaps the command line string in the processes' basepage. Any use of the **Fsfirst()** or **Fsnext()** call without first reallocating a new **DTA** will cause the processes' command line to be corrupted.

To prevent this, you should use **Fsetdta()** to define a new **DTA** structure for your process prior to using **Fsfirst()** or **Fsnext()**. Be careful to avoid assigning your **DTA** to a local or automatic variable without setting it to its original value before the variable goes out of scope.

SEE ALSO **Fgetdta(), Fsfirst(), Fsnext()**

Fsfirst()

WORD Fsfirst(***fspec*****,** ***attribs*** **)**
char ****fspec***;**
WORD ***attribs*****;**

Fsfirst() searches the file/pathspec given for the first occurrence of a file or subdirectory with named attributes and if found, fill in the current **DTA** with that file's information.

OPCODE 78 (0x4E)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *fspec* is the **GEMDOS** file specification of the file or subdirectory to search for. This specification may use wildcard characters (? or *) within the filename, however they may not be used within the pathname. This function is the only **GEMDOS** function which accepts wildcard characters in the path specification.

attribs is a bit mask which can combine several file characteristics that further narrows the search as follows:

Name	Bit Mask	Meaning
FA_READONLY	0x01	Include files which are read-only.
FA_HIDDEN	0x02	Include hidden files.
FA_SYSTEM	0x04	Include system files.
FA_VOLUME	0x08	Include volume labels.
FA_DIR	0x10	Include subdirectories.
FA_ARCHIVE	0x20	Include files with archive bit set.

BINDING

```
move.w      attribs,-(sp)
pea         fspec
move.w      #$4E,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Fsfirst()** returns **E_OK** (0) if a file was found and the **DTA** was successfully filled in with the file information. Otherwise, it returns a negative **GEMDOS** error code.

The **DTA** structure is defined as:

```
typedef struct
{
      BYTE    d_reserved[21];
      BYTE    d_attrib;
      UWORD   d_time;
      UWORD   d_date;
      LONG    d_length;
      char    d_fname[14];
} DTA;
```

COMMENTS This function uses the application's **DTA** which is initially located in the same memory location as the processes' command line. Using this function without first assigning a new **DTA** will corrupt the command line.

When running in the **MiNT** domain (see **Pdomain()**), **Fsfirst()** and **Fsnext()** will fill in the **DTA** with lowercase filenames rather than the standard **TOS** uppercase.

SEE ALSO **Fsnext(), Fgetdta(), Fsetdta()**

Fsnext()

WORD Fsnext(VOID)

Fsnext() should be called as many times as necessary after a corresponding **Fsfirst()** call to reveal all files which match the search criteria.

OPCODE 79 (0x4F)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$4F,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Fsnext()** returns **E_OK** (0) if another file matching the search criteria given in **Fsfirst()** is found and the **DTA** has been properly filled in with the file's information. Otherwise, a negative **GEMDOS** error code is returned.

COMMENTS This function uses the application's **DTA** which is initially located in the same memory location as the processes' command line. Using this function without first assigning a new **DTA** will corrupt the command line.

This call should only be used after **Fsfirst()** and the contents of the **DTA** should not be modifed between the calls.

SEE ALSO **Fsfirst()**

Fsymlink()

LONG Fsymlink(*oldname, newname*)
char ****oldname, *newname*;**

Fsymlink() creates a symbolic link to a file.

OPCODE 302 (0x12E)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *oldname* points to the file specification of the file to create a link to. *newname*

points to the file specification of the link to create.

BINDING

```
pea         newname
pea         oldname
move.w      #$12E,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE **Fsymlink()** returns 0 if successful or a negative **GEMDOS** error code otherwise.

COMMENTS **Fsymlink()**, unlike **Flink()**, creates symbolic links, which, unlike hard links, can be setup between physical devices and file systems.

An **Fdelete()** call to a symbolic link will delete the link, not the file. A call to **Fdelete()** on the original file will cause future references to the created symbolic link to fail.

SEE ALSO **Flink(), Freadlink()**

Fwrite()

LONG Fwrite(***handle, count, buf*** **)**
WORD ***handle*****;**
LONG ***count*****;**
VOIDP ***buf*****;**

Fwrite() writes the contents of a buffer to the specified **GEMDOS** file.

OPCODE 64 (0x40)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *handle* is the handle of the file to write to. *count* specifies the number of bytes to write. *buf* indicates the starting address of the data to write.

BINDING

```
pea         buf
move.l      count,-(sp)
move.w      handle,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE **Fwrite()** returns the positive number of bytes actually written or a negative **GEMDOS** error code if the operation failed.

CAVEATS Prior to **GEMDOS** version 0.15, calling **Fwrite()** with a *count* parameter of 0 will hang the system.

SEE ALSO **Fread()**

Fxattr()

LONG Fxattr(*flag, name, xattr*)
WORD *flag*;
char **name*;
XATTR **xattr*;

Fxattr() returns extended information about the specified file.

OPCODE 300 (0x12C)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *flag* specifies whether attributes returned by this call on symbolic links should be those of the file to which the link points or the link itself. A value of **FX_FILE** (0) causes the attributes to be those of the actual file whereas a value of **FX_LINK** (1) returns the attributes of the link itself.

name specifies the name of the file from which attributes are to be read and placed in the **XATTR** structure pointed to by *xattr*. **XATTR** is defined as follows:

```
typedef struct
{
      UWORD mode;
      LONG index;
      UWORD dev;
      UWORD reserved1;
      UWORD nlink;
      UWORD uid;
      UWORD gid;
      LONG size;
      LONG blksize;
      LONG nblocks;
      WORD mtime;
      WORD mdate;
      WORD atime;
      WORD adate;
      WORD ctime;
      WORD cdate;
      WORD attr;
      WORD reserved2;
      LONG reserved3;
      LONG reserved4;
} XATTR;
```

XATTR's members have the following meaning:

XATTR Element	Meaning
mode	Masking *mode* with 0xF000 reveals the file type as one of the following: **S_IFCHR** (0x2000) **S_IFDIR** (0x4000) **S_IFREG** (0x8000) **S_IFIFO** (0xA000) **S_IMEM** (0xC000) **S_IFLNK** (0xE000) The lower three nibbles of *mode* is a bit mask which specifies the legal file access mode(s) as defined in **Fchmod()**.
index	This member combined with the *dev* field are designed to provide a unique identifier for a file under file systems which allow multiple files with the same filename.
dev	This value represents either a **BIOS** device number or an identifier created by the file system to represent a remote device.
reserved1	This structure element is currently reserved for future implementations of **MiNT**.
nlink	This value specifies the current number of hard links attached to the file. On a file system that does not support hard links and for most regular files, *nlink* is 1.
uid	*uid* is the user ID of the owner of the file.
gid	*gid* is the group ID of the owner of the file.
size	*size* is the length of the file in bytes.
blksize	*blksize* specifies the size of blocks (in bytes) in this file system.
nblocks	*nblocks* is the actual number of blocks the file is using on the device. This number may include data storage elements other used to keep track of the file (aside from the actual data).
mtime, mdate	Time and date of the last file modification in **GEMDOS** format.
atime, adate	Time and date of the last file access in **GEMDOS** format.
ctime, cdate	Time and date of the file's creation in **GEMDOS** format.
attr	Standard file attributes (same as read by **Fattrib()**).
reserved2	This structure element is currently reserved for future implementations of **MiNT**.
reserved3	This structure element is currently reserved for future implementations of **MiNT**.
reserved4	This structure element is currently reserved for future implementations of **MiNT**.

BINDING

```
pea         xattr
pea         name
move.w      flag,-(sp)
move.w      #$12C,-(sp)
trap        #1
lea         12(sp),sp
```

RETURN VALUE **Fxattr()** returns 0 if successful or a negative **GEMDOS** error code otherwise.

SEE ALSO **Fattrib()**

Maddalt()

LONG Maddalt(***start, size*** **)**
VOIDP ***start*****;**
LONG ***size*****;**

Maddalt() informs **GEMDOS** of the existence of additional 'alternative' RAM that would not normally have been identified by the system.

OPCODE 20 (0x14)

AVAILABILITY Available as of **GEMDOS** version 0.19 only.

PARAMETERS *start* indicates the starting address for the block of memory to be added to the **GEMDOS** free list. *size* indicates the length of this block in bytes.

BINDING

```
move.l      size,-(sp)
pea         start
move.w      #$14,-(sp)
trap        #1
lea         10(sp),sp
```

RETURN VALUE **Maddalt()** returns **E_OK** (0) if the call succeeds or a negative **GEMDOS** error code otherwise.

COMMENTS This call should only be used to identify RAM not normally identified by the **BIOS** at startup (added through a VME-card or hardware modification). Once this RAM has been identified to the system it may not be removed and should only be allocated and used via the standard system calls. In addition, programs wishing to use this RAM must have their alternative RAM load bit set or use **Mxalloc()** to specifically request alternative RAM.

See the discussion earlier in this chapter for more information about the types of available RAM.

SEE ALSO **Mxalloc()**

Malloc()

VOIDP Malloc(***amount*** **)**
LONG ***amount*****;**

Malloc() requests a block of memory for use by an application.

OPCODE 72 (0x48)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *amount* specifies the amount of memory (in bytes) you wish to allocate. You may pass a value of -1L in which case the function will return the size of the largest free block of memory.

BINDING

```
move.l      amount,-(sp)
move.w      #$48,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Malloc()** returns **NULL** if there is no block large enough to fill the request or a pointer to the block if the request was satisfied. The memory allocated will be chosen based on the status of the processes' load flags. To specify the memory requirements in more detail, use **Mxalloc()**.

CAVEATS Prior to **GEMDOS** version 0.15, **Malloc(** 0L **)** will return a pointer to invalid memory as opposed to failing as it should.

COMMENTS Because **GEMDOS** can only allocate a limited amount of blocks per process (as few as 20 depending on the version of **GEMDOS**), applications should limit their usage of this call by allocating a few large blocks instead of many small blocks or use a 'C' memory manager (like **malloc()**) if possible.

SEE ALSO **Mxalloc()**

Mfree()

WORD Mfree(***startadr*** **)**
VOIDP ***startadr*****;**

Mfree() releases a block of memory previously reserved with **Malloc()** or **Mxalloc()** back into the **GEMDOS** free list.

OPCODE 73 (0x49)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *startadr* is the starting address of the block to be freed. This address must be the same as that returned by the corresponding **Malloc()** or **Mxalloc()** call.

BINDING

```
pea         startadr
move.w      #$49,-(sp)
trap        #1
addq.       #6,sp
```

RETURN VALUE **Mfree()** returns **E_OK** (0) if the block was freed successfully or a negative **GEMDOS** error code otherwise.

SEE ALSO **Malloc(), Mxalloc()**

Mshrink()

WORD Mshrink(*startadr, newsize*)
VOIDP *startadr*;
LONG *newsize*;

Mshrink() releases a portion of a block's memory to the **GEMDOS** free list.

OPCODE 74 (0x4A)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *startadr* is the address of the block whose size you wish to decrease. *newsize* is the length you now desire for the block.

BINDING

```
move.l      newsize,-(sp)
pea         startadr
clr.w       -(sp)       // Required/Reserved Value
move.w      #$4A,-(sp)
trap        #1
lea         12(sp),sp
```

RETURN VALUE **Mshrink()** returns **E_OK** (0) if the operation was successful or a negative **GEMDOS** error code otherwise.

CAVEATS This call should be used only to 'shrink' a memory block, not to enlarge it.

SEE ALSO **Malloc(), Mxalloc(), Mfree()**

Mxalloc()

VOIDP Mxalloc(***amount, mode*** **)**
LONG ***amount*****;**
WORD ***mode*****;**

Mxalloc() allocates a block of memory according to specified preferences.

OPCODE 68 (0x44)

AVAILABILITY Available from **GEMDOS** version 0.19.

PARAMETERS *amount* specifies the length (in bytes) of the block requested. As with **Malloc()**, specifying -1L for *amount* will return the size of the largest block of memory available. With modes 0 or 1, the size of the largest block of available RAM from the specified type of RAM is returned. Modes 2 and 3 return the size of the largest available block or whichever type of RAM had the largest block.

mode is a **WORD** bit array which specifies the type of memory requested as follows:

Bit	Meaning
0-1	Bits 0-1 represent a possible value of 0-3 representing the type of RAM to allocate as follows: **Name** / **Value** / **Meaning** **MX_STRAM** 0 Allocate only ST-RAM **MX_TTRAM** 1 Allocate only TT-RAM **MX_PREFSTRAM** 2 Allocate either, preferring ST-RAM **MX_PREFTTRAM** 3 Allocate either, preferring TT-RAM
2	Not used (should be set to 0).
3	If set, refer to bits 4-7 for memory protection advice, otherwise default to protection specified in program header. This bit is only valid in the presence of **MiNT**.
4-7	Bits 4-7 represent a possible value of 0-7 representing the memory protection mode to place on the allocated block of memory. Currently valid values are: **Name** / **Value** / **Meaning** **MX_HEADER** 0 Refer to Program Header **MX_PRIVATE** 1 Private **MX_GLOBAL** 2 Global **MX_SUPERVISOR** 3 Supervisor Mode Only Access **MX_READABLE** 4 Read Only Access These bits are only consulted if bit 3 is set and **MiNT** is present.
8-15	Not used (should be set to 0).

BINDING

```
move.w      mode,-(sp)
move.l      amount,-(sp)
move.w      #$44,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Mxalloc()** returns **NULL** if the request could not be granted or a valid pointer to the start of the block allocated otherwise.

COMMENTS **Mxalloc()** should be used instead of **Malloc()** whenever it is available.

SEE ALSO **Malloc(), Mfree()**

Pause()

VOID Pause(VOID)

Pause() suspends the process until a signal is received.

OPCODE 289 (0x121)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$121,-(sp)
trap        #1
addq.l      #2,sp
```

COMMENTS If the signal handler does a 'C' **longjmp()** to a different point in the process or if the handler's purpose is to exit the process, this call will never return.

SEE ALSO **Psigblock(), Psignal(), Psigsetmask()**

Pdomain()

WORD Pdomain(*domain*)
WORD *domain*;

Pdomain() determines/modifies the calling processes' execution domain.

OPCODE 281 (0x119)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *domain* contains the domain code of the new process domain. Currently the only

valid values are **DOMAIN_TOS** (0) for the **TOS** compatibility domain and **DOMAIN_MINT** (1) for the **MiNT** domain. Passing a negative value for *domain* will not change domains but it will return the current domain.

BINDING

```
move.w      domain,-(sp)
move.w      #$119,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Pdomain()** returns the domain in effect prior to the call.

COMMENTS Process domain affects system calls like **Fread()**, **Fwrite()**, **Fsfirst()**, and **Fsnext()**. Processes behave as expected when under the **TOS** domain.

When processes run under the **MiNT** domain, however, the behavior of **Fread**() and **Fwrite()** calls when dealing with terminals can be modified by **Fcntl()**. Also, **Fsfirst()** and **Fsnext()** may not necessarily return the standard **DOS** 8 + 3 file name format. **MiNT** domain processes must understand filenames formatted for different file systems.

SEE ALSO **Fcntl()**

Pexec()

LONG Pexec(*mode, fname, cmdline, envstr*)
WORD *mode*;
char **fname,*cmdline,*envstr*;

Pexec() has many functions designed to spawn child processes depending on the selected mode.

OPCODE 75 (0x4B)

AVAILABILITY **Pexec()** modes 0, 3, 4, and 5, are available in all **GEMDOS** versions. Mode 6 is available as of **GEMDOS** version 0.15. Mode 6 is available as of **GEMDOS** version 0.19. Modes 100, 104, 106, and 200 are only available in the presence of **MiNT**.

PARAMETERS *mode* defines the function of **Pexec()** and the meaning of its parameters and return value as defined below. For modes which load a program, *fname* specifies the **GEMDOS** file specification of the file to load. *cmdline* is pointer to a string containg the command line which will be passed to the calling program. The first byte of the string should indicate the length of the command line (maximum of 125 bytes). The actual command line starts at byte 2. *envstr* is a pointer to an environment which is copied and assigned to the child process. If *envstr* is **NULL**,

the child inherits a copy of the parent's environment.

Name	*mode*	Meaning
PE_LOADGO	0	**'LOAD AND GO'** - Load and execute named program file and return a **WORD** exit code when the child terminates.
PE_LOAD	3	**'LOAD, DON'T GO'** - Load named program. If successful, the **LONG** return value is the starting address of the child processes' basepage. The parent owns the memory of the child's environment and basepage and must therefore free them when completed with the child.
PE_GO	4	**'JUST GO'** - Execute process with basepage at specified address. With this mode, *fname* and *envstr* are **NULL**. The starting address of the basepage of the process to execute is given in the *cmdline* parameter.
PE_BASEPAGE	5	**'CREATE BASEPAGE'** - This mode allocates the largest block of free memory and creates a basepage in the first 256 bytes of it. *fname* should be set to **NULL**. It is the responsibility of the parent to load or define the child's code, shrink the memory block as necessary, and initialize the basepage pointers to the TEXT, DATA, and BSS segments of the program. With **MiNT**, use of this mode in conjunction with mode **PE_CGO** can be used to emulate the **Pvfork()** call without blocking the parent.
PE_GOTHENFREE	6	**'JUST GO, THEN FREE'** - This mode is identical to mode **PE_GO** except that memory ownership of the child's environment and basepage belong to the child rather than the parent so that when the child **Pterm()**'s, that memory is automatically freed.
PE_CLOADGO	100	**'LOAD, GO, DON'T WAIT'** - This mode is identical to mode **PE_LOADGO** except that the parent process is returned to immediately while the child continues to execute. The positive process ID of the child is returned. Environment and basepage memory blocks are freed automatically when the child **Pterm()**'s
PE_CGO	104	**'JUST GO, DON'T WAIT'** - This mode is similar to mode **PE_GO** except that the parent process is returned to immediately while the child continues to execute concurrently. The positive process ID of the child is returned. Memory ownership of the environment and basepage are shared by the parent and child (this sharing extends to all memory owned by the parent). *fname* may be used to supply a name for the child, otherwise, if **NULL** is used, the name of the parent will be used. *cmdline* should point to the process basepage. *envstr* should be **NULL**.
PE_NOSHARE	106	**'JUST GO, DON'T WAIT, NO SHARING'** - This mode is exactly the same as mode **PE_CGO** except that the child process owns its own environment and basepage sharing no memory with the parent.

PE_REPLACE	200	**'REPLACE PROGRAM AND GO'** - This mode works like mode **PE_CLOADGO** except that the parent process is terminated immediately and the child process completely replaces the parent in memory retaining the same process ID. *fname*, *cmdline*, and *envstr*, are all normally passed and valid.

BINDING

```
pea         envstr
pea         cmdline
pea         fname
move.w      word,-(sp)
move.w      #$4B,-(sp)
trap        #1
lea         16(sp),sp
```

RETURN VALUE The value returned by **Pexec()** is dependent on the *mode* value and is therefore explained above. All **Pexec()** modes return a **LONG** negative **GEMDOS** error code when the call fails. A **WORD** negative value indicates the child was successfully run but it terminated returning a negative error code. In all cases, a process returning after having been interrupted with CTRL-C returns 0x0000FFE0 (-32).

COMMENTS Command lines longer than 126 bytes may be passed to processes aware of the **Atari Extended Command Line Specification** (see discussion earlier in this chapter).

SEE ALSO **shel_write()**

Pfork()

WORD Pfork(VOID)

Pfork() creates a copy of the current process.

OPCODE 283 (0x11B)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$11B,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Pfork()** returns the new process ID in the parent and a 0 in the child.

CAVEATS If the parent is in supervisor mode when this call is made, the child is started in user mode anyway.

COMMENTS After a **Pfork()** call, two instances of one process will exist in memory. Program execution in both processes continue at the same point in the TEXT segment following this call. The parent's DATA and BSS segments are physically copied so that any variables that change in the child will not affect the parent and vice versa.

New processes started with this call should not call **Mshrink()** but are required to do any **GEM** initialization such as **appl_init()** and **v_opnvwk()** again (if **GEM** usage is needed). Both the parent and child use **Pterm()** or **Pterm0()** to terminate themselves.

SEE ALSO **Pexec(), Pvfork()**

Pgetegid()

WORD Pgetegid(VOID)

Pgetegid() returns the effective group ID of the process.

OPCODE 313 (0x139)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.95 exists.

BINDING

```
move.w      #$139,-(sp)
trap        #1
addq.l      #2,sp
```

COMMENTS The effective group ID of a process will be different than its actual group ID if its set gid bit is set. This mechanism allows users to grant file access to other users.

SEE ALSO **Pgetgid(), Pgeteuid()**

Pgeteuid()

WORD Pgeteuid(VOID)

Pgeteuid() returns the effective user ID of the process.

OPCODE 312 (0x138)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.95 exists.

BINDING

```
move.w      #$138,-(sp)
trap        #1
```

```
addq.l      #2,sp
```

COMMENTS The effective group ID of a process will be different than its actual group ID if its set gid bit is set. This mechanism allows users to grant file access to other users.

SEE ALSO **Pgetuid(), Pgetegid()**

Pgetgid()

WORD Pgetgid(VOID)

Pgetgid() returns the group ID (0-255) of the calling process.

OPCODE 271 (0x10F)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$10F,-(sp)
trap        #1
addq.l      #2,sp
```

SEE ALSO **Psetgid()**

Pgetpgrp()

WORD Pgetpgrp(VOID)

Pgetpgrp() returns the process group ID code for the calling process.

OPCODE 269 (0x10D)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$10D,-(sp)
trap        #1
addq.l      #2
```

COMMENTS Process groups are closely related processes which are used for job control and signaling purposes. Process groups usually terminate together rather than one at a time.

SEE ALSO **Psetpgrp(), Pkill()**

Pgetpid()

WORD Pgetpid(VOID)

Pgetpid() returns the positive **WORD** process ID code for the calling process. This identifer uniquely identifies the process within the system.

OPCODE 267 (0x10B)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$10B,-(sp)
trap        #1
addq.l      #2,sp
```

Pgetppid()

WORD Pgetppid(VOID)

Pgetppid() returns the process ID for the calling processes' parent.

OPCODE 268 (0x10C)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$10C,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Pgetppid()** returns the process ID code for the parent of the calling process or 0 if it was started by the kernel (not a child process).

Pgetuid()

WORD Pgetuid(VOID)

Pgetuid() returns the user ID code (0-255) of the calling process which determines access permissions and can be used in a multi-user system to differentiate users.

OPCODE 271 (0x10F)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$10F,-(sp)
trap        #1
addq.l      #2
```

SEE ALSO **Psetuid()**

Pkill()

WORD Pkill(*pid*, *sig*)
WORD *pid*, *sig*;

Pkill() sends a signal to one or more processes.

OPCODE 273 (0x111)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS **Pkill()** sends signal *sig* to certain processes based on the value of *pid*. If *pid* is positive, the signal is sent the the process with process identifier pid. If *pid* is 0, the signal is sent to all processes who belong to the same process group as the caller as well as the caller itself. If *pid* is negative, the signal is sent to all processes with process group number -*pid*.

BINDING

```
move.w      sig,-(sp)
move.w      pid,-(sp)
move.w      #$111,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Pkill()** returns 0 if successful or a negative **GEMDOS** error code otherwise.

COMMENTS If the caller is also a recipient of a signal and that signal causes program termination this call will never return.

SEE ALSO **Psignal()**

Pmsg()

WORD Pmsg(*mode, mboxid, msgptr*)
WORD *mode*;
LONG *mboxid*;
PMSG **msgptr*;

Pmsg() sends/receives a message to/from a 'message box'.

OPCODE 293 (0x125)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS *mode* specifies the action to take as follows:

Name	*mode*	Operation
MSG_READ	0	Block the process and don't return until a message is read from the specified mailbox ID *mboxid* and placed in the structure pointed to by *msgptr*.
MSG_WRITE	1	Block the process and don't return until a process waiting for a message with mailbox ID *mboxid* has received the message contained in the structure pointed to by *msgptr*.
MSG_READWRITE	2	Block the process until a process waiting for a message with mailbox ID *mboxid* has received the message contained in the structure pointed to by *msgptr* and a return message is received with mailbox ID 0xFFFFxxxx where 'xxxx' is the process ID of the current process.

PMSG is defined as:

```
typedef struct
{
       LONG userlong1;
       LONG userlong2;
       WORD pid;
} PMSG;
```

On return from writes, *pmsg.pid* contains the process ID of the process who read your message, on return from reads, its the process ID of the writer. The contents of *userlong1* and *userlong2* is completely up to the sender.

By OR'ing mode with **MSG_NOWAIT** (0x8000), you can prevent the call from blocking the process and simply return -1 if another process wasn't waiting to

read or send your process a message.

BINDING

```
pea         msgptr
move.l      mboxid,-(sp)
move.w      mode,-(sp)
move.w      #$125,-(sp)
trap        #1
lea         12(sp),sp
```

RETURN VALUE **Pmsg()** returns 0 if successful, -1 if bit 0x8000 is set and no process was ready to receive/send the desired message, or a negative **GEMDOS** error code.

Pnice()

WORD Pnice(*delta*)
WORD *delta*;

Pnice() alters the process priority of the calling process.

OPCODE 266 (0x10A)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *delta* is a signed number which is added to the current process priority value. Positive values decrease process priority while negative values increase it.

BINDING

```
move.w      delta,-(sp)
move.w      #$10A,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Pnice()** returns the prior process priority.

COMMENTS The process priority value has no fixed formula so it is hard to be able to predict the results of this call with any accuracy. This call is the same as **Prenice(Pgetpid()**, *delta* **)**.

SEE ALSO **Prenice()**

Prenice()

LONG Prenice(***pid, delta*** **)**
WORD ***pid, delta*****;**

Prenice() adjusts the process priority of the specified process.

OPCODE 295 (0x127)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.90 exists.

PARAMETERS The process priority for the process with process ID *pid* is adjusted by signed value *delta*. Positive values for *delta* decrease process priority while negative values increase it.

BINDING

```
move.w      delta,-(sp)
move.w      pid,-(sp)
move.w      #$127,-(sp)
trap        #1
addq.l      #6
```

RETURN VALUE **Prenice()** returns a 32-bit negative **GEMDOS** error code if unsuccessful. Otherwise, the lower 16-bit signed value can be interpreted as the previous process priority code.

COMMENTS The exact effect adjusting process priorites will have is difficult to determine.

SEE ALSO **Pnice()**

Prusage()

VOID Prusage(***rusg*** **)**
LONG ***rusg*****;**

Prusage() returns resource information about the current process.

OPCODE 286 (0x11E)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *rusg* is a pointer to an array of 8 **LONG**s as follows:

Name	*rusg[x]*	Meaning
PRU_KERNELTIME	0	Time spent by process in **MiNT** kernel.
PRU_PROCESSTIME	1	Time spent by process in its own code.
PRU_CHILDKERNALTIME	2	Total **MiNT** kernel time spent by children of this process.
PRU_CHILDPROCESSTIME	3	Total user code time spent by children of this process.
PRU_MEMORY	4	Total memory allocated by process (in bytes).
—	5-7	Reserved for future use.

BINDING

```
pea         rusg
move.w      #$11E,-(sp)
trap        #1
addq.l      #6,sp
```

COMMENTS All times given are in milliseconds.

SEE ALSO **Psetlimit()**

Psemaphore()

LONG Psemaphore(***mode, id, timeout*** **)**
WORD ***mode*****;**
LONG ***id*****;**
LONG ***timeout*****;**

Psemaphore() creates a semaphore which may only be accessed by one process at a time.

OPCODE 308 (0x134)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.92 exists.

PARAMETERS *mode* specifies the mode of the operation which affects the other two parameters as follows:

Name	*mode*	Meaning
SEM_CREATE	0	Create a semaphore with called *id* and grant ownership to the calling process. *timeout* is ignored.
SEM_DESTROY	1	Destroy the semaphore called *id*. This only succeeds if the semaphore is owned by the caller. *timeout* is ignored.

SEM_LOCK	2	Request ownership of semaphore *id*. The process will wait for the semaphore to become available for *timeout* milliseconds and then return. If *timeout* value of 0 will force the call to return immediately whether or not the semaphore is available. A *timeout* value of -1 will cause the call to wait indefinitely.
SEM_UNLOCK	3	Release ownership of semaphore *id*. The caller must be the current owner of the semaphore to release control. *timeout* is ignore.

BINDING

```
move.l      timeout,-(sp)
move.l      id,-(sp)
move.w      mode,-(sp)
move.w      #$134,-(sp)
trap        #1
lea         12(sp),sp
```

RETURN VALUE **Psemaphore()** returns a 0 if successful, **ERROR** (-1) if the process requested a semaphore it already owned, or a negative **GEMDOS** error code.

COMMENTS If your process is waiting for ownership of a semaphore and it is destroyed by another process, an **ERANGE** (-64) error will result. Any semaphores owned by a process when it terminates are released but not deleted.

Psetgid()

WORD Psetgid(*gid*)
WORD *gid*;

Psetgid() sets the group ID of the calling process.

OPCODE 277 (0x115)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *gid* is the group ID code to assign the calling process (0-255).

BINDING

```
move.w      gid,-(sp)
move.w      #$115,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Psetgid()** returns gid if successful or **EACCDN** (-36) if the process did not have the authority to change the group ID.

COMMENTS The group ID of a process may only be changed when it is currently 0. Therefore, once the group ID has been set, it is fixed and unchangeable. Further attempts to modify it will result in an **EACCDN** error.

SEE ALSO **Pgetgid()**

Psetlimit()

LONG Psetlimit(*limit, value*)
WORD *limit*;
LONG *value*;

Psetlimit() reads/modifies resource allocation limits for the calling process and all of its children.

OPCODE 287 (0x11F)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *limit* defines the resource to read or modify as follows:

Name	*limit*	Meaning
LIM_MAXTIME	1	Maximum CPU time in milliseconds. If *value* is positive, *value* determines the new maximum. If *value* is 0, then the limit is set at 'unlimited'. If *value* is negative, the current value is returned but not modified.
LIM_MAXMEM	2	Maximum total memory allowed for process. If *value* is positive, *value* determines the new maximum. If *value* is 0, then the limit is set at 'unlimited'. If *value* is negative, the current value is returned but not modified.
LIM_MAXMALLOC	3	Maximum total size of each Malloc (Mxalloc). If *value* is positive, *value* determines the new maximum. If *value* is 0, then the limit is set at 'unlimited'. If *value* is negative, the current value is returned but not modified.

BINDING

```
move.l      value,-(sp)
move.w      limit,-(sp)
move.w      #$11F,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Psetlimit()** returns the previous value or **ERANGE** (-64) if the value for *limit* was out of range.

COMMENTS The limits imposed by **Psetlimit()** are inherited from the parent by child processes.

SEE ALSO **Prusage()**

Psetpgrp()

LONG Psetpgrp(*pid, newgrp*)
WORD *pid, newgrp*;

Psetpgrp() sets the process group ID of the specified process.

OPCODE 270 (0x10E)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS The process group ID of the process with process ID *pid* will have its process group ID changed to *newgrp* if the calling process has the same user ID or is the parent of the specified process. If *pid* is 0, the process group ID of the current process is sent. If *newgrp* is 0, the process group ID is set to equal the processes' (not the callers' unless *pid* is also set to 0) process ID.

BINDING

```
move.w      newgrp,-(sp)
move.w      pid,-(sp)
move.w      #$10E,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Psetpgrp()** returns *newgrp* if successful or a negative **GEMDOS** error code otherwise.

SEE ALSO **Pgetpgrp()**

Psetuid()

WORD Psetuid(*uid*)
WORD *uid*;

Psetuid() sets the user ID of the calling process.

OPCODE 272 (0x110)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *uid* is the user ID to assign to the calling process.

BINDING

```
move.w      uid,-(sp)
move.w      #$110,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Psetuid()** returns *uid* if successful or a negative **GEMDOS** error code otherwise.

COMMENTS As with the process group ID, the user ID of a process may only be set if it is currently 0. This means that once the user ID is set, it may not be changed.

SEE ALSO **Pgetuid()**

Psigaction()

LONG Psigaction(*sig*, *act*, *oact*)
WORD *sig*;
SIGACTION **act*, **oact*;

Psigaction() specifies a default action for the specified signal.

OPCODE 311 (0x137)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.95 exists.

PARAMETERS *sig* specifies the signal whose action you wish to change. *act* points to a **SIGACTION** structure (as defined below) which defines the handling of future signals of type *sig*. *oact* points to a **SIGACTION** structure which defines the handling of pending signals of type *sig*.

```
typedef struct
{
       LONG sa_handler;
       WORD sa_mask;
       WORD sa_flags;
} SIGACTION;
```

Setting *sa_hander* to **SIG_DFL** (0) wll cause the default action to take place for the signal. A value of **SIG_IGN** (1) will cause the signal to be ignored. Any other value specifies the address of a signal handler.

The signal handler should expect one **LONG** argument on its stack which contains the signal number being delivered. During execution of the handler, all signals specified in *sa_mask* are blocked.

sa_flags is a signal-specific flag. When *sig* is **SIGCHLD**, setting Bit #0 (**SA_NOCLDSTOP**) will cause the **SIGCHLD** signal to be delivered only when the child process terminated (not when stopped).

BINDING

```
move.w      sig,-(sp)
pea         act
```

```
pea         oact
move.w      #$137,-(sp)
trap        #1
add.l       #12,sp
```

RETURN VALUE **Psigaction()** returns 0 if successful or a negative **GEMDOS** error code otherwise.

COMMENTS Calling **Psigaction()** automatically unmasks the specified signal for delivery.

SEE ALSO **Psignal**

Psigblock()

LONG Psigblock(*mask*)
LONG *mask*;

Psigblock() blocks selected signals from delivery.

OPCODE 278 (0x116)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *mask* is a bit mask of signals block. For each bit *n* set, signal *n* is added to the 'blocked' list.

BINDING

```
move.l      mask,-(sp)
move.w      #$116,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Psigblock()** returns the original set of blocked signals in effect prior to the call.

COMMENTS Blocked signals are preserved with **Pfork()** and **Pvfork()** calls, however, children started with **Pexec()** start with an empty list of blocked signals.

SIGKILL may not be blocked and will be reset by the system.

SEE ALSO **Pkill(), Psignal(), Psigpending()**

Psignal()

LONG Psignal(*sig*, *handler*)
WORD *sig*;
VOID (handler*)(LONG);**

Psignal() determines the action taken when a signal is received by the process.

OPCODE 274 (0x112)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *sig* specifies the signal whose response you wish to modify. If *handler* is cast to **SIG_DFL** (0) then the default action for the signal will occur when received. If *handler* is cast to **SIG_IGN** (1) then the signal will be ignored by the process. Otherwise, *handler* points to a user function which is designed to take action on a signal. This function is called when a signal is received with a **LONG** signal number on the stack.

BINDING

```
pea         handler
move.w      sig,-(sp)
move.w      #$112,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Psignal()** returns the old value of the signal handler if successful or a negative **GEMDOS** error code otherwise.

COMMENTS Signal handler functions may make any **GEMDOS**, **BIOS**, or **XBIOS** calls desired but must not make any **AES** or **VDI** calls. Signal handlers must either return with a 680x0 **RTS** instruction to resume program execution or call **Psigreturn()** to clean the stack if it intends to do a 'C' **longjmp()**.

Signal handling is preserved across **Pfork()** and **Pvfork()** calls. Child processes started with **Pexec()** ignore and follow the default action the same as their parents. Signals which have user functions assigned to them are reset to the default action for child processes.

SEE ALSO **Psigreturn(), Psigblock(), Pkill()**

Psigpause()

LONG Psigpause(*mask*)
LONG *mask*;

Psigpause() sets a new signal mask and then suspends the process until a signal is received.

OPCODE 310 (0x136)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.95 exists.

PARAMETERS *mask* specifies the signal mask to wait for.

BINDING

```
move.l      mask,-(sp)
move.w      #$136,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Psigpause()** returns 0 if successful or non-zero otherwise.

COMMENTS Depending on the state of the signal handler, this call may never return.

SEE ALSO **Psigaction(), Pause()**

Psigpending()

LONG Psigpending(VOID)

Psigpending() indicates which signals have been sent but not yet delivered to the calling process.

OPCODE 291 (0x123)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #123,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Psigpending()** returns a bit mask of which signals have been sent but not yet delivered to the calling process because they are being blocked. For each bit *n* set in the returned **LONG**, signal *n* is waiting for reception.

SEE ALSO **Psigblock(), Psignal(), Psigsetmask()**

Psigreturn()

VOID Psigreturn(VOID)

Psigreturn() prepares exit from a signal handler not planning to return via a 680x0 RTS.

OPCODE 282 (0x11A)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$11A,-(sp)
trap        #1
addq.l      #2,sp
```

CAVEATS Calling this function and then calling the 680x0 RTS opcode to return will produce undesired results.

COMMENTS **Psigreturn()** is only needed by 'C' programs which intend to exit the signal handler by doing a 'C' **longjmp()** rather than simply using the 680x0 RTS.

SEE ALSO **Psignal()**

Psigsetmask()

LONG Psigsetmask(*mask*)
LONG *mask*;

Psigsetmask() defines which signals are to be blocked before being delivered to the calling application.

OPCODE 279 (0x117)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *mask* is a **LONG** bit mask which defines which signals to block and which signals to allow. For each bit *n* set, signal *n* will be blocked. For each bit *n* clear, signal *n* will be delivered.

BINDING

```
move.l      mask,-(sp)
move.w      #$117,-(sp)
trap        #1
```

```
addq.l      #6,sp
```

RETURN VALUE **Psigsetmask()** returns the original mask of blocked/unblocked signals prior to the call or a negative **GEMDOS** error code.

COMMENTS Unlike **Psigblock()**, *mask* completely replaces the old mask rather than simply OR'ing it.

SEE ALSO **Pkill(), Psignal(), Psigpending()**

Pterm()

VOID Pterm(*retcode*)
WORD *retcode*;

Pterm() terminates an application returning the specified error code.

OPCODE 76 (0x4C)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *retcode* indicates the error status upon termination. Some recommended return values are:

Name	*retcode*	Meaning
TERM_OK	0	Program completion without errors
TERM_ERROR	1	Generic Error
TERM_BADPARAMS	2	Bad parameters
TERM_CRASH	-1	Process crashed (returned by **GEMDOS** versions from 0.15.)
TERM_CTRLC	-32	Process terminated by CTRL-C

BINDING

```
move.w      retcode,-(sp)
move.w      #$4C,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Pterm()** never returns.

COMMENTS **GEMDOS** jumps through the *etv_term* (0x102) vector when this call is made prior to process termination to allow the process one last chance to clean up. In addition, all files opened by the process are closed and all memory blocks allocated by the process are freed.

SEE ALSO **Pexec(), Pterm0()**

PtermØ()

VOID PtermØ(VOID)

PtermØ() terminates the application returning an exit code of 0 indicating no errors.

OPCODE 0 (0x00)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
clr.w       -(sp)
trap        #1
```

RETURN VALUE **PtermØ()** never returns.

COMMENTS Same as **Pterm(** 0 **)**.

SEE ALSO **Pterm()**

Ptermres()

VOID Ptermres(*keep*, *retcode*)
LONG *keep*;
WORD *retcode*;

Ptermres() terminates a process leaving a portion of the program's TPA intact and removing the memory left from **GEMDOS**'s memory list.

OPCODE 49 (0x31)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *keep* is the length (in bytes) of the processes' TPA to retain in memory after exit. *retcode* is the code returned on exit.

BINDING

```
move.w      retcode,-(sp)
move.l      keep,-(sp)
move.w      #$31,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE **Ptermres()** never returns.

COMMENTS This function is normally used by TSR's to stay resident in memory. Any files opened by the process are closed. Any memory allocated is, however, retained.

The value for *keep* is usually the sum of the length of the basepage (0x100), the length of the text, data, and bss segments of the application, and the length of the stack. It is important to note that the memory retained by this call may not be freed at a later point as it is removed from the **GEMDOS** memory list altogether.

SEE ALSO **Pterm0(), Pterm()**

Pumask()

WORD Pumask(*mode*)
WORD *mode*;

Pumask() defines an inital file and directory creation mask.

OPCODE 307 (0x133)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.92 exists.

PARAMETERS *mode* specifies the new file access permission mask to apply to all future files created with **Fcreate()** and **Dcreate()**. *mode* is a **WORD** bit mask of various access permission flags as defined in **Fchmod()**.

BINDING

```
move.w      mode,-(sp)
move.w      #$133,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Pumask()** returns the original mask in effect prior to the call.

SEE ALSO **Dcreate(), Fcreate(), Fchmod()**

Pusrval()

LONG Pursval(*val*)
LONG *val*;

Pusrval() reads/modifies a user defined value associated with a process.

OPCODE 280 (0x118)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *val* specifies the new value of the **LONG** associated with this process. If *val* is -1 then this value is not changed but still returned.

BINDING

```
move.w      #$118,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Pusrval()** returns the original value of the user **LONG** prior to the call.

COMMENTS The user-defined longword set by this call is inherited by child processes and may be utilized as desired.

Pvfork()

WORD Pvfork(VOID)

Pvfork() creates a duplicate of the current process which shares address and data space with the parent.

OPCODE 275 (0x113)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$113,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Pvfork()** returns the new process ID to the parent and 0 to the child. If an error occurs the parent receives a negative **GEMDOS** error code.

CAVEATS If the parent is in supervisor mode when this call is made the child is placed in user mode anyway.

COMMENTS The child process spawned by this function shares all address and data space with the parent. In other words, any variables altered by the parent will also be altered by the child and vice versa. The child process should not call **Mshrink()** as its TPA is already correctly sized.

The two processes do not execute concurrently. The parent is blocked until either the child terminates or calls **Pexec()**'s mode 200.

SEE ALSO **Pexec(), Pfork()**

Pwait()

LONG Pwait(VOID)

Pwait() attempts to determine the exit code of a stopped or terminated child process.

OPCODE 265 (0x109)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$109,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Pwait()** returns 0 if no child processes have terminated or a 32-bit return code for a child process which has been terminated or stopped.

The process ID of the child process is placed in the upper 16 bits. A process which returned an exit status (via **Pterm()**, **Ptermres()**, or **Pterm0()**) returns the exit code in the lower 16 bits.

A process which was stopped as the result of a signal returns 0x*nn*7F where *nn* is the signal number which stopped it. A process which was terminated as the result of a signal returns 0x*nn*00 where *nn* is the signal number which killed the process.

COMMENTS **Pwait()** will block the calling process until at least one child has been stopped or terminated. Once the exit code of a process has been returned with this call it will be not be returned again with this call (unless it had been stopped and is restarted and stopped again). This call is identical to **Pwait3(** 2, **NULL**);

SEE ALSO **Pexec(), Pterm(), Ptermres(), Pterm0()**

Pwait3()

LONG Pwait3(*flag*, *rusage*)
WORD *flag*;
LONG **rusage*;

Pwait3() determines the exit code of any children of the calling process which were stopped and/or terminated.

OPCODE 284 (0x11C)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *flag* is a bit mask which specifies the specifics of this call as follows:

Name	Mask	Meaning
PW_NOBLOCK	0x01	If set, the function will not block the calling process if no child has been stopped or terminated, rather it will simply return 0. If clear, the process will be blocked until a child of the process has terminated or is stopped.
PW_STOPPED	0x02	If set, return exit codes for processes which have been terminated as well as stopped. If clear, only return exit codes for processes which have actually terminated.

rusage points to an array of two **LONG**s which are filled in with resource usage information of the stopped or terminated process. The first **LONG** contains the number of milliseconds used by the child in user code. The second **LONG** indicates the number of milliseconds spent by the process in the kernel. *rusage* may be set to **NULL** if this information is undesired.

BINDING

```
pea         rusage
move.w      flag,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Pwait3()** returns 0 if no child processes have been stopped and/or terminated (depending on *flag*) or a 32-bit return code for a child process which has been terminated or stopped.

The process ID of the child process is placed in the upper 16 bits. A process which returned an exit status (via **Pterm()**, **Ptermres()**, or **Pterm0()**) returns the exit code in the lower 16 bits.

A process which was stopped as the result of a signal returns 0x*nn*7F where *nn* is the signal number which stopped it. A process which was terminated as the result of a signal returns 0x*nn*00 where *nn* is the signal number which killed the process.

SEE ALSO **Pwait(), Pexec(), Pterm(), Pterm0(), Ptermres(), Prusage()**

Pwaitpid()

LONG Pwaitpid(***pid, flag, rusage*** **)**
WORD ***pid, flag*****;**
LONG ****rusage***;**

Pwaitpid() returns exit code information about one or more child processes.

OPCODE 314 (0x13A)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.96 exists.

PARAMETERS *pid* specifies the children whose exit codes are of interest as follows.

A *pid* of **PWP_ALL** (-1) indicates that all children are of interest. A *pid* of less than -1 indicates that any child whose process group is -*pid* is of interest. A *pid* of **PWP_GROUP** (0) indicates that any child with the same process group ID of the parent is of interest. A *pid* greater than 0 indicates that the child with the given process ID is of interest.

For the usage of *flag* and *rusage* see **Pwait3()**.

BINDING

```
pea         rusage
move.w      flag,-(sp)
move.w      #$13A,-(sp)
trap        #1
addq.l      #8,sp
```

RETURN VALUE See **Pwait3()**.

SEE ALSO **Pwait(), Pwait3()**

Salert()

VOID Salert(***str*** **)**
char ****str***;**

Salert() sends an alert string to the alert pipe 'U:\PIPE\ALERT\'.

OPCODE 316 (0x13C)

AVAILABILITY Available when a '**MiNT**' cookie with a version of at least 0.98 exists.

PARAMETERS *str* should point to a **NULL** terminated character string containing the alert

message to display. The message should not contain any carriage returns or escape characters. The string should *not* be formatted as in **form_alert()**.

BINDING

```
pea         str
move.w      #$13C,-(sp)
trap        #1
addq.l      #6,sp
```

CAVEATS

Messages sent by **Salert()** are only delivered if a separate application is present which was designed to listen to the alert pipe and post its contents.

SEE ALSO

form_alert()

Super()

VOIDP Super(*stack*)
VOIDP *stack*;

Super() allows you to interrogate or alter the state of the 680x0.

OPCODE

32 (0x20)

AVAILABILITY

All **GEMDOS** versions.

PARAMETERS

stack defines the meaning of the call as follows:

Name	*stack*	Meaning
SUP_SET	(**VOIDP**)0	The processor is placed in supervisor mode and the old supervisor stack is returned.
SUP_INQUIRE	(**VOIDP**)1	This interrogates the current mode of the processor. If the processor is in user mode a **SUP_USER** (0) is returned, otherwise a **SUP_SUPER** (1) is returned.
—	>1	The processor is placed in user mode and the supervisor stack is reset to *stack*.

BINDING

```
pea         stack
move.w      #$20,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE

Super() returns a different value based on the *stack* parameter. The various return values are explained above.

CAVEATS

You should never call the **AES** in supervisor mode. In addition, supervisor mode should be entered and left in the same stack context (same 'C' function) or stack corruption can result.

COMMENTS To execute portion of a program in supervisor mode you normally call **Super()** with a parameter of 0 and save the return value. When ready to return to user mode you call **Super()** again with the saved return value as a parameter.

Supervisor mode should be used sparingly under **MiNT** as no task switching can occur.

SEE ALSO **Supexec()**

Sversion()

UWORD Sversion(VOID)

Sversion() returns the current **GEMDOS** version number.

OPCODE 48 (0x30)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$30,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Sversion()** returns a **UWORD** containing the **GEMDOS** minor version number in the upper word and the major version number in the lower word. Current values returned by Atari **TOS**'s are:

Return Value	TOS versions (normally) found in:
0x1300 (0.13)	**TOS** 1.0, **TOS** 1.02
0x1500 (0.15)	**TOS** 1.04, **TOS** 1.06
0x1700 (0.17)	**TOS** 1.62
0x1900 (0.19)	**TOS** 2.01, **TOS** 2.05, **TOS** 2.06, **TOS** 3.01, **TOS** 3.05, **TOS** 3.06
0x3000 (0.30)	**TOS** 4.00, **TOS** 4.01, **TOS** 4.02, **TOS** 4.03, **TOS** 4.04, **MultiTOS** 1.00, **MultiTOS** 1.08

COMMENTS The **GEMDOS** number is not associated with the **TOS** or **AES** version number. You should check for **GEMDOS** or **MiNT** version numbers when trying to determine the presence or properties of a **GEMDOS** function.

Syield()

VOID Syield(VOID)

Syield() surrenders the remainder of the callers' current process timeslice.

OPCODE 255 (0xFF)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

BINDING

```
move.w      #$FF,-(sp)
trap        #1
addq.l      #2,sp
```

SEE ALSO **Pause(), Fselect()**

Sysconf()

LONG Sysconf(*inq*)
WORD *inq*;

Sysconf() returns information about the limits or capabilities of the currently running version of **MiNT**.

OPCODE 290 (0x122)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *inq* determines the return value as follows:

Name	inq	Return Value
SYS_MAXINQ	-1	Maximum legal value for *inq*.
SYS_MAXREGIONS	0	Maximum memory regions per process.
SYS_MAXCOMMAND	1	Maximum length of **Pexec()** command string.
SYS_MAXFILES	2	Maximum number of open files per process.
SYS_MAXGROUPS	3	Maximum number of supplementary group ID's.
SYS_MAXPROCS	4	Maximum number of processes per user.

BINDING

```
move.w      inq,-(sp)
move.w      #$122,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE See above.

COMMENTS If the requested item returns **UNLIMITED** (0x7FFFFFFF) then that item is unlimited.

SEE ALSO **Dpathconf()**

Talarm()

LONG Talarm(*time*)
LONG *time*;

Talarm() reads/sets a process alarm for the current process.

OPCODE 288 (0x120)

AVAILABILITY This function is available under all **MiNT** versions integrated with **MultiTOS**.

PARAMETERS *time* specifies the length of time (in milliseconds) to wait before a **SIGALRM** signal is delivered. If time is 0 then any previously set alarm is cancelled. If *time* is negative the function does not modify any alarm currently set.

BINDING

```
move.l      time,-(sp)
move.w      #$120,-(sp)
trap        #1
addq.l      #6,sp
```

RETURN VALUE **Talarm()** returns 0 i f no alarm was scheduled prior to this call or the amount of time remaining (in milliseconds) before the alarm is triggered.

CAVEATS An alarm with less than 1000 remaining milliseconds will return a value of 0.

COMMENTS If no **SIGALRM** signal handler has been set up when the alarm is triggered, the process will be killed.

SEE ALSO **Pause(), Psignal()**

Tgetdate()

UWORD Tgetdate(VOID)

Tgetdate() returns the current **GEMDOS** date.

OPCODE 42 (0x2A)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$2A,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Tgetdate()** returns a bit array **UWORD** arranged as follows:

Bits 15-9	Bits 8-5	Bits 4-0
Years since 1980	Month (1-12)	Date (0-31)

SEE ALSO **Tgettime(), Tsetdate(), Gettime()**

Tgettime()

UWORD Tgettime(VOID)

Tgettime() returns the **GEMDOS** system time.

OPCODE 44 (0x2C)

AVAILABILITY All **GEMDOS** versions.

BINDING

```
move.w      #$2C,-(sp)
trap        #1
addq.l      #2,sp
```

RETURN VALUE **Tgettime()** returns a bit array arranged as follows:

Bits 15-11	Bits 10-5	Bits 4-0
Hour (0-23)	Minute (0 to 59)	Secs/2 (0 to 29)

SEE ALSO **Tgetdate(), Tsettime(), Gettime()**

Tsetdate()

WORD Tsetdate(***date*** **)**
UWORD ***date*****;**

Tsetdate() sets the current **GEMDOS** date.

OPCODE 43 (0x2B)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *date* is a bit array arranged as illustrated under **Tgetdate()**.

BINDING

```
move.w      date,-(sp)
move.w      #$2B,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Tsetdate()** returns 0 if the operation was successful or non-zero if a bad date is given.

CAVEATS **GEMDOS** version 0.13 did not inform the **BIOS** of the date change and hence would not change the **IKBD** date or the date of a battery backed-up clock,

SEE ALSO **Tgetdate(), Tsettime(), Settime()**

Tsettime()

WORD Tsettime(*time*)
UWORD *time*;

Tsettime() sets the current **GEMDOS** time.

OPCODE 45 (0x2D)

AVAILABILITY All **GEMDOS** versions.

PARAMETERS *time* is a bit array arranged as illustrated under **Tgettime()**.

BINDING

```
move.w      time,-(sp)
move.w      #$2D,-(sp)
trap        #1
addq.l      #4,sp
```

RETURN VALUE **Tsettime()** returns 0 if the time was set or non-zero if the time given was invalid.

CAVEATS **GEMDOS** version 0.13 did not inform the **BIOS** of the date change and hence would not change the **IKBD** date or the date of a battery backed-up clock.

SEE ALSO **Tgettime(), Tsetdate(), Settime()**