NAME

fcntl − miscellaneous file descriptor control functions

SYNOPSIS

#include <fcntl.h>

int fcntl(int *fd***, int** *cmd***,** [*data*]**)**

DESCRIPTION

Fcntl() performs several file descriptor related functions, like duplicating a file descriptor, setting the "close on exec" attribute, etc. The *fd* argument is the file descriptor to operate on, *cmd* is the command code of the operation to perform, and *data* is an optional argument to give or receive parameters. The command codes and other symbols and types are declared in <fcntl.h>. The commands are:

fcntl(*fd***, F_DUPFD, int** *fd2***)**

Returns a new file descriptor that is a duplicate of file descriptor *fd*. It shares the same file pointer and the same file status flags, but has separate file descriptor flags that are initially off. The value of the duplicate file descriptor is the first free file descriptor greater than or equal to *fd2*.

fcntl(*fd***, F_GETFD)**

Returns the file descriptor flags associated with file descriptor *fd*. The flags are the "close on exec" flag **FD_CLOEXEC** that, when set, causes the file descriptor to be closed when the process executes another program. The Minix-vmd specific **FD_ASYNCHIO** flag marks a file descriptor for asynchronous I/O operation.

fcntl(*fd***, F_SETFD, int** *flags***)**

Set the file descriptor flags of *fd* to *flags*.

fcntl(*fd***, F_GETFL)**

Return the file status flags and file access modes associated with the file associated with file descriptor *fd*. The file status flags are **O_NONBLOCK** (non blocking I/O) and **O_APPEND** (append mode). The file access modes are **O_RDONLY** (read-only), **O_WRONLY** (write-only) and **O_RDWR** (read-write). These flags are also used in the second argument of **open**(2).

fcntl(*fd***, F_SETFL, int** *flags***)**

Set the file status flags of the file referenced by *fd* to *flags*. Only **O_NONBLOCK** and **O_APPEND** may be changed. Access mode flags are ignored.

The next four commands use a parameter of type **struct flock** that is defined in <fcntl.h> as:

```
struct flock {
    short   l_type;     /* F_RDLCK, F_WRLCK, or F_UNLCK */
    short   l_whence;   /* SEEK_SET, SEEK_CUR, or SEEK_END */
    off_t   l_start;    /* byte offset to start of segment */
    off_t   l_len;      /* length of segment */
    pid_t   l_pid;      /* process id of the locks' owner */
};
```

This structure describes a segment of a file. **L_type** is the lock operation performed on the file segment: **F_RDLCK** to set a read lock, **F_WRLCK** to set a write lock, and **F_UNLCK** to remove a lock. Several processes may have a read lock on a segment, but only one process can have a write lock. **L_whence** tells if the **l_start** offset must be interpreted from the start of the file (**SEEK_SET**), the current file position (**SEEK_CUR**), or the end of the file (**SEEK_END**). This is analogous to the third parameter of **lseek**(2). These **SEEK_*** symbols are declared in <unistd.h>. **L_start** is the starting offset of the segment of the file. **L_end** is the length of the segment. If zero then the segment extends until end of file. **L_pid** is the process-id of the process currently holding a lock on the segment. It is returned by **F_GETLK**.

fcntl(*fd***, F_GETLK, struct flock** ***lkp***)**

Find out if some other process has a lock on a segment of the file associated by file descriptor *fd* that overlaps with the segment described by the **flock** structure pointed to by *lkp*. If the segment is not locked then **l_type** is set to **F_UNLCK**. Otherwise an **flock** structure is returned through *lkp* that describes the lock held by the other process. **L_start** is set relative to the start of the file.

fcntl(*fd***, F_SETLK, struct flock ****lkp***)**

Register a lock on a segment of the file associated with file descriptor *fd*. The file segment is described by the **struct flock** pointed to by *lkp*. This call returns an error if any part of the segment is already locked.

fcntl(*fd***, F_SETLKW, struct flock ****lkp***)**

Register a lock on a segment of the file associated with file descriptor *fd*. The file segment is described by the **struct flock** pointed to by *lkp*. This call blocks waiting for the lock to be released if any part of the segment is already locked.

fcntl(*fd***, F_FREESP, struct flock ****lkp***)**

This Minix-vmd specific call frees a segment of disk space occupied by the file associated with file descriptor *fd*. The segment is described by the **struct flock** pointed to by *lkp*. The file is truncated in length to the byte position indicated by **l_start** if **l_len** is zero. If **l_len** is nonzero then the file keeps its size, but the freed bytes now read as zeros. (Other than sharing the flock structure, this call has nothing to do with locking.) (This call is common among UNIX(-like) systems.)

fcntl(*fd***, F_SEEK, u64_t** *pos***)**

This Minix-vmd specific call sets the file position of the file associated with file descriptor *fd* to the byte offset indicated by the 64-bit number *pos*. This is analogous to the call

lseek(*fd***,** *pos***, SEEK_SET)**

except that **F_SEEK** can be used on devices larger than 4 gigabyte.

SEE ALSO

open(2), **dup**(2), **lseek**(2), **ftruncate**(3), **int64**(3).

DIAGNOSTICS

Fcntl returns a file descriptor, flags, or **0** to indicate success. On error **−1** is returned, with **errno** set to the appropriate error code. The most notable errors are:

EINTR

If a blocked **F_SETLKW** operation is interrupted by a signal that is caught.

EAGAIN

By **F_SETLK** if a segment cannot be locked.

EBADF

A bad file descriptor in general, or an attempt to place a write lock on a file that is not open for writing, etc.

ENOLCK

No locks available, the file system code has run out of internal table space.

AUTHOR

Kees J. Bot <kjb@cs.vu.nl>