NAME

opendir – open a directory / readdir – read a directory

SYNOPSIS

#include <sys/types.h>

#include <dirent.h>

DIR *opendir(const char *_name_);

struct dirent *readdir(DIR *_dir_);

DESCRIPTION opendir

The **opendir()** function opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

RETURN VALUE

The **opendir()** function returns a pointer to the directory stream or NULL if an error occurred.

DESCRIPTION readdir

The **readdir()** function returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to by *dir*. It returns NULL on reaching the end-of-file or if an error occurred.

The data returned by **readdir()** is overwritten by subsequent calls to **readdir()** for the same directory stream.

The *dirent* structure is defined as follows:

```
struct dirent {
    long          d_ino;          /* inode number */
    off_t         d_off;          /* offset to the next dirent */
    unsigned short  d_reclen;     /* length of this record */
    unsigned char   d_type;     /* type of file */
    char          d_name[256];  /* filename */
};
```

RETURN VALUE

The **readdir()** function returns a pointer to a dirent structure, or NULL if an error occurs or end-of-file is reached.

ERRORS

EACCES
Permission denied.

EMFILE
Too many file descriptors in use by process.

ENFILE
Too many files are currently open in the system.

ENOENT
Directory does not exist, or *name* is an empty string.

ENOMEM
Insufficient memory to complete the operation.

ENOTDIR
name is not a directory.

SEE ALSO

open(2), **readdir**(3), **closedir**(3), **rewinddir**(3), **seekdir**(3), **telldir**(3), **scandir**(3)

NAME

exec, execl, execv, execle, execve, execlp, execvp – execute a file

SYNOPSIS

#include <unistd.h>

int execl(const char *_path_, const char *_arg0_, ..., const char *_argn_, char * /*NULL*/);

int execv(const char *_path_, char *const _argv[ ]_);

int execle(const char *_path_,char *const _arg0[ ]_, ... , const char *_argn_, char * /*NULL*/, char *const _envp[ ]_);

int execve (const char *_path_, char *const _argv[ ]_ char *const _envp[ ]_);

int execlp (const char *_file_, const char *_arg0_, ..., const char *_argn_, char * /*NULL*/);

int execvp (const char *_file_, char *const _argv[ ]_);

DESCRIPTION

Each of the functions in the **exec** family overlays a new process image on an old process. The new process image is constructed from an ordinary, executable file. This file is either an executable object file, or a file of data for an interpreter. There can be no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.

When a C program is executed, it is called as follows:

int main (int argc, char *argv[], char *envp[]);

where *argc* is the argument count, *argv* is an array of character pointers to the arguments themselves, and *envp* is an array of character pointers to the environment strings. As indicated, *argc* is at least one, and the first member of the array points to a string containing the name of the file.

The arguments *arg0*, ..., *argn* point to null-terminated character strings. These strings constitute the argument list available to the new process image. Conventionally at least *arg0* should be present. The *arg0* argument points to a string that is the same as *path* (or the last component of *path*). The list of argument strings is terminated by a **(char *)0** argument.

The *argv* argument is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process image. By convention, *argv* must have at least one member, and it should point to a string that is the same as *path* (or its last component). The *argv* argument is terminated by a null pointer.

The *path* argument points to a path name that identifies the new process file.

The *file* argument points to the new process file. If *file* does not contain a slash character, the path prefix for this file is obtained by a search of the directories passed in the **PATH** environment variable (see **environ**(5)).

File descriptors open in the calling process remain open in the new process.

Signals that are being caught by the calling process are set to the default disposition in the new process image (see **signal**(3C)). Otherwise, the new process image inherits the signal dispositions of the calling process.

RETURN VALUES

If a function in the **exec** family returns to the calling process, an error has occurred; the return value is **–1** and **errno** is set to indicate the error.

NAME

fopen – open a stream

SYNOPSIS

#include <stdio.h>

FILE *fopen(const char * *filename***, const char *** *mode***);**

DESCRIPTION

The **fopen( )** function opens the file whose pathname is the string pointed to by *filename*, and associates a stream with it.

The argument *mode* points to a string beginning with one of the following sequences:

r or **rb**	open file for reading
w or **wb**	truncate to zero length or create file for writing
a or **ab**	append; open or create file for writing at end-of-file
r+ or **rb+** or **r+b**	open file for update (reading and writing)
w+ or **wb+** or **w+b**	truncate to zero length or create file for update
a+ or **ab+** or **a+b**	append; open or create file for update, writing at end-of-file

The character **b** has no effect, but is allowed for ISO C standard conformance. Opening a file with read mode (**r** as the first character in the *mode* argument) fails if the file does not exist or cannot be read.

When a file is opened with update mode (**+** as the second or third character in the *mode* argument), both input and output may be performed on the associated stream. However, output must not be directly followed by input without an intervening call to **fflush**(3S) or to a file positioning function (**fseek**(3S), **fsetpos**(3S) or **rewind**(3S)), and input must not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.

When opened, a stream is fully buffered if and only if it can be determined not to refer to an interactive device. The error and end-of-file indicators for the stream are cleared.

If *mode* is **w**, **a**, **w+** or **a+** and the file did not previously exist, upon successful completion, **fopen( )** function will mark for update the **st_atime**, **st_ctime** and **st_mtime** fields of the file and the **st_ctime** and **st_mtime** fields of the parent directory.

If *mode* is **w** or **w+** and the file did previously exist, upon successful completion, **fopen( )** will mark for update the **st_ctime** and **st_mtime** fields of the file. The **fopen( )** function will allocate a file descriptor as **open**(2) does.

The largest value that can be represented correctly in an object of type **off_t** will be established as the offset maximum in the open file description.

RETURN VALUES

Upon successful completion, **fopen( )** returns a pointer to the object controlling the stream. Otherwise, a null pointer is returned, and **errno** is set to indicate the error.

fopen( ) may fail and not set **errno** if there are no free **stdio** streams.

ERRORS

The **fopen( )** function will fail if:

EACCES Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *mode* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.

EINTR A signal was caught during **fopen( )**.

EISDIR The named file is a directory and *mode* requires write access.

SEE ALSO

fclose(3S), **fdopen**(3S), **fflush**(3S), **freopen**(3S), **fsetpos**(3S), **rewind**(3S),

NAME

gets, fgets – get a string from a stream

SYNOPSIS

#include <stdio.h>

char *gets(char * *s***);**

char *fgets(char * *s***, int** *n***, FILE *** *stream***);**

DESCRIPTION

The **gets( )** function reads characters from the standard input stream (see **intro**(3)), **stdin**, into the array pointed to by *s*, until a newline character is read or an end-of-file condition is encountered. The newline character is discarded and the string is terminated with a null character.

The **fgets( )** function reads characters from the *stream* into the array pointed to by *s*, until *n*−1 characters are read, or a newline character is read and transferred to *s*, or an end-of-file condition is encountered. The string is then terminated with a null character.

When using **gets( )**, if the length of an input line exceeds the size of *s*, indeterminate behavior may result. For this reason, it is strongly recommended that **gets( )** be avoided in favor of **fgets( )**.

RETURN VALUES

If end-of-file is encountered and no characters have been read, no characters are transferred to *s* and a null pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a null pointer is returned and the error indicator for the stream is set. If end-of-file is encountered, the **EOF** indicator for the stream is set. Otherwise *s* is returned.

ERRORS

The **gets( )** and **fgets( )** functions will fail if data needs to be read and:

EOVERFLOW The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding *stream*.

SEE ALSO

lseek(2), **read**(2), **ferror**(3S), **fopen**(3S), **fread**(3S), **getc**(3S), **scanf**(3S), **stdio**(3S), **ungetc**(3S), **attributes**(5)

NAME

sigaction – POSIX signal handling functions.

SYNOPSIS

#include <signal.h>

int sigaction(int *signum***, const struct sigaction ****act***, struct sigaction ****oldact***);**

DESCRIPTION

The **sigaction** system call is used to change the action taken by a process on receipt of a specific signal.

signum specifies the signal and can be any valid signal except **SIGKILL** and **SIGSTOP**.

If *act* is non−null, the new action for signal *signum* is installed from *act*. If *oldact* is non−null, the previous action is saved in *oldact*.

The **sigaction** structure is defined as something like

```
struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

On some architectures a union is involved - do not assign to both *sa_handler* and *sa_sigaction*.

The *sa_restorer* element is obsolete and should not be used. POSIX does not specify a *sa_restorer* element.

sa_handler specifies the action to be associated with *signum* and may be **SIG_DFL** for the default action, **SIG_IGN** to ignore this signal, or a pointer to a signal handling function.

sa_mask gives a mask of signals which should be blocked during execution of the signal handler. In addition, the signal which triggered the handler will be blocked, unless the **SA_NODEFER** or **SA_NOMASK** flags are used.

sa_flags specifies a set of flags which modify the behaviour of the signal handling process. It is formed by the bitwise OR of zero or more of the following:

SA_NOCLDSTOP
: If *signum* is **SIGCHLD**, do not receive notification when child processes stop (i.e., when child processes receive one of **SIGSTOP**, **SIGTSTP**, **SIGTTIN** or **SIGTTOU**).

SA_RESTART
: Provide behaviour compatible with BSD signal semantics by making certain system calls restartable across signals.

RETURN VALUES

sigaction returns 0 on success and -1 on error.

ERRORS

EINVAL
: An invalid signal was specified. This will also be generated if an attempt is made to change the action for **SIGKILL** or **SIGSTOP**, which cannot be caught.

SEE ALSO

kill(1), **kill**(2), **killpg**(2), **pause**(2), **sigsetops**(3),

NAME

sigprocmask – change and/or examine caller's signal mask
sigsuspend – install a signal mask and suspend caller until signal

SYNOPSIS

#include <signal.h>

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

int sigsuspend(const sigset_t ****set*);**

DESCRIPTION sigprocmask

The **sigprocmask( )** function is used to examine and/or change the caller's signal mask. If the value is **SIG_BLOCK**, the set pointed to by the argument *set* is added to the current signal mask. If the value is **SIG_UNBLOCK**, the set pointed by the argument *set* is removed from the current signal mask. If the value is **SIG_SETMASK**, the current signal mask is replaced by the set pointed to by the argument *set*. If the argument *oset* is not NULL, the previous mask is stored in the space pointed to by *oset*. If the value of the argument *set* is NULL, the value *how* is not significant and the caller's signal mask is unchanged; thus, the call can be used to inquire about currently blocked signals.

If there are any pending unblocked signals after the call to **sigprocmask( )**, at least one of those signals will be delivered before the call to **sigprocmask( )** returns.

It is not possible to block those signals that cannot be ignored this restriction is silently imposed by the system. See **sigaction**(2).

If **sigprocmask( )** fails, the caller's signal mask is not changed.

RETURN VALUES

On success, **sigprocmask( )** returns **0**. On failure, it returns **−1** and sets **errno** to indicate the error.

ERRORS

sigprocmask( ) fails if any of the following is true:

EFAULT *set* or *oset* points to an illegal address.

EINVAL The value of the *how* argument is not equal to one of the defined values.

DESCRIPTION sigsuspend

sigsuspend( ) replaces the caller's signal mask with the set of signals pointed to by the argument *set* and then suspends the caller until delivery of a signal whose action is either to execute a signal catching function or to terminate the process.

If the action is to terminate the process, **sigsuspend( )** does not return. If the action is to execute a signal catching function, **sigsuspend( )** returns after the signal catching function returns. On return, the signal mask is restored to the set that existed before the call to **sigsuspend( )**.

It is not possible to block those signals that cannot be ignored (see **signal**(5)); this restriction is silently imposed by the system.

RETURN VALUES

Since **sigsuspend( )** suspends process execution indefinitely, there is no successful completion return value. On failure, it returns −1 and sets **errno** to indicate the error.

ERRORS

sigsuspend( ) fails if either of the following is true:

EFAULT *set* points to an illegal address.

EINTR A signal is caught by the calling process and control is returned from the signal catching function.

SEE ALSO

sigaction(2), **sigsetops**(3C),

NAME

sigsetops, sigemptyset, sigfillset, sigaddset, sigdelset, sigismember – manipulate sets of signals

SYNOPSIS

#include <signal.h>

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

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

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

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

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

DESCRIPTION

These functions manipulate *sigset_t* data types, representing the set of signals supported by the implementation.

sigemptyset( ) initializes the set pointed to by *set* to exclude all signals defined by the system.

sigfillset( ) initializes the set pointed to by *set* to include all signals defined by the system.

sigaddset( ) adds the individual signal specified by the value of *signo* to the set pointed to by *set*.

sigdelset( ) deletes the individual signal specified by the value of *signo* from the set pointed to by *set*.

sigismember( ) checks whether the signal specified by the value of *signo* is a member of the set pointed to by *set*.

Any object of type *sigset_t* must be initialized by applying either **sigemptyset( )** or **sigfillset( )** before applying any other operation.

RETURN VALUES

Upon successful completion, the **sigismember( )** function returns a value of one if the specified signal is a member of the specified set, or a value of 0 if it is not. Upon successful completion, the other functions return a value of 0. Otherwise a value of −1 is returned and **errno** is set to indicate the error.

ERRORS

sigaddset( ), **sigdelset( )**, and **sigismember( )** will fail if the following is true:

EINVAL The value of the *signo* argument is not a valid signal number.

sigfillset( ) will fail if the following is true:

EFAULT The *set* argument specifies an invalid address.

SEE ALSO

sigaction(2), **sigpending**(2), **sigprocmask**(2), **sigsuspend**(2), **attributes**(5), **signal**(5)

NAME

unlink – remove directory entry

SYNOPSIS

#include <unistd.h>

int unlink(const char ****path*);**

DESCRIPTION

The **unlink( )** function removes a link to a file. It removes the link named by the pathname pointed to by *path* and decrements the link count of the file referenced by the link.

When the file's link count becomes 0 and no process has the file open, the space occupied by the file will be freed and the file will no longer be accessible. If one or more processes have the file open when the last link is removed, the link will be removed before **unlink( )** returns, but the removal of the file contents will be postponed until all references to the file are closed.

RETURN VALUES

Upon successful completion, **0** is returned. Otherwise, **–1** is returned and **errno** is set to indicate the error.

ERRORS

The **unlink( )** function will fail and not unlink the file if:

EACCES Search permission is denied for a component of the *path* prefix.

EACCES Write permission is denied on the directory containing the link to be removed.

ENOENT The named file does not exist or is a null pathname.

ENOTDIR A component of the *path* prefix is not a directory.

EPERM The named file is a directory and the effective user of the calling process is not super-user.

SEE ALSO

rm(1), **close**(2), **link**(2), **open**(2), **rmdir**(2),