NAME

directory, opendir, readdir, readdir_r, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS

#include <dirent.h>

DIR ∗**opendir(const char** ∗*filename***);**

struct dirent ∗**readdir(DIR** ∗*dirp***);**

long telldir(DIR ∗*dirp***);**

void seekdir(DIR ∗*dirp***, long** *loc***);**

void rewinddir(DIR ∗*dirp***);**

int closedir(DIR ∗*dirp***);**

Solaris 2.4

struct dirent ∗**readdir_r(DIR** ∗*dirp,***struct dirent** ∗*res***);**

POSIX

cc [*flag* **...]** *file* **...** –**D_POSIX_PTHREAD_SEMANTICS [** *library* **...]**

int ∗**readdir_r(DIR** ∗*dirp,***struct dirent** ∗*entry* **struct dirent** ∗∗*result***);**

MT-LEVEL

See the **NOTES** section of this page.

DESCRIPTION

opendir( ) opens the directory named by *filename* and associates a directory stream with it. **opendir( )** returns a pointer to be used to identify the directory stream in subsequent operations. The directory stream is positioned at the first entry. A null pointer is returned if *filename* cannot be accessed or is not a directory, or if it cannot **malloc**(3C) enough memory to hold a **DIR** structure or a buffer for the directory entries.

readdir( ) returns a pointer to a structure representing the directory entry at the current position in the directory stream to which *dirp* refers, and positions the directory stream at the next entry, except on read-only filesystems. It returns a **NULL** pointer upon reaching the end of the directory stream, or upon detecting an invalid location in the directory. **readdir( )** shall not return directory entries containing empty names. It is unspecified whether entries are returned for dot or dot-dot. The pointer returned by **readdir( )** points to data that may be overwritten by another call to **readdir( )** on the same directory stream. This data shall not be overwritten by another call to **readdir( )** on a different directory stream. **readdir( )** may buffer several directory entries per actual read operation; **readdir( )** marks for updata the *st_atime* field of the directory each time the directory is actually read.

readdir_r( ) has the equivalent functionality as **readdir( )** except that a buffer *res* must be supplied by the caller to store the result. To allocate *res* correctly a *struct dirent res* is not sufficient, thus the size should be *sizeof(struct dirent)* + **_POSIX_PATH_MAX** (defined in **<limits.h>**).

The POSIX **readdir_r( )** function initializes the structure referenced by *entry* and stores a pointer to this structure in *result*.

telldir( ) returns the current location associated with the named directory stream.

seekdir( ) sets the position of the next **readdir( )** operation on the directory stream. The new position reverts to the position associated with the directory stream at the time the **telldir( )** operation that provides *loc* was performed. Values returned by **telldir( )** are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the **telldir( )** value may be invalidated due to undetected directory compaction. It is safe to use a previous **telldir( )** value immediately after a call to **opendir( )** and before any calls to **readdir**.

rewinddir( ) resets the position of the named directory stream to the beginning of the directory. It also causes the directory stream to refer to the current state of the corresponding directory, as a call to **opendir( )** would.

closedir( ) closes the named directory stream and frees the **DIR** structure.

RETURN VALUES

opendir( ), **readdir( )**, and **readdir_r( )** return **NULL** on failure and set **errno** to indicate the error. The POSIX **readdir_r( )** returns zero if successful, or an error number to indicate failure. **telldir( )**, **seekdir( )**, and **closedir( )** return –1 on failure and set **errno** to indicate the error.

ERRORS

opendir( ) will fail if one or more of the following are true:

EACCES	Read permission is denied on the specified directory.
EFAULT	*filename* points outside the allocated address space.
ELOOP	Too many symbolic links were encountered in translating *filename*.
ENOTDIR	A component of *filename* is not a directory.
EMFILE	The maximum number of file descriptors are currently open.
ENFILE	The system file table is full.
ENAMETOOLONG	The length of the *filename* argument exceeds **{PATH_MAX}**, or the length of a *filename* component exceeds **{NAME_MAX}** while **{_POSIX_NO_TRUNC}** is in effect.
ENOENT	A component of *filename* does not exist or is a null pathname.
EACCES	A component of *filename* denies search permission.

readdir( ) and **readdir_r( )** will fail if one or more of the following are true:

EAGAIN	Mandatory file/record locking was set, **O_NDELAY** or **O_NONBLOCK** was set, and there was a blocking record lock.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EAGAIN	No data is waiting to be read on a file associated with a tty device and **O_NONBLOCK** was set.
EAGAIN	No message is waiting to be read on a stream and **O_NDELAY** or **O_NONBLOCK** was set.
EBADF	The file descriptor determined by the **DIR** stream is no longer

valid. This results if the **DIR** stream has been closed.

EBADMSG Message waiting to be read on a stream is not a data message.

EDEADLK The **read( )** was going to go to sleep and cause a deadlock to occur.

EFAULT *buf* points to an illegal address.

EINTR A signal was caught during the **read( )** or **readv( )** function.

EINVAL Attempted to read from a stream linked to a multiplexor.

EIO A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the **SIGTTIN** signal or the process group of the process is orphaned.

ENOENT The current file pointer for the directory is not located at a valid entry.

ENOLCK The system record lock table was full, so the **read( )** or **readv( )** could not go to sleep until the blocking record lock was removed.

ENOLINK *fildes* is on a remote machine and the link to that machine is no longer active.

ENXIO The device associated with *fildes* is a block special or character special file and the value of the file pointer is out of range.

telldir( ), **seekdir( )**, and **closedir( )** return 0 on success and will fail if one or more of the following are true:

EBADF The file descriptor determined by the **DIR** stream is no longer valid. This results if the **DIR** stream has been closed.

EXAMPLES

Here is a sample program that prints the names of all the files in the current directory:

```
#include <stdio.h>
#include <dirent.h>

main( )
{
        DIR *dirp;
        struct dirent *direntp;

        dirp = opendir( "." );
        while ( (direntp = readdir( dirp )) != NULL )
                (void)printf( "%s\n", direntp->d_name );
        (void)closedir( dirp );
        return (0);
}
```

SEE ALSO

getdents(2), **dirent**(4)

NOTES

When compiling multi-thread applications, the **_REENTRANT** flag must be defined on the compile line. This flag should only be used in multi-thread applications.

readdir( ) is unsafe in multi-thread applications. **readdir_r( )** is safe, and should be used instead. **closedir( )**, **directory( )**, **opendir( )**, **rewinddir( )**, **seekdir( )**, and **telldir( )** are safe in multi-thread applications.

The POSIX **readdir_r( )** interface is as specified in POSIX 1003.1c Draft #10.