NAME
SHA256_Init, SHA256_Update, SHA256_Final − Secure Hash Algorithm

SYNOPSIS
```
 #include <openssl/sha.h>

 int SHA256_Init(SHA256_CTX *c);
 int SHA256_Update(SHA256_CTX *c, const void *data, size_t len);
 int SHA256_Final(unsigned char *md, SHA256_CTX *c);
```

DESCRIPTION
SHA−256 (Secure Hash Algorithm) is a cryptographic hash function with a 256 bit output.

The following functions may be used if the message is not completely stored in memory:

SHA256_Init() initializes a **SHA256_CTX** structure.

SHA256_Update() can be called repeatedly with chunks of the message to be hashed (**len** bytes at **data**).

SHA256_Final() places the message digest in **md**, which must have space for SHA256_DIGEST_LENGTH == 32 bytes of output, and erases the **SHA256_CTX**.

RETURN VALUES
SHA256_Init(), **SHA256_Update()** and **SHA256_Final()** return 1 for success, 0 otherwise.

COPYRIGHT

NAME
closedir − close a directory

LIBRARY
Standard C library (*libc*, −*lc*)

SYNOPSIS
#include <sys/types.h>
#include <dirent.h>

int closedir(DIR ****dirp*);**

DESCRIPTION
The **closedir**() function closes the directory stream associated with *dirp*. A successful call to **closedir**() also closes the underlying file descriptor associated with *dirp*. The directory stream descriptor *dirp* is not available after this call.

RETURN VALUE
The **closedir**() function returns 0 on success. On error, −1 is returned, and *errno* is set to indicate the error.

ERRORS
EBADF
Invalid directory stream descriptor *dirp*.

ATTRIBUTES
For an explanation of the terms used in this section, see **attributes**(7).

Interface	**Attribute**	**Value**
closedir()	Thread safety	MT-Safe

STANDARDS
POSIX.1-2008.

HISTORY
POSIX.1-2001, SVr4, 4.3BSD.

SEE ALSO
close(2), **opendir**(3), **readdir**(3), **rewinddir**(3), **scandir**(3), **seekdir**(3), **telldir**(3)

NAME

fclose − close a stream

LIBRARY

Standard C library (*libc*, −*lc*)

SYNOPSIS

#include <stdio.h>

int fclose(FILE ****stream*);**

DESCRIPTION

The **fclose**() function flushes the stream pointed to by *stream* (writing any buffered output data using **fflush**(3)) and closes the underlying file descriptor.

RETURN VALUE

Upon successful completion, 0 is returned. Otherwise, **EOF** is returned and *errno* is set to indicate the error. In either case, any further access (including another call to **fclose**()) to the stream results in undefined behavior.

ERRORS

EBADF
The file descriptor underlying *stream* is not valid.

The **fclose**() function may also fail and set *errno* for any of the errors specified for the routines **close**(2), **write**(2), or **fflush**(3).

ATTRIBUTES

For an explanation of the terms used in this section, see **attributes**(7).

Interface	**Attribute**	**Value**
fclose()	Thread safety	MT-Safe

STANDARDS

C11, POSIX.1-2008.

HISTORY

C89, POSIX.1-2001.

NOTES

Note that **fclose**() flushes only the user-space buffers provided by the C library. To ensure that the data is physically stored on disk the kernel buffers must be flushed too, for example, with **sync**(2) or **fsync**(2).

SEE ALSO

close(2), **fcloseall**(3), **fflush**(3), **fileno**(3), **fopen**(3), **setbuf**(3)

NAME

clearerr, feof, ferror, fileno − check and reset stream status

SYNOPSIS

#include <stdio.h>

void clearerr(FILE ****stream*);**
int feof(FILE ****stream*);**
int ferror(FILE ****stream*);**
int fileno(FILE ****stream*);**

DESCRIPTION

The function **clearerr**() clears the end-of-file and error indicators for the stream pointed to by *stream*.

The function **feof**() tests the end-of-file indicator for the stream pointed to by *stream*, returning non-zero if it is set. The end-of-file indicator can only be cleared by the function **clearerr**().

The function **ferror**() tests the error indicator for the stream pointed to by *stream*, returning non-zero if it is set. The error indicator can only be reset by the **clearerr**() function.

The function **fileno**() examines the argument *stream* and returns its integer descriptor.

For non-locking counterparts, see **unlocked_stdio**(3).

ERRORS

These functions should not fail and do not set the external variable *errno*. (However, in case **fileno**() detects that its argument is not a valid stream, it must return −1 and set *errno* to **EBADF**.)

CONFORMING TO

The functions **clearerr**(), **feof**(), and **ferror**() conform to C89 and C99.

SEE ALSO

open(2), **fdopen**(3), **stdio**(3), **unlocked_stdio**(3)

NAME

fopen, fdopen, fileno – stream open functions

SYNOPSIS

#include <stdio.h>

FILE *fopen(const char ****path*, const char ****mode***);**
FILE *fdopen(int *fildes***, const char ****mode***);**
int fileno(FILE ****stream*);**
int fclose(FILE ****stream*);**

DESCRIPTION

The **fopen** function opens the file whose name is the string pointed to by *path* and associates a stream with it.

The argument *mode* points to a string beginning with one of the following sequences (Additional characters may follow these sequences.):

r	Open text file for reading. The stream is positioned at the beginning of the file.
r+	Open for reading and writing. The stream is positioned at the beginning of the file.
w	Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
w+	Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
a	Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
a+	Open for reading and appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.

The **fdopen** function associates a stream with the existing file descriptor, *fildes*. The *mode* of the stream (one of the values "r", "r+", "w", "w+", "a", "a+") must be compatible with the mode of the file descriptor. The file position indicator of the new stream is set to that belonging to *fildes*, and the error and end-of-file indicators are cleared. Modes "w" or "w+" do not cause truncation of the file. The file descriptor is not dup'ed, and will be closed when the stream created by **fdopen** is closed. The result of applying **fdopen** to a shared memory object is undefined.

The function **fileno**() examines the argument *stream* and returns its integer descriptor.

The **fclose**() function flushes the stream pointed to by *stream* (writing any buffered output data using **fflush**(3)) and closes the underlying file descriptor.

RETURN VALUE

Upon successful completion **fopen**, **fdopen** and **freopen** return a **FILE** pointer. Otherwise, **NULL** is returned and the global variable *errno* is set to indicate the error. Upon successful completion of **fclose**, 0 is returned. Otherwise, **EOF** is returned and *errno* is set to indicate the error.

ERRORS

EINVAL
The *mode* provided to **fopen**, **fdopen**, or **freopen** was invalid.

EBADF
The file descriptor underlying *stream* passed to **fclose** is not valid.

The **fopen**, **fdopen** and **freopen** functions may also fail and set *errno* for any of the errors specified for the routine **malloc**(3).

The **fopen** function may also fail and set *errno* for any of the errors specified for the routine **open**(2).

The **fdopen** function may also fail and set *errno* for any of the errors specified for the routine **fcntl**(2).

NAME

fread, fwrite – binary stream input/output

LIBRARY

Standard C library (*libc*, −*lc*)

SYNOPSIS

#include <stdio.h>

size_t fread(void *ptr***[restrict .***size* *** .***nmemb***],**
size_t *size***, size_t** *nmemb***,**
FILE *restrict *stream***);**
size_t fwrite(const void *ptr***[restrict .***size* *** .***nmemb***],**
size_t *size***, size_t** *nmemb***,**
FILE *restrict *stream***);**

DESCRIPTION

The function **fread**() reads *nmemb* items of data, each *size* bytes long, from the stream pointed to by *stream*, storing them at the location given by *ptr*.

The function **fwrite**() writes *nmemb* items of data, each *size* bytes long, to the stream pointed to by *stream*, obtaining them from the location given by *ptr*.

For nonlocking counterparts, see **unlocked_stdio**(3).

RETURN VALUE

On success, **fread**() and **fwrite**() return the number of items read or written. This number equals the number of bytes transferred only when *size* is 1. If an error occurs, or the end of the file is reached, the return value is a short item count (or zero).

The file position indicator for the stream is advanced by the number of bytes successfully read or written.

fread() does not distinguish between end-of-file and error, and callers must use **feof**(3) and **ferror**(3) to determine which occurred.

ATTRIBUTES

For an explanation of the terms used in this section, see **attributes**(7).

Interface	Attribute	Value
fread(), **fwrite**()	Thread safety	MT-Safe

STANDARDS

C11, POSIX.1-2008.

HISTORY

POSIX.1-2001, C89.

EXAMPLES

The program below demonstrates the use of **fread**() by parsing /bin/sh ELF executable in binary mode and printing its magic and class:

```
$ ./a.out ELF magic: 0x7f454c46 Class: 0x02
```

Program source

```
#include <stdio.h> #include <stdlib.h> #define ARRAY_SIZE(arr) (sizeof(arr) / sizeof((arr)[0])) int
main(void) {
  FILE          *fp;
  size_t        ret;
  unsigned char  buffer[4];
  fp = fopen("/bin/sh", "rb");
  if (!fp) {
    perror("fopen");
    return EXIT_FAILURE;
  }
ret = fread(buffer, sizeof(*buffer), ARRAY_SIZE(buffer), fp);
if (ret != ARRAY_SIZE(buffer)) {
    fprintf(stderr, "fread() failed: %zu\n", ret);
    exit(EXIT_FAILURE);
}
printf("ELF magic: %#04x%02x%02x%02x\n", buffer[0], buffer[1],
       buffer[2], buffer[3]);
ret = fread(buffer, 1, 1, fp);
if (ret != 1) {
    fprintf(stderr, "fread() failed: %zu\n", ret);
    exit(EXIT_FAILURE);
}
printf("Class: %#04x\n", buffer[0]);
fclose(fp);
exit(EXIT_SUCCESS); }
```

SEE ALSO

read(2), **write**(2), **feof**(3), **ferror**(3), **unlocked_stdio**(3)

NAME

opendir – open a directory / readdir – read a directory

SYNOPSIS

#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char ****name*);**
int closedir(DIR ****dirp*);**
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. On error, NULL is returned, and *errno* is set appropriately.

DESCRIPTION closedir

The **closedir()** function closes the directory stream associated with *dirp*. A successful call to **closedir()** also closes the underlying file descriptor associated with *dirp*. The directory stream descriptor *dirpP is not available after this call.*

RETURN VALUE

The **closedir()** function returns 0 on success. On error, −1 is returned, and *errno* is set appropriately.

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. It is safe to use **readdir()** inside threads if the pointers passed as *dir* are created by distinct calls to **opendir()**.

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 */
    char        d_name[256];    /* filename */
};
```

RETURN VALUE

On success, **readdir**() returns a pointer to a *dirent* structure. (This structure may be statically allocated; do not attempt to **free**(3) it.)

If the end of the directory stream is reached, NULL is returned and *errno* is not changed. If an error occurs, NULL is returned and *errno* is set appropriately. To distinguish end of stream and from an error, set *errno* to zero before calling **readdir**() and then check the value of *errno* if NULL is returned.

ERRORS

EACCES
Permission denied.

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

ENOTDIR
name is not a directory.

NAME

readdir − read a directory

LIBRARY

Standard C library (*libc*, −*lc*)

SYNOPSIS

#include <dirent.h>

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

DESCRIPTION

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

In the glibc implementation, the *dirent* structure is defined as follows:

```
struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; see below */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* Null−terminated filename */ };
```

The only fields in the *dirent* structure that are mandated by POSIX.1 are *d_name* and *d_ino*. The other fields are unstandardized, and not present on all systems; see NOTES below for some further details.

The fields of the *dirent* structure are as follows:

d_ino This is the inode number of the file.

d_off The value returned in *d_off* is the same as would be returned by calling **telldir**(3) at the current position in the directory stream. Be aware that despite its type and name, the *d_off* field is seldom any kind of directory offset on modern filesystems. Applications should treat this field as an opaque value, making no assumptions about its contents; see also **telldir**(3).

d_reclen
This is the size (in bytes) of the returned record. This may not match the size of the structure definition shown above; see NOTES.

d_type This field contains a value indicating the file type, making it possible to avoid the expense of calling **lstat**(2) if further actions depend on the type of the file.

When a suitable feature test macro is defined (**_DEFAULT_SOURCE** since glibc 2.19, or **_BSD_SOURCE** on glibc 2.19 and earlier), glibc defines the following macro constants for the value returned in *d_type*:

DT_BLK	This is a block device.
DT_CHR	This is a character device.
DT_DIR	This is a directory.
DT_FIFO	This is a named pipe (FIFO).
DT_LNK	This is a symbolic link.
DT_REG	This is a regular file.
DT_SOCK	This is a UNIX domain socket.
DT_UNKNOWN	The file type could not be determined.

Currently, only some filesystems (among them: Btrfs, ext2, ext3, and ext4) have full support for returning the file type in *d_type*. All applications must properly handle a return of

DT_UNKNOWN.

d_name
This field contains the null terminated filename. *See NOTES*.

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

RETURN VALUE

On success, **readdir**() returns a pointer to a *dirent* structure. (This structure may be statically allocated; do not attempt to **free**(3) it.)

If the end of the directory stream is reached, NULL is returned and *errno* is not changed. If an error occurs, NULL is returned and *errno* is set to indicate the error. **To distinguish end of stream from an error, set errno to zero before calling readdir() and then check the value of errno if NULL is returned.**

ERRORS

EBADF
Invalid directory stream descriptor *dirp*.

ATTRIBUTES

For an explanation of the terms used in this section, see **attributes**(7).

Interface	Attribute	Value
readdir()	Thread safety	MT-Unsafe race:dirstream

In the current POSIX.1 specification (POSIX.1-2008), **readdir**() is not required to be thread-safe. However, in modern implementations (including the glibc implementation), concurrent calls to **readdir**() that specify different directory streams are thread-safe. In cases where multiple threads must read from the same directory stream, using **readdir**() with external synchronization is still preferable to the use of the deprecated **readdir_r**(3) function. It is expected that a future version of POSIX.1 will require that **readdir**() be thread-safe when concurrently employed on different directory streams.

VERSIONS

Only the fields *d_name* and (as an XSI extension) *d_ino* are specified in POSIX.1. Other than Linux, the *d_type* field is available mainly only on BSD systems.

The d_name field

The *dirent* structure definition shown above is taken from the glibc headers, and shows the *d_name* field with a fixed size.

Warning: applications should avoid any dependence on the size of the *d_name* field. POSIX defines it as *char d_name[]*, a character array of unspecified size, with at most **NAME_MAX** characters preceding the terminating null byte ('\0').

POSIX.1 explicitly notes that this field should not be used as an lvalue. The standard also notes that the use of *sizeof(d_name)* is incorrect; use *strlen(d_name)* instead. (On some systems, this field is defined as *char d_name[1]*!) By implication, the use *sizeof(struct dirent)* to capture the size of the record including the size of *d_name* is also incorrect.

NOTES

A directory stream is opened using **opendir**(3).

The order in which filenames are read by successive calls to **readdir**() depends on the filesystem implementation; it is unlikely that the names will be sorted in any fashion.

SEE ALSO

getdents(2), **read**(2), **closedir**(3), **dirfd**(3), **ftw**(3), **offsetof**(3), **opendir**(3), **readdir_r**(3), **rewinddir**(3), **scandir**(3), **seekdir**(3), **telldir**(3)