
*Full credit is given to the above companies including the OS that this PDF file was generated!*

## *Red Hat Enterprise Linux Release 9.2 Manual Pages on 'close.3p' command*

***$ man close.3p***

CLOSE(3P) POSIX Programmer's Manual CLOSE(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

close ? close a file descriptor

SYNOPSIS

#include <unistd.h>

int close(int fildes);

DESCRIPTION

The close() function shall deallocate the file descriptor indicated by fildes. To deallocate means to make the file descriptor available for return by subsequent calls to open() or other functions that allocate file descriptors. All outstanding record locks owned by the process on the file associated with the file descriptor shall be removed (that is, unlocked).

If close() is interrupted by a signal that is to be caught, it shall return -1 with errno set to [EINTR] and the state of fildes is unspeci? fied. If an I/O error occurred while reading from or writing to the file system during close(), it may return -1 with errno set to [EIO]; if this error is returned, the state of fildes is unspecified.

When all file descriptors associated with a pipe or FIFO special file are closed, any data remaining in the pipe or FIFO shall be discarded.

When all file descriptors associated with an open file description have been closed, the open file description shall be freed.

If the link count of the file is 0, when all file descriptors associ? ated with the file are closed, the space occupied by the file shall be freed and the file shall no longer be accessible.

If a STREAMS-based fildes is closed and the calling process was previ? ously registered to receive a SIGPOLL signal for events associated with that STREAM, the calling process shall be unregistered for events asso? ciated with the STREAM. The last close() for a STREAM shall cause the STREAM associated with fildes to be dismantled. If O_NONBLOCK is not set and there have been no signals posted for the STREAM, and if there is data on the module's write queue, close() shall wait for an unspeci? fied time (for each module and driver) for any output to drain before dismantling the STREAM. The time delay can be changed via an I_SET? CLTIME ioctl() request. If the O_NONBLOCK flag is set, or if there are any pending signals, close() shall not wait for output to drain, and shall dismantle the STREAM immediately.

If the implementation supports STREAMS-based pipes, and fildes is asso? ciated with one end of a pipe, the last close() shall cause a hangup to occur on the other end of the pipe. In addition, if the other end of the pipe has been named by fattach(), then the last close() shall force the named end to be detached by fdetach(). If the named end has no open file descriptors associated with it and gets detached, the STREAM associated with that end shall also be dismantled.

If fildes refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal flushes all queued input and output.

If fildes refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message may be sent to the master.

When there is an outstanding cancelable asynchronous I/O operation against fildes when close() is called, that I/O operation may be can? celed. An I/O operation that is not canceled completes as if the close() operation had not yet occurred. All operations that are not canceled shall complete as if the close() blocked until the operations completed. The close() operation itself need not block awaiting such I/O completion. Whether any I/O operation is canceled, and which I/O operation may be canceled upon close(), is implementation-defined.

If a memory mapped file or a shared memory object remains referenced at the last close (that is, a process has it mapped), then the entire con? tents of the memory object shall persist until the memory object be? comes unreferenced. If this is the last close of a memory mapped file or a shared memory object and the close results in the memory object becoming unreferenced, and the memory object has been unlinked, then the memory object shall be removed.

If fildes refers to a socket, close() shall cause the socket to be de? stroyed. If the socket is in connection-mode, and the SO_LINGER option is set for the socket with non-zero linger time, and the socket has un? transmitted data, then close() shall block for up to the current linger interval until all data is transmitted.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, -1 shall be returned and errno set to indicate the error.

ERRORS

The close() function shall fail if:

EBADF The fildes argument is not a open file descriptor.

EINTR The close() function was interrupted by a signal.

The close() function may fail if:

EIO An I/O error occurred while reading from or writing to the file system.

The following sections are informative.

EXAMPLES

Reassigning a File Descriptor

The following example closes the file descriptor associated with stan? dard output for the current process, re-assigns standard output to a new file descriptor, and closes the original file descriptor to clean up. This example assumes that the file descriptor 0 (which is the de? scriptor for standard input) is not closed.

```
#include <unistd.h>
...
int pfd;
...
close(1);
dup(pfd);
close(pfd);
...
```

Incidentally, this is exactly what could be achieved using:

```
dup2(pfd, 1);
close(pfd);
```

Closing a File Descriptor

In the following example, close() is used to close a file descriptor after an unsuccessful attempt is made to associate that file descriptor with a stream.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define LOCKFILE "/etc/ptmp"
...
int pfd;
FILE *fpfd;
...
if ((fpfd = fdopen (pfd, "w")) == NULL) {
    close(pfd);
    unlink(LOCKFILE);
    exit(1);
}
```

...

APPLICATION USAGE

An application that had used the stdio routine fopen() to open a file should use the corresponding fclose() routine rather than close(). Once a file is closed, the file descriptor no longer exists, since the integer corresponding to it no longer refers to a file.

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls close() on an arbitrary integer risks non-conforming behavior, and close() can only portably be used on file descriptor values that the application has obtained through explicit actions, as well as the three file descriptors corresponding to the standard file streams. In multi-threaded parent applications, the practice of calling close() in a loop after fork() and before an exec call in order to avoid a race condition of leaking an unintended file descriptor into a child process, is therefore unsafe, and the race should instead be combatted by opening all file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited across exec.

Usage of close() on file descriptors STDIN_FILENO, STDOUT_FILENO, or STDERR_FILENO should immediately be followed by an operation to reopen these file descriptors. Unexpected behavior will result if any of these file descriptors is left in a closed state (for example, an [EBADF] er? ror from perror()) or if an unrelated open() or similar call later in the application accidentally allocates a file to one of these well-known file descriptors. Furthermore, a close() followed by a reopen op? eration (e.g., open(), dup(), etc.) is not atomic; dup2() should be used to change standard file descriptors.

RATIONALE

The use of interruptible device close routines should be discouraged to avoid problems with the implicit closes of file descriptors by exec and exit(). This volume of POSIX.1?2017 only intends to permit such behav? ior by specifying the [EINTR] error condition.

Note that the requirement for close() on a socket to block for up to the current linger interval is not conditional on the O_NONBLOCK set? ting.

The standard developers rejected a proposal to add closefrom() to the standard. Because the standard permits implementations to use inherited file descriptors as a means of providing a conforming environment for the child process, it is not possible to standardize an interface that closes arbitrary file descriptors above a certain value while still guaranteeing a conforming environment.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6, STREAMS, dup(), exec, exit(), fattach(), fclose(), fde? tach(), fopen(), fork(), ioctl(), open(), perror(), unlink()

The Base Definitions volume of POSIX.1?2017, <unistd.h>