close - Man Page

close a file descriptor

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.

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 unspecified. 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 associated 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 previously registered to receive a SIGPOLL signal for events associated with that STREAM, the calling process shall be unregistered for events associated 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 unspecified time (for each module and driver) for any output to drain before

close - Man Page

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
associated 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
canceled. 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
contents of the memory object shall persist until the memory object
becomes 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
destroyed. 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
untransmitted 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

close - Man Page

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 При чтении или записи в файловую систему произошла ошибка ввода-вывода.

Следующие разделы являются информативными.

Примеры

Переназначение файлового дескриптора

Следующий пример закрывает файловый дескриптор, связанный со стандартным выводом для текущего процесса, переназначает стандартный вывод новому файловому дескриптору и закрывает исходный файловый дескриптор для очистки. В этом примере предполагается, что файловый дескриптор 0 (который является дескриптором для стандартного ввода) не закрыт.

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

Кстати, это именно то, что может быть достигнуто с помощью:

```
dup2 (pfd, 1);
закрыть (pfd);
```

Закрытие файлового дескриптора

В следующем примере *close*() используется для закрытия файлового дескриптора после неудачной попытки связать этот файловый дескриптор с потоком.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd;
ФАЙЛ *fpfd;
...
```

close - Man Page

```
  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]** error 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 operation (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 behavior 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 setting.

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

close - Man Page

Future Directions

None.

See Also

Section 2.6, *STREAMS*, dup(), exec, exit(), fattach(), fclose(),
fdetach(), fopen(), fork(), ioctl(), open(), perror(), unlink()

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

Copyright

Portions of this text are reprinted and reproduced in electronic form
from IEEE Std 1003.1-2017, Standard for Information Technology --
Portable Operating System Interface (POSIX), The Open Group Base
Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute
of Electrical and Electronics Engineers, Inc and The Open Group. In the
event of any discrepancy between this version and the original IEEE and
The Open Group Standard, the original IEEE and The Open Group Standard is
the referee document. The original Standard can be obtained online at
http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most
likely to have been introduced during the conversion of the source files
to man page format. To report such errors, see
https://www.kernel.org/doc/man-pages/reporting_bugs.html .

Referenced By

aio_error(3p), aio_read(3p), aio_return(3p), aio_write(3p), connect(3p),
dup(3p), exec(3p), fclose(3p), fcntl(3p), getsockopt(3p), ioctl(3p),
lio_listio(3p), lockf(3p), open(3p), posix_spawn(3p),
posix_spawn_file_actions_addclose(3p), posix_typed_mem_open(3p),
shm_open(3p), shm_unlink(3p), stropts.h(0p), unistd.h(0p), unlink(3p).

2017 IEEE/The Open Group POSIX Programmer's Manual

close - Man Page

Главная   Блог   0 нас

close - Man Page