

# *File descriptor*

In [Unix](#) and [Unix-like](#) computer operating systems, a **file descriptor** (**FD**, less frequently **fil**des) is a process-unique identifier ([handle](#)) for a [file](#) or other [input/output resource](#), such as a [pipe](#) or [network socket](#).

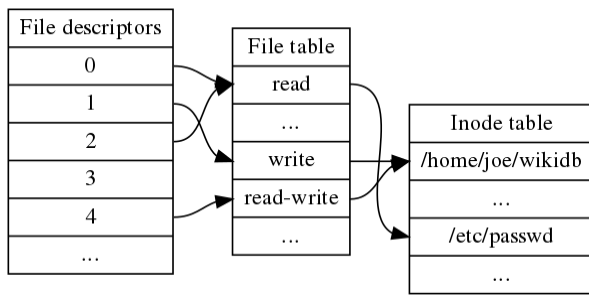
File descriptors typically have non-negative integer values, with negative values being reserved to indicate "no value" or error conditions.

File descriptors are a part of the [POSIX API](#). Each Unix [process](#) (except perhaps [daemons](#)) should have three standard POSIX file descriptors, corresponding to the three [standard streams](#):

Integer value	Name	< <a href="#">unistd.h</a> > symbolic constant <sup>[1]</sup>	< <a href="#">stdio.h</a> > file stream <sup>[2]</sup>
0	<a href="#">Standard input</a>	STDIN_FILENO	stdin
1	<a href="#">Standard output</a>	STDOUT_FILENO	stdout
2	<a href="#">Standard error</a>	STDERR_FILENO	stderr

## Overview

---



File descriptors for a single process, file table and [inode](#) table. Note that multiple file descriptors can refer to the same file table entry (e.g., as a result of the [dup](#) system call<sup>[3]:104</sup>) and that multiple file table entries can in turn refer to the same inode (if it has been opened multiple times; the table is still simplified because it represents inodes by file names, even though an inode can have [multiple names](#)). File descriptor 3 does not refer to anything in the file table, signifying that it has been closed.

In the traditional implementation of Unix, file descriptors index into a per-process **file descriptor table** maintained by the kernel, that in turn indexes into a system-wide table of files opened by all processes, called the **file table**. This table records the *mode* with which the file (or other resource) has been opened: for reading, writing, appending, and possibly other modes. It also indexes into a third table called the [inode table](#) that describes the actual underlying files.<sup>[3]</sup> To perform input or output, the process passes the file descriptor to the kernel through a [system call](#), and the kernel will access the file on behalf of the process. The process does not have direct access to the file or inode tables.

On [Linux](#), the set of file descriptors open in a process can be accessed under the path

`/proc/PID/fd/`, where PID is the [process identifier](#). File descriptor `/proc/PID/fd/0` is `stdin`, `/proc/PID/fd/1` is `stdout`, and `/proc/PID/fd/2` is `stderr`. As a shortcut to these, any running process can also access *its own* file descriptors through the folders `/proc/self/fd` and `/dev/fd`.<sup>[4]</sup>

In [Unix-like](#) systems, file descriptors can refer to any [Unix file type](#) named in a file system. As well as regular files, this includes [directories](#), [block](#) and [character devices](#) (also called "special files"), [Unix domain sockets](#), and [named pipes](#). File descriptors can also refer to other objects that do not normally exist in the file system, such as [anonymous pipes](#) and [network sockets](#).

The FILE data structure in the [C standard I/O library](#) usually includes a low level file descriptor for the object in question on Unix-like systems. The overall data structure provides additional abstraction and is instead known as a *file handle*.

## Operations on file descriptors

## Operations on file descriptors

---

The following lists typical operations on file descriptors on modern [Unix-like](#) systems. Most of these functions are declared in the `<unistd.h>` header, but some are in the `<fcntl.h>` header instead.

### Creating file descriptors

- [open\(\)](#)
- [creat\(\)](#) <sup>[5]</sup>
- [socket\(\)](#)
- [accept\(\)](#)
- [socketpair\(\)](#)
- [pipe\(\)](#)
- [epoll\\_create\(\)](#) (Linux)
- [signalfd\(\)](#) (Linux)
- [eventfd\(\)](#) (Linux)
- [timerfd\\_create\(\)](#) (Linux)
- [memfd\\_create\(\)](#) (Linux)
- [userfaultfd\(\)](#) (Linux)
- [fanotify\\_init\(\)](#) (Linux)
- [inotify\\_init\(\)](#) (Linux)
- [clone\(\)](#) (with flag `CLONE_PIDFD`, Linux)
- [pidfd\\_open\(\)](#) (Linux)
- [open\\_by\\_handle\\_at\(\)](#) (Linux)

### Deriving file descriptors

- [dirfd\(\)](#)
- [fileno\(\)](#)

### Operations on a single file descriptor

- `read()`, `write()`
- `readv()`, `writew()`
- `pread()`, `pwrite()`
- `recv()`, `send()`
- `recvfrom()`, `sendto()`
- `recvmsg()`, `sendmsg()` (also used for sending FDs to other processes over a Unix domain socket)
- `recvmsg()`, `sendmsg()`
- `lseek()`, `llseek()`
- `fstat()`
- `fstatvfs()`
- `fchmod()`
- `fchown()`
- `ftruncate()`
- `fsync()`
- `fdatasync()`
- `fdopendir()`
- `fgetxattr()`, `fsetxattr()` (Linux)
- `flistxattr()`, `fremovexattr()` (Linux)
- `statx` (Linux)
- `setns` (Linux)
- `vmsplice()` (Linux)
- `pidfd_send_signal()` (Linux)
- `waitid()` (with `P_PIDFD` ID type, Linux)
- `fdopen()` (stdio function: converts file descriptor to `FILE*`)
- `dprintf()` (stdio function: prints to file descriptor)

## Operations on multiple file descriptors

- `select()`, `pselect()`
- `poll()`, `ppoll()`
- `epoll_wait()`, `epoll_pwait()`, `epoll_pwait2()` (Linux, takes a single `epoll` filedescriptor to wait on many other file descriptors)
- `epoll_ctl()` (for Linux)
- `kqueue()` (for BSD-based systems).
- `sendfile()`
- `splice()`, `tee()` (for Linux)
- `copy_file_range()` (for Linux)
- `close_range()` (for Linux)<sup>[6]</sup>

## Operations on the file descriptor table

The `fcntl()` function is used to perform various operations on a file descriptor, depending on the command argument passed to it. There are commands to get and set attributes associated with a file descriptor, including `F_GETFD`, `F_SETFD`, `F_GETFL` and `F_SETFL`.

- `close()`
- `closefrom()` (BSD and Solaris only; deletes all file descriptors greater than or equal to specified number)
- `dup()` (duplicates an existing file descriptor guaranteeing to be the lowest number available file descriptor)
- `dup2()`, `dup3()` (Close `fd1` if necessary, and make file descriptor `fd1` point to the open file of `fd2`)
- `fcntl (F_DUPFD)`

## Operations that modify process state

- `fchdir()` (sets the process's current working directory based on a directory file descriptor)
- `mmap()` (maps ranges of a file into the process's address space)

## File locking

- `flock()`

- `fcntl()` (`F_GETLK`, `F_SETLK` and `F_SETLKW`)
- `lockf()`

## Sockets

- `connect()`
- `bind()`
- `listen()`
- `accept()` (creates a new file descriptor for an incoming connection)
- `getsockname()`
- `getpeername()`
- `getsockopt()`
- `setsockopt()`
- `shutdown()` (shuts down one or both halves of a full duplex connection)

## Miscellaneous

- `ioctl()` (a large collection of miscellaneous operations on a single file descriptor, often associated with a device)

## Upcoming operations

---

A series of new operations on file descriptors has been added to many modern Unix-like systems, as well as numerous C libraries, to be standardized in a future version of [POSIX](#).<sup>[7]</sup> The `at` suffix signifies that the function takes an additional first argument supplying a file descriptor from which [relative paths](#) are resolved, the forms lacking the `at` suffix thus becoming equivalent to passing a file descriptor corresponding to the current [working directory](#). The purpose of these new operations is to defend against a certain class of [TOCTOU](#) attacks.

- `openat()`
- `faccessat()`
- `fchmodat()`
- `fchownat()`
- `fstatat()`

- `futimesat()`
- `linkat()`
- `mkdirat()`
- `mknodat()`
- `readlinkat()`
- `renameat()`
- `symlinkat()`
- `unlinkat()`
- `mkfifoat()`
- `fdopendir()`

## File descriptors as capabilities

---

Unix file descriptors behave in many ways as [capabilities](#). They can be passed between processes across [Unix domain sockets](#) using the `sendmsg()` system call. Note, however, that what is actually passed is a reference to an "open file description" that has mutable state (the file offset, and the file status and access flags). This complicates the secure use of file descriptors as capabilities, since when programs share access to the same open file description, they can interfere with each other's use of it by changing its offset or whether it is blocking or non-blocking, for example.<sup>[8][9]</sup> In operating systems that are specifically designed as capability systems, there is very rarely any mutable state associated with a capability itself.

A Unix process' file descriptor table is an example of a [C-list](#).

## See also

---

- [fuser \(Unix\)](#)
- [lsof](#)
- [File Control Block](#) (FCB) - an alternative scheme in CP/M and early versions of DOS

## References

---

1. *The Open Group. "The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008, 2016 Edition" (htt*

- <p://pubs.opengroup.org/onlinepubs/9699919799/basedefs/unistd.h.html>) . Retrieved 2017-09-21.
2. The Open Group. "The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008, 2016 Edition" ([http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/stdio.h.html](p://pubs.opengroup.org/onlinepubs/9699919799/basedefs/stdio.h.html)) . <stdio.h>. Retrieved 2017-09-21.
  3. Bach, Maurice J. (1986). *The Design of the UNIX Operating System* (<https://archive.org/details/designofunixoper00bach/page/92>) (8 ed.). Prentice-Hall. pp. 92–96 (<https://archive.org/details/designofunixoper00bach/page/92>) . ISBN 9780132017992.
  4. "Devices - What does the output of 'll /Proc/Self/Fd/' (From 'll /Dev/Fd') mean?" (<https://unix.stackexchange.com/questions/676683/what-does-the-output-of-ll-proc-self-fd-from-ll-dev-fd-mean>) .
  5. The Open Group. "The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008, 2018 Edition – creat" (<http://pubs.opengroup.org/onlinepubs/9699919799/functions/creat.html>) . Retrieved 2019-04-11.
  6. Stephen Kitt, Michael Kerrisk. "close\_range(2) – Linux manual page" ([https://man7.org/linux/man-pages/man2/close\\_range.2.html](https://man7.org/linux/man-pages/man2/close_range.2.html)) . Retrieved 2021-03-22.
  7. *Extended API Set, Part 2* (<https://www2.opengroup.org/ogsys/catalog/c063>) . The Open Group. October 2006. ISBN 1931624674.
  8. Brinkmann, Marcus (2009-02-04). "Building a bridge: library API's and file descriptors?" (<https://archive.today/20120730163337/http://www.eros-os.org/pipermail/cap-talk/2009-February/012137.html>) . cap-talk. Archived from *the original* (<http://www.eros-os.org/pipermail/cap-talk/2009-February/012137.html>) on 2012-07-30. Retrieved 2017-09-21.
  9. de Boyne Pollard, Jonathan (2007). "Don't set shared file descriptors to non-blocking I/O mode" (<http://jdeb.eu/FGA/dont-set-shared-file-descriptors-to-non-blocking-mode.html>) . Retrieved 2017-09-21.

Retrieved from

["https://en.wikipedia.org/w/index.php?title=File\\_descriptor&oldid=1115665100"](https://en.wikipedia.org/w/index.php?title=File_descriptor&oldid=1115665100)



WIKIPEDIA

---