



Linux Ubuntu 26.04 Manual Pages on command 'exit.2'

\$ man exit.2

_exit(2) System Calls Manual _exit(2)

NAME

_exit, _Exit - terminate the calling process

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

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

```
[[noreturn]] void _exit(int status);
```

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

```
[[noreturn]] void _Exit(int status);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

_Exit():

```
_ISOC99_SOURCE || _POSIX_C_SOURCE >= 200112L
```

DESCRIPTION

_exit() terminates the calling process "immediately". Any open file descriptors belonging to the process are closed. Any children of the process are inherited by init(1) (or by the nearest "subreaper" process as defined through the use of the prctl(2) PR_SET_CHILD_SUB? REAPER operation). The process's parent is sent a SIGCHLD signal.

The value status & 0xFF is returned to the parent process as the process's exit status, and can be collected by the parent using one of the wait(2) family of calls.

The function _Exit() is equivalent to _exit().

RETURN VALUE

These functions do not return.

STANDARDS

`_exit()`

POSIX.1-2024.

`_Exit()`

C11, POSIX.1-2024.

HISTORY

`_exit()`

4.3BSD, SVr4, POSIX.1-1988.

`_Exit()`

C99, POSIX.1-2001.

NOTES

For a discussion on the effects of an `exit`, the transmission of exit status, zombie processes, signals sent, and so on, see `exit(3)`.

The function `_exit()` is like `exit(3)`, but does not call any functions registered with `atexit(3)` or `on_exit(3)`. Open `stdio(3)` streams are not flushed. On the other hand, `_exit()` does close open file descriptors, and this may cause an unknown delay, waiting for pending output to finish. If the delay is undesired, it may be useful to call functions like `tcflush(3)` before calling `_exit()`. Whether any pending I/O is canceled, and which pending I/O may be canceled upon `_exit()`, is implementation-dependent.

C library/kernel differences

The text above in DESCRIPTION describes the traditional effect of `_exit()`, which is to terminate a process, and these are the semantics specified by POSIX.1 and implemented by the C library wrapper function. On modern systems, this means termination of all threads in the process.

By contrast with the C library wrapper function, the raw Linux `_exit()` system call terminates only the calling thread, and actions such as reparenting child processes or sending `SIGCHLD` to the parent process are performed only if this is the last thread in the thread group.

Up to glibc 2.3, the `_exit()` wrapper function invoked the kernel system call of the same name. Since glibc 2.3, the wrapper function invokes `exit_group(2)`, in order to terminate all of the threads in a process.

SEE ALSO

`execve(2)`, `exit_group(2)`, `fork(2)`, `kill(2)`, `wait(2)`, `wait4(2)`, `waitpid(2)`, `atexit(3)`,

exit(3), on_exit(3), termios(3)

Linux man-pages 6.17

2026-02-08

_exit(2)