NAME

truncate, ftruncate − truncate a file to a specified length

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

```
#include <unistd.h>

int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length);
```

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

truncate():
_XOPEN_SOURCE >= 500
|| /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L
|| /* glibc <= 2.19: */ _BSD_SOURCE

ftruncate():
_XOPEN_SOURCE >= 500
|| /* Since glibc 2.3.5: */ _POSIX_C_SOURCE >= 200112L
|| /* glibc <= 2.19: */ _BSD_SOURCE

DESCRIPTION

The **truncate**() and **ftruncate**() functions cause the regular file named by *path* or referenced by *fd* to be truncated to a size of precisely *length* bytes.

If the file previously was larger than this size, the extra data is lost. If the file previously was shorter, it is extended, and the extended part reads as null bytes ('\0').

The file offset is not changed.

If the size changed, then the st_ctime and st_mtime fields (respectively, time of last status change and time of last modification; see **inode**(7)) for the file are updated, and the set-user-ID and set-group-ID mode bits may be cleared.

With **ftruncate**(), the file must be open for writing; with **truncate**(), the file must be writable.

RETURN VALUE

On success, zero is returned. On error, −1 is returned, and *errno* is set to indicate the error.

ERRORS

For **truncate**():

EACCES
Search permission is denied for a component of the path prefix, or the named file is not writable by the user. (See also**path_r esolution**(7).)

EFAULT
The argument *path* points outside the process's allocated address space.

EFBIG
The argument *length* is larger than the maximum file size. (XSI)

EINTR
While blocked waiting to complete, the call was interrupted by a signal handler; see **fcntl**(2) and **signal**(7).

EINVAL
The argument *length* is negative or larger than the maximum file size.

EIO An I/O error occurred updating the inode.

EISDIR
The named file is a directory.

ELOOP
Too many symbolic links were encountered in translating the pathname.

ENAMETOOLONG
A component of a pathname exceeded 255 characters, or an entire pathname exceeded 1023 characters.

ENOENT
The named file does not exist.

ENOTDIR
A component of the path prefix is not a directory.

EPERM
The underlying filesystem does not support extending a file beyond its current size.

EPERM
The operation was prevented by a file seal; see **fcntl**(2).

EROFS
The named file resides on a read-only filesystem.

ETXTBSY
The file is an executable file that is being executed.

For **ftruncate**() the same errors apply, but instead of things that can be wrong with *path*, we now have things that can be wrong with the file descriptor, *fd*:

EBADF
fd is not a valid file descriptor.

EBADF or **EINVAL**
fd is not open for writing.

EINVAL
fd does not reference a regular file or a POSIX shared memory object.

EINVAL or **EBADF**
The file descriptor *fd* is not open for writing. POSIX permits, and portable applications should handle, either error for this case. (Linux produces **EINVAL**.)

STANDARDS

POSIX.1-2001, POSIX.1-2008, 4.4BSD, SVr4 (these calls first appeared in 4.2BSD).

NOTES

ftruncate() can also be used to set the size of a POSIX shared memory object; see **shm_open**(3).

The details in DESCRIPTION are for XSI-compliant systems. For non-XSI-compliant systems, the POSIX standard allows two behaviors for **ftruncate**() when *length* exceeds the file length (note that **truncate**() is not specified at all in such an environment): either returning an error, or extending the file. Like most UNIX implementations, Linux follows the XSI requirement when dealing with native filesystems. However, some nonnative filesystems do not permit **truncate**() and **ftruncate**() to be used to extend a file beyond its current length: a notable example on Linux is VFAT.

The original Linux **truncate**() and **ftruncate**() system calls were not designed to handle large file offsets. Consequently, Linux 2.4 added **truncate64**() and **ftruncate64**() system calls that handle large files. However, these details can be ignored by applications using glibc, whose wrapper functions transparently employ the more recent system calls where they are available.

On some 32-bit architectures, the calling signature for these system calls differ, for the reasons described in **syscall**(2).

BUGS

A header file bug in glibc 2.12 meant that the minimum value of **_POSIX_C_SOURCE** required to expose the declaration of **ftruncate**() was 200809L instead of 200112L. This has been fixed in later glibc versions.

SEE ALSO

truncate(1), **open**(2), **stat**(2), **path_resolution**(7)