

## 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

`access`, `faccessat` — determine accessibility of a file relative to directory file descriptor

## SYNOPSIS

```
#include <unistd.h>

int access(const char *path, int amode);
int faccessat(int fd, const char *path, int amode, int flag);
```

## DESCRIPTION

The `access()` function shall check the file named by the pathname pointed to by the `path` argument for accessibility according to the bit pattern contained in `amode`, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID.

The value of `amode` is either the bitwise-inclusive OR of the access permissions to be checked (R\_OK, W\_OK, X\_OK) or the existence test (F\_OK).

If any access permissions are checked, each shall be checked individually, as described in the Base Definitions volume of POSIX.1-2008, *Section 4.4, File Access Permissions*, except that where that description refers to execute permission for a process with appropriate privileges, an implementation may indicate success for X\_OK even if execute permission is not granted to any user.

The `faccessat()` function shall be equivalent to the `access()` function, except in the case where `path` specifies a relative path. In this case the file whose accessibility is to be determined shall be located relative to the directory associated with the file descriptor `fd` instead of the current working directory. If the file descriptor was opened without O\_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the file descriptor was opened with O\_SEARCH, the function shall not perform the check.

If `faccessat()` is passed the special value AT\_FDCWD in the `fd` parameter, the current working directory shall be used and the behavior shall be identical to a call to `access()`.

Values for `flag` are constructed by a bitwise-inclusive OR of flags from the following list, defined in `<fcntl.h>`:

AT\_EACCESS

The checks for accessibility are performed using the effective user and group IDs instead of the real user and group ID as required in a call to `access()`.

## RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return -1 and set `errno` to indicate the error.

## ERRORS

These functions shall fail if:

### EACCES

Permission bits of the file mode do not permit the requested access, or search permission is denied on a component of the path prefix.

### ELOOP

A loop exists in symbolic links encountered during resolution of the `path` argument.

### ENAMETOOLONG

The length of a component of a pathname is longer than {NAME\_MAX}.

**ENOENT**

A component of *path* does not name an existing file or *path* is an empty string.

**ENOTDIR**

A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-`<slash>` character and ends with one or more trailing `<slash>` characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

**EROFS**

Write access is requested for a file on a read-only file system.

The *faccessat()* function shall fail if:

**EACCES**

*fd* was not opened with `O_SEARCH` and the permissions of the directory underlying *fd* do not permit directory searches.

**EBADF**

The *path* argument does not specify an absolute path and the *fd* argument is neither `AT_FDCWD` nor a valid file descriptor open for reading or searching.

**ENOTDIR**

The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

**EINVAL**

The value of the *amode* argument is invalid.

**ELOOP**

More than `{SYMLOOP_MAX}` symbolic links were encountered during resolution of the *path* argument.

**ENAMETOOLONG**

The length of a pathname exceeds `{PATH_MAX}`, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds `{PATH_MAX}`.

**ETXTBSY**

Write access is requested for a pure procedure (shared text) file that is being executed.

The *faccessat()* function may fail if:

**EINVAL**

The value of the *flag* argument is not valid.

*The following sections are informative.*

**EXAMPLES****Testing for the Existence of a File**

The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
#include <unistd.h>
...
int result;
const char *pathname = "/tmp/myfile";
result = access (pathname, F_OK);
```

**APPLICATION USAGE**

Additional values of *amode* other than the set defined in the description may be valid; for example, if a system has extended access controls.

The use of the `AT_EACCESS` value for *flag* enables functionality not available in *access()*.

## RATIONALE

In early proposals, some inadequacies in the *access()* function led to the creation of an *eaccess()* function because:

1. Historical implementations of *access()* do not test file access correctly when the process' real user ID is superuser. In particular, they always return zero when testing execute permissions without regard to whether the file is executable.
2. The superuser has complete access to all files on a system. As a consequence, programs started by the superuser and switched to the effective user ID with lesser privileges cannot use *access()* to test their file access permissions.

However, the historical model of *eaccess()* does not resolve problem (1), so this volume of POSIX.1-2008 now allows *access()* to behave in the desired way because several implementations have corrected the problem. It was also argued that problem (2) is more easily solved by using *open()*, *chdir()*, or one of the *exec* functions as appropriate and responding to the error, rather than creating a new function that would not be as reliable. Therefore, *eaccess()* is not included in this volume of POSIX.1-2008.

The sentence concerning appropriate privileges and execute permission bits reflects the two possibilities implemented by historical implementations when checking superuser access for *X\_OK*.

New implementations are discouraged from returning *X\_OK* unless at least one execution permission bit is set.

The purpose of the *faccessat()* function is to enable the checking of the accessibility of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *access()*, resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *faccessat()* function it can be guaranteed that the file tested for accessibility is located relative to the desired directory.

## FUTURE DIRECTIONS

None.

## SEE ALSO

*chmod()*, *fstatat()*

The Base Definitions volume of POSIX.1-2008, *Section 4.4, File Access Permissions*, **<fcntl.h>**, **<unistd.h>**

## COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1, 2013 Edition, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, Copyright (C) 2013 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. (This is POSIX.1-2008 with the 2013 Technical Corrigendum 1 applied.) 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.unix.org/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](https://www.kernel.org/doc/man-pages/reporting_bugs.html) .