

NAME

access, **eaccess**, **faccessat** - check accessibility of a file

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

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

int

```
access(const char *path, int mode);
```

int

```
eaccess(const char *path, int mode);
```

int

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

DESCRIPTION

The **access()**, **eaccess()** and **faccessat()** system calls report whether an attempt to access the file designated by their *path* in the manner described by their *mode* argument is likely to succeed. The value of *mode* is either the bitwise-inclusive OR of the desired permissions (R_OK for read permission, W_OK for write permission, and X_OK for execute / search permission) or F_OK to simply check whether the file exists.

For a number of reasons, these system calls cannot be relied upon to give a correct and definitive answer. They can at best provide an early indication of the expected outcome, to be confirmed by actually attempting the operation. For existence checks, either `stat(2)` or `lstat(2)` should be used instead. See also *SECURITY CONSIDERATIONS* below.

The **eaccess()** system call uses the effective user ID and the group access list to authorize the request; the **access()** system call uses the real user ID in place of the effective user ID, the real group ID in place of the effective group ID, and the rest of the group access list.

See the *DEFINITIONS* section of `intro(2)` for additional information on file access permissions and real vs. effective user and group IDs.

The **faccessat()** system call is equivalent to **access()** except in the case where *path* specifies a relative path. In this case the file whose accessibility is to be determined is located relative to the directory associated with the file descriptor *fd* instead of the current working directory. If **faccessat()** is passed the

special value `AT_FDCWD` in the *fd* parameter, the current working directory is used and the behavior is 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 are performed using the effective user and group IDs, like `eaaccess()`, instead of the real user and group ID, like `access()`.

AT_RESOLVE_BENEATH

Only walk paths below the directory specified by the *fd* descriptor. See the description of the `O_RESOLVE_BENEATH` flag in the `open(2)` manual page.

AT_EMPTY_PATH

If the *path* argument is an empty string, operate on the file or directory referenced by the descriptor *fd*. If *fd* is equal to `AT_FDCWD`, operate on the current working directory.

AT_SYMLINK_NOFOLLOW

If *path* names a symbolic link, access of the symbolic link is evaluated.

Even if a process's real or effective user has appropriate privileges and indicates success for `X_OK`, the file may not actually have execute permission bits set. Likewise for `R_OK` and `W_OK`.

RETURN VALUES

Upon successful completion, the value 0 is returned; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The `access()`, `eaaccess()`, and `faaccessat()` system calls may fail if:

- | | |
|----------------|---|
| [EINVAL] | The value of the <i>mode</i> argument is invalid. |
| [ENOTDIR] | A component of the path prefix is not a directory. |
| [ENAMETOOLONG] | A component of a pathname exceeded 255 characters, or an entire path name exceeded 1023 characters. |
| [ENOENT] | The named file does not exist. |
| [ELOOP] | Too many symbolic links were encountered in translating the pathname. |

- [EROFS] Write access is requested for a file on a read-only file system.
- [ETXTBSY] Write access is requested for a pure procedure (shared text) file presently being executed.
- [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.
- [EFAULT] The *path* argument points outside the process's allocated address space.
- [EIO] An I/O error occurred while reading from or writing to the file system.
- [EINTEGRITY] Corrupted data was detected while reading from the file system.

Also, the **faccessat()** system call may fail if:

- [EBADF] The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor.
- [EINVAL] The value of the *flag* argument is not valid.
- [ENOTDIR] The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a file descriptor associated with a directory.
- [ENOTCAPABLE] *path* is an absolute path, or contained a "." component leading to a directory outside of the directory hierarchy specified by *fd*, and the process is in capability mode.

SEE ALSO

chmod(2), intro(2), stat(2)

STANDARDS

The **access()** system call is expected to conform to IEEE Std 1003.1-1990 ("POSIX.1"). The **faccessat()** system call follows The Open Group Extended API Set 2 specification.

HISTORY

The **access()** function appeared in Version 7 AT&T UNIX. The **faccessat()** system call appeared in FreeBSD 8.0.

SECURITY CONSIDERATIONS

The **access()**, **eaccess()**, and **faccessat()** system calls are subject to time-of-check-to-time-of-use races and should not be relied upon for file permission enforcement purposes. Instead, applications should perform the desired action using the requesting user's credentials.