NAME

access − check real user's permissions for a file

SYNOPSIS

#include <unistd.h>

int access(const char ***_pathname_, int** _mode_**);**

DESCRIPTION

access() checks whether the calling process can access the file *pathname*. If *pathname* is a symbolic link, it is dereferenced.

The *mode* specifies the accessibility check(s) to be performed, and is either the value **F_OK**, or a mask consisting of the bitwise OR of one or more of **R_OK**, **W_OK**, and **X_OK**. **F_OK** tests for the existence of the file. **R_OK**, **W_OK**, and **X_OK** test whether the file exists and grants read, write, and execute permissions, respectively.

The check is done using the calling process's *real* UID and GID, rather than the effective IDs as is done when actually attempting an operation (e.g., **open**(2)) on the file. This allows set-user-ID programs to easily determine the invoking user's authority.

If the calling process is privileged (i.e., its real UID is zero), then an **X_OK** check is successful for a regular file if execute permission is enabled for any of the file owner, group, or other.

RETURN VALUE

On success (all requested permissions granted), zero is returned. On error (at least one bit in *mode* asked for a permission that is denied, or some other error occurred), −1 is returned, and *errno* is set appropriately.

ERRORS

access() shall fail if:

EACCES
The requested access would be denied to the file, or search permission is denied for one of the directories in the path prefix of *pathname*. (See also **path_resolution**(7).)

ELOOP
Too many symbolic links were encountered in resolving *pathname*.

ENAMETOOLONG
pathname is too long.

ENOENT
A component of *pathname* does not exist or is a dangling symbolic link.

ENOTDIR
A component used as a directory in *pathname* is not, in fact, a directory.

EROFS
Write permission was requested for a file on a read-only file system.

access() may fail if:

EFAULT
pathname points outside your accessible address space.

EINVAL
mode was incorrectly specified.

EIO An I/O error occurred.

ENOMEM
Insufficient kernel memory was available.

ETXTBSY
Write access was requested to an executable which is being executed.

CONFORMING TO

SVr4, 4.3BSD, POSIX.1-2001.

NOTES

Warning: Using **access**() to check if a user is authorized to, for example, open a file before actually doing so using **open**(2) creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it. **For this reason, the use of this system call should be avoided**.

access() returns an error if any of the access types in *mode* is denied, even if some of the other access types in *mode* are permitted.

If the calling process has appropriate privileges (i.e., is superuser), POSIX.1-2001 permits implementation to indicate success for an **X_OK** check even if none of the execute file permission bits are set. Linux does not do this.

A file is only accessible if the permissions on each of the directories in the path prefix of *pathname* grant search (i.e., execute) access. If any directory is inaccessible, then the **access**() call will fail, regardless of the permissions on the file itself.

Only access bits are checked, not the file type or contents. Therefore, if a directory is found to be writable, it probably means that files can be created in the directory, and not that the directory can be written as a file. Similarly, a DOS file may be found to be "executable," but the **execve**(2) call will still fail.

access() may not work correctly on NFS file systems with UID mapping enabled, because UID mapping is done on the server and hidden from the client, which checks permissions.

BUGS

In kernel 2.4 (and earlier) there is some strangeness in the handling of **X_OK** tests for superuser. If all categories of execute permission are disabled for a non-directory file, then the only **access**() test that returns −1 is when *mode* is specified as just **X_OK**; if **R_OK** or **W_OK** is also specified in *mode*, then **access**() returns 0 for such files. Early 2.6 kernels (up to and including 2.6.3) also behaved in the same way as kernel 2.4.

In kernels before 2.6.20, **access**() ignored the effect of the **MS_NOEXEC** flag if it was used to **mount**(2) the underlying file system. Since kernel 2.6.20, **access**() honors this flag.

SEE ALSO

chmod(2), **chown**(2), **faccessat**(2), **open**(2), **setgid**(2), **setuid**(2), **stat**(2), **euidaccess**(3), **credentials**(7), **path_resolution**(7)

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.