access - Man Page

determine accessibility of a file descriptor

Пролог

Эта страница руководства является частью Руководства программиста POSIX. Реализация этого интерфейса в Linux может отличаться (обратитесь к соответствующей странице руководства Linux для получения подробной информации о поведении Linux), или интерфейс может быть не реализован в Linux.

Краткое описание

```
#include <unistd.h>

int access(const char *path, int amode);

#include <fcntl.h>

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

Описание

Функция *access*() проверяет файл с именем пути, на который указывает аргумент *path*, на доступность в соответствии с битовым шаблоном, содержащимся в *amode*. Проверки доступности (включая разрешения каталога, проверяемые при разрешении пути) должны выполняться с использованием реального идентификатора пользователя вместо эффективного идентификатора пользователя и реального идентификатора группы вместо эффективного идентификатора группы.

Значение *amode* является либо побитовым включением, либо проверяемыми разрешениями доступа (R_OK, W_OK, X_OK), либо тестом существования (F_OK).

Если какие-либо разрешения доступа проверены, каждое должно быть проверено индивидуально, как описано в Базовом томе определений POSIX.1-2017, *Раздел 4.5*, Права *доступа к файлам*, за исключением случаев, когда это описание относится к разрешению на выполнение для процесса с соответствующими привилегиями, реализация может указывать на успех для X_OK, даже если разрешение на выполнениене предоставляется ни одному пользователю.

Функция *faccessat*() при вызове с нулевым значением флага должна быть эквивалентна функции *access*(), за исключением случая, когда *path* указывает относительный путь. В этом случае файл, доступность которого должна быть определена, должен располагаться относительно каталога, связанного с файловым дескриптором *fd* вместо текущего рабочего каталога. Если режим доступа описания открытого файла, связанного с файловым

access - Man Page

основе файлового дескриптора. Если режим доступа 0_SEARCH, функция не должна выполнять проверку.

Если *faccessat*() передается специальное значение AT_FDCWD в параметре *fd* , должен использоваться текущий рабочий каталог, а если флаг равен нулю, поведение должно быть идентичным вызову *access*() .

Значения для *флага* строятся побитовым включением ИЛИ флагами из следующего списка, определенного в *<fcntl.h>*:

AT_EACCESS

Проверки доступности (включая разрешения каталога, проверяемые при разрешении пути) должны выполняться с использованием действительного идентификатора пользователя и идентификатора группы вместо реального идентификатора пользователя и идентификатора группы, как требуется в вызове *access*() .

Возвращаемое значение

После успешного завершения эти функции должны возвращать 0. В противном случае эти функции должны возвращать -1 и устанавливать *errno* для указания ошибки.

Ошибки

Эти функции не будут работать, если:

EACCES

Биты разрешения файлового режима не разрешают запрошенный доступ, или разрешение на поиск отказано в компоненте префикса пути.

ELOOP

Цикл существует в символических ссылках, встречающихся при разрешении аргумента *path* .

ENAMETOOLONG

Длина компонента пути больше {NAME_MAX} .

ENOENT

Компонент *path* не называет существующий файл или *путь* является пустой строкой.

ENOTDIR

Компонент префикса path называет существующий файл, который не является ни каталогом, ни символической ссылкой на каталог, или аргумент *path* содержит по крайней мере один символ, не являющийся <slash>, и заканчивается одним или несколькими конечными символами <slash>, а последний компонент pathname называет существующий файл, который не

access - Man Page

EROFS

доступ на запись запрашивается для файла в файловой системе, доступной только для чтения.

Функция *faccessat*() завершится ошибкой, если:

EACCES

Режим доступа описания открытого файла, связанного с *fd*, не является O_SEARCH, а разрешения каталога, лежащего *в основе fd*, не разрешают поиск в каталоге.

EBADF

Аргумент *path* не указывает абсолютный путь, а аргумент *fd* не является ни AT_FDCWD, ни допустимым файловым дескриптором, открытым для чтения или поиска.

ENOTDIR

Аргумент *path* не является абсолютным путем, а *fd* является файловым дескриптором, связанным с файлом, не относящимся к каталогу.

Эти функции могут выйти из строя, если:

EINVAL

Значение аргумента *amode* недопустимо.

ELOOP

Во время разрешения аргумента path было обнаружено более {SYMLOOP_MAX} символических ссылок.

ENAMETOOLONG

Длина пути превышает {PATH_MAX}, или разрешение пути символической ссылки дало промежуточный результат с длиной, превышающей {PATH_MAX} .

ETXTBSY

Доступ на запись запрашивается для выполняемого файла чистой процедуры (общий текст).

Функция *faccessat*() может выйти из строя, если:

EINVAL

Значение аргумента *flag* недопустимо.

Следующие разделы являются информативными.

Примеры

access - Man Page

В следующем примере проверяется, существует ли файл с именем **myfile** в каталоге /tmp.

```
#include <unistd.h>
...
результат int;
const char *pathname = "/tmp/myfile";

результат = access (путь, F_OK);
```

Использование приложений

Использование этих функций не рекомендуется, поскольку к моменту обработки возвращаемой информации она устарела. (То есть действие на информацию всегда приводит к состоянию гонки от времени проверки до времени использования.) Вместо этого приложение должно попытаться выполнить само действие и обработать **ошибку [EACCES]**, которая возникает, если файл недоступен (с предварительным изменением эффективных идентификаторов пользователей и групп и, возможно, с последующим изменением, в случае, когда *access*() или *faccessat*() без AT_EACCES были бы использованы.)

Исторически сложилось так, *что access*() использовался в корневых программах set-user-ID для проверки того, имеет ли пользователь, запускающий программу, доступ к файлу. Это опиралось на привилегии “суперпользователя”, которые были предоставлены на основе эффективного идентификатора пользователя, равного нулю, так что когда *access*() использовал реальный идентификатор пользователя для проверки доступности, эти привилегии не учитывались. В более новых системах, где могут быть назначены привилегии, которые не связаны с идентификаторами пользователей или групп, если программа с такими привилегиями вызывает *access*(), изменение идентификаторов не влияет на привилегии, и поэтому они учитываются при проверке доступности. Таким образом, *access*() (и *faccessat*() с нулевым флагом) не могут использоваться для этой исторической цели в таких программах. Аналогично, если система предоставляет какие-либо дополнительные или альтернативные механизмы управления доступом к файлам, которые не основаны на идентификаторе пользователя, они все равно будут приняты во внимание.

Если используется относительный путь, не учитывается, доступен ли текущий каталог (или каталог, связанный с файловым дескриптором *fd*) через любой абсолютный путь. *Следовательно, приложения, использующие access*() или *faccessat*() без AT_EACCES, могут действовать так, как если бы файл был доступен пользователю с реальным идентификатором пользователя и идентификатором группы процесса, когда такой пользователь на практике не сможет получить доступ к файлу, поскольку доступ будет запрещен в какой-то момент вышетекущий каталог (или каталог, связанный с файловым дескриптором *fd*) в файловой иерархии.

access - Man Page

указывающее, что каталог доступен для записи, может вводить в заблуждение, поскольку некоторые операции с файлами в каталоге не будут разрешены в зависимости от владельца этих файлов (см.Базовый том определений POSIX.1-2017, *Раздел 4.3*, *Защита каталогов*).

Допустимыми могут быть дополнительные значения amode, отличные от набора, определенного в описании; например, если система имеет расширенные элементы управления доступом.

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

Обоснование

В ранних предложениях некоторые недостатки в функции *access*() привели к созданию функции *eaccess*(), потому что:

1. Исторические реализации *access*() не проверяют доступ к файлам правильно, когда реальный идентификатор пользователя процесса является суперпользователем. В частности, они всегда возвращают ноль при тестировании разрешений на выполнение независимо от того, является ли файл исполняемым.
2. Суперпользователь имеет полный доступ ко всем файлам в системе. Как следствие, программы, запущенные суперпользователем и переключенные на эффективный идентификатор пользователя с меньшими привилегиями, не могут использовать *access*() для проверки своих прав доступа к файлам.

Однако историческая модель *eaccess*() не решает проблему (1), поэтому этот том POSIX.1-2017 теперь позволяет *access*() вести себя желаемым образом, поскольку несколько реализаций исправили проблему. Также утверждалось, что проблему (2) легче решить, используя *open*() , *chdir*() или одну из функций *exec* в зависимости от обстоятельств и реагируя на ошибку, а не создавая новую функцию, которая не была бы такой надежной. Поэтому *функция eaccess*() не включена в данный том POSIX.1-2017.

Предложение, касающееся соответствующих привилегий и битов разрешения на выполнение, отражает две возможности, реализованные историческими реализациями при проверке доступа суперпользователя к X_OK.

Новым реализациям не рекомендуется возвращать X_OK, если не установлен хотя бы один бит разрешения на выполнение.

Цель функции *faccessat*() - включить проверку доступности файлов в каталогах, отличных от текущего рабочего каталога, без воздействия условий гонки. Любая часть пути к файлу может быть изменена параллельно вызову *access*(), что приведет к неопределенному поведению. Открыв файловый дескриптор для целевого каталога и используя функцию *faccessat*(), можно гарантировать, что файл, проверенный на доступность, находится относительно нужного каталога.

access - Man Page

Эти функции могут быть формально устаревшими (например, путем затенения их OB) в будущей версии этого стандарта.

См. Также

chmod(), fstatat()

Базовый том определений POSIX.1-2017, *Раздел 4.5*, *Права доступа к файлам*, <fcntl.h>, <unistd.h>

Авторские права

Ссылка на

chmod (3p), ex (1p), fstatat (3p), unistd.h (0p).

2017 IEEE/The Open Group POSIX Programmer's Manual

access - Man Page