**O\_TMPFILE** (since Linux 3.11)

Create an unnamed temporary regular file. The *pathname* argument specifies a directory; an unnamed inode will be created in that directory's filesystem. Anything written to the resulting file will be lost when the last file descriptor is closed, unless the file is given a name.

**Создать неназванный временный обычный файл. Аргумент pathname указывает каталог; безымянный индекс будет создан в файловой системе этого каталога. Все, что записано в результирующий файл, будет потеряно при закрытии последнего дескриптора файла, если файлу не присвоено имя.**

**O\_TMPFILE** must be specified with one of **O\_RDWR** or **O\_WRONLY**

and, optionally, **O\_EXCL**. If **O\_EXCL** is not specified, then

[linkat(2)](http://man7.org/linux/man-pages/man2/linkat.2.html) can be used to link the temporary file into the

filesystem, making it permanent, using code like the follow‐

ing:

               O\_TMPFILE должен быть указан с одним из O\_RDWR или O\_WRONLY и, необязательно, O\_EXCL. Если O\_EXCL не указан, то linkat (2) может использоваться для связывания временного файла с файловой системой, делая его постоянным, используя код, подобный следующему:

char path[PATH\_MAX];

fd = open("/path/to/dir", O\_TMPFILE | O\_RDWR, S\_IRUSR | S\_IWUSR);

/\* File I/O on 'fd'... \*/

linkat(fd, NULL, AT\_FDCWD, "/path/for/file", AT\_EMPTY\_PATH);

/\* If the caller doesn't have the CAP\_DAC\_READ\_SEARCH

capability (needed to use AT\_EMPTY\_PATH with linkat(2)),

and there is a proc(5) filesystem mounted, then the

linkat(2) call above can be replaced with:

snprintf(path, PATH\_MAX, "/proc/self/fd/%d", fd);

linkat(AT\_FDCWD, path, AT\_FDCWD, "/path/for/file",

AT\_SYMLINK\_FOLLOW);

\*/

In this case, the **open**() *mode* argument determines the file permission mode, as with **O\_CREAT**.

Specifying **O\_EXCL** in conjunction with **O\_TMPFILE** prevents a

temporary file from being linked into the filesystem in the

above manner. (Note that the meaning of **O\_EXCL** in this case

is different from the meaning of **O\_EXCL** otherwise.)

There are two main use cases for **O\_TMPFILE**:

\* Improved [tmpfile(3)](http://man7.org/linux/man-pages/man3/tmpfile.3.html) functionality: race-free creation of

temporary files that (1) are automatically deleted when

closed; (2) can never be reached via any pathname; (3) are

not subject to symlink attacks; and (4) do not require the

caller to devise unique names.

В этом случае аргумент режима open () определяет режим доступа к файлу, как и в случае O\_CREAT.

  Указание O\_EXCL в сочетании с O\_TMPFILE предотвращает привязку временного файла к файловой системе вышеуказанным способом. (Обратите внимание, что значение O\_EXCL в этом случае отличается от значения O\_EXCL в противном случае.)

   Существует два основных варианта использования O\_TMPFILE:

               \* Улучшена функциональность tmpfile (3): создание временных файлов без скачек, которые (1) автоматически удаляются при закрытии; (2) никогда не может быть достигнут через любое имя пути; (3) не подвергаются атакам по символическим ссылкам; и (4) не требуют, чтобы вызывающий абонент придумывал уникальные имена.

\* Creating a file that is initially invisible, which is then

populated with data and adjusted to have appropriate

filesystem attributes ([fchown(2)](http://man7.org/linux/man-pages/man2/fchown.2.html), [fchmod(2)](http://man7.org/linux/man-pages/man2/fchmod.2.html), [fsetxattr(2)](http://man7.org/linux/man-pages/man2/fsetxattr.2.html),

etc.) before being atomically linked into the filesystem

in a fully formed state (using [linkat(2)](http://man7.org/linux/man-pages/man2/linkat.2.html) as described

above).

**O\_TMPFILE** requires support by the underlying filesystem; only

a subset of Linux filesystems provide that support. In the

initial implementation, support was provided in the ext2,

ext3, ext4, UDF, Minix, and shmem filesystems. Support for

other filesystems has subsequently been added as follows: XFS

(Linux 3.15); Btrfs (Linux 3.16); F2FS (Linux 3.16); and ubifs

(Linux 4.9)

O\_TMPFILE требует поддержки базовой файловой системой; только подмножество файловых систем Linux обеспечивают такую поддержку. В первоначальной реализации поддержка предоставлялась в файловых системах ext2, ext3, ext4, UDF, Minix и shmem. Впоследствии была добавлена поддержка других файловых систем: XFS (Linux 3.15); Btrfs (Linux 3.16); F2FS (Linux 3.16); и ubifs (Linux 4.9)

**O\_PATH** (since Linux 2.6.39)

Obtain a file descriptor that can be used for two purposes: to indicate a location in the filesystem tree and to perform operations that act purely at the file descriptor level. The file itself is not opened, and other file operations (e.g., [read(2)](http://man7.org/linux/man-pages/man2/read.2.html), [write(2)](http://man7.org/linux/man-pages/man2/write.2.html), [fchmod(2)](http://man7.org/linux/man-pages/man2/fchmod.2.html), [fchown(2)](http://man7.org/linux/man-pages/man2/fchown.2.html), [fgetxattr(2)](http://man7.org/linux/man-pages/man2/fgetxattr.2.html), [ioctl(2)](http://man7.org/linux/man-pages/man2/ioctl.2.html), [mmap(2)](http://man7.org/linux/man-pages/man2/mmap.2.html)) fail with the error **EBADF**.

The following operations *can* be performed on the resulting file descriptor:

Получите файловый дескриптор, который можно использовать для двух целей: указать местоположение в дереве файловой системы и выполнить операции, которые действуют исключительно на уровне файлового дескриптора. Сам файл не открывается, и другие файловые операции (например, read (2), write (2), fchmod (2), fchown (2), fgetxattr (2), ioctl (2), mmap (2)) завершаются неудачно с ошибкой EBADF.

  Следующие операции могут быть выполнены с результирующим дескриптором файла:

\* [close(2)](http://man7.org/linux/man-pages/man2/close.2.html).

\* [fchdir(2)](http://man7.org/linux/man-pages/man2/fchdir.2.html), if the file descriptor refers to a directory

(since Linux 3.5).

\* [fstat(2)](http://man7.org/linux/man-pages/man2/fstat.2.html) (since Linux 3.6).

\* [fstatfs(2)](http://man7.org/linux/man-pages/man2/fstatfs.2.html) (since Linux 3.12).

\* Duplicating the file descriptor ([dup(2)](http://man7.org/linux/man-pages/man2/dup.2.html), [fcntl(2)](http://man7.org/linux/man-pages/man2/fcntl.2.html) **F\_DUPFD**,

etc.).

\* Getting and setting file descriptor flags ([fcntl(2)](http://man7.org/linux/man-pages/man2/fcntl.2.html) **F\_GETFD**

and **F\_SETFD**).

\* Retrieving open file status flags using the [fcntl(2)](http://man7.org/linux/man-pages/man2/fcntl.2.html)

**F\_GETFL** operation: the returned flags will include the bit

**O\_PATH**.

\* Passing the file descriptor as the *dirfd* argument of

**openat**() and the other "\*at()" system calls. This includes

[linkat(2)](http://man7.org/linux/man-pages/man2/linkat.2.html) with **AT\_EMPTY\_PATH** (or via procfs using

**AT\_SYMLINK\_FOLLOW**) even if the file is not a directory.

\* Passing the file descriptor to another process via a UNIX

domain socket (see **SCM\_RIGHTS** in [unix(7)](http://man7.org/linux/man-pages/man7/unix.7.html)).

When **O\_PATH** is specified in *flags*, flag bits other than **O\_CLOEXEC**, **O\_DIRECTORY**, and **O\_NOFOLLOW** are ignored.

Когда O\_PATH указан в флагах, биты флага, отличные от O\_CLOEXEC, O\_DIRECTORY и O\_NOFOLLOW, игнорируются.

Opening a file or directory with the **O\_PATH** flag requires no permissions on the object itself (but does require execute permission on the directories in the path prefix). Depending on the subsequent operation, a check for suitable file permissions may be performed (e.g., [fchdir(2)](http://man7.org/linux/man-pages/man2/fchdir.2.html) requires execute permission on the directory referred to by its file descriptor argument). By contrast, obtaining a reference to a filesystem object by opening it with the **O\_RDONLY** flag requires that the caller have read permission on the object, even when the subsequent operation (e.g., [fchdir(2)](http://man7.org/linux/man-pages/man2/fchdir.2.html), [fstat(2)](http://man7.org/linux/man-pages/man2/fstat.2.html)) does not require read permission on the object.

Открытие файла или каталога с флагом O\_PATH не требует никаких полномочий для самого объекта (но требует разрешения на выполнение для каталогов в префиксе пути). В зависимости от последующей операции может быть выполнена проверка подходящих прав доступа к файлу (например, fchdir (2) требует разрешения на выполнение для каталога, на который ссылается аргумент дескриптора файла). Напротив, получение ссылки на объект файловой системы путем открытия его с флагом O\_RDONLY требует, чтобы вызывающая сторона имела разрешение на чтение объекта, даже когда последующая операция (например, fchdir (2), fstat (2)) не требует чтения разрешение на объект.

If *pathname* is a symbolic link and the **O\_NOFOLLOW** flag is also specified, then the call returns a file descriptor referring to the symbolic link. This file descriptor can be used as the *dirfd* argument in calls to [fchownat(2)](http://man7.org/linux/man-pages/man2/fchownat.2.html), [fstatat(2)](http://man7.org/linux/man-pages/man2/fstatat.2.html), [linkat(2)](http://man7.org/linux/man-pages/man2/linkat.2.html), and [readlinkat(2)](http://man7.org/linux/man-pages/man2/readlinkat.2.html) with an empty pathname to have the calls operate on the symbolic link.

Если pathname является символической ссылкой и также указан флаг O\_NOFOLLOW, то вызов возвращает дескриптор файла, ссылающийся на символическую ссылку. Этот файловый дескриптор может использоваться в качестве аргумента dirfd в вызовах fchownat (2), fstatat (2), linkat (2) и readlinkat (2) с пустым путем, чтобы вызовы работали с символической ссылкой.

If *pathname* refers to an automount point that has not yet been

triggered, so no other filesystem is mounted on it, then the

call returns a file descriptor referring to the automount

directory without triggering a mount. [fstatfs(2)](http://man7.org/linux/man-pages/man2/fstatfs.2.html) can then be

used to determine if it is, in fact, an untriggered automount

point (**.f\_type == AUTOFS\_SUPER\_MAGIC**).

Если pathname ссылается на точку автомонтирования, которая еще не была запущена, и никакая другая файловая система не смонтирована на нем, то вызов возвращает файловый дескриптор, ссылающийся на каталог автомонтирования, без запуска монтирования. Затем можно использовать fstatfs (2), чтобы определить, является ли она фактически неуправляемой точкой автоматического монтирования (.f\_type == AUTOFS\_SUPER\_MAGIC).

One use of **O\_PATH** for regular files is to provide the

equivalent of POSIX.1's **O\_EXEC** functionality. This permits us

to open a file for which we have execute permission but not

read permission, and then execute that file, with steps

something like the following:

char buf[PATH\_MAX];

fd = open("some\_prog", O\_PATH);

snprintf(buf, PATH\_MAX, "/proc/self/fd/%d", fd);

execl(buf, "some\_prog", (char \*) NULL);

An **O\_PATH** file descriptor can also be passed as the argument

of [fexecve(3)](http://man7.org/linux/man-pages/man3/fexecve.3.html).

static int [**do\_tmpfile**](https://elixir.bootlin.com/linux/latest/ident/do_tmpfile)(struct [**nameidata**](https://elixir.bootlin.com/linux/latest/ident/nameidata) \*nd, unsigned flags,

const struct [**open\_flags**](https://elixir.bootlin.com/linux/latest/ident/open_flags) \*[**op**](https://elixir.bootlin.com/linux/latest/ident/op),

struct [**file**](https://elixir.bootlin.com/linux/latest/ident/file) \*[**file**](https://elixir.bootlin.com/linux/latest/ident/file))

{

struct [**dentry**](https://elixir.bootlin.com/linux/latest/ident/dentry) \*[**child**](https://elixir.bootlin.com/linux/latest/ident/child);

struct [**path**](https://elixir.bootlin.com/linux/latest/ident/path) [**path**](https://elixir.bootlin.com/linux/latest/ident/path);

int error = [**path\_lookupat**](https://elixir.bootlin.com/linux/latest/ident/path_lookupat)(nd, flags | [**LOOKUP\_DIRECTORY**](https://elixir.bootlin.com/linux/latest/ident/LOOKUP_DIRECTORY), &[**path**](https://elixir.bootlin.com/linux/latest/ident/path));

if ([**unlikely**](https://elixir.bootlin.com/linux/latest/ident/unlikely)(error))

[**return**](https://elixir.bootlin.com/linux/latest/ident/return) error;

error = [**mnt\_want\_write**](https://elixir.bootlin.com/linux/latest/ident/mnt_want_write)(**[path](https://elixir.bootlin.com/linux/latest/ident/path)**.[**mnt**](https://elixir.bootlin.com/linux/latest/ident/mnt));

if ([**unlikely**](https://elixir.bootlin.com/linux/latest/ident/unlikely)(error))

goto out;

[**child**](https://elixir.bootlin.com/linux/latest/ident/child) = [**vfs\_tmpfile**](https://elixir.bootlin.com/linux/latest/ident/vfs_tmpfile)(**[path](https://elixir.bootlin.com/linux/latest/ident/path)**.[**dentry**](https://elixir.bootlin.com/linux/latest/ident/dentry), [**op**](https://elixir.bootlin.com/linux/latest/ident/op)->mode, [**op**](https://elixir.bootlin.com/linux/latest/ident/op)->**[open\_flag](https://elixir.bootlin.com/linux/latest/ident/open_flag)**);

error = [**PTR\_ERR**](https://elixir.bootlin.com/linux/latest/ident/PTR_ERR)([**child**](https://elixir.bootlin.com/linux/latest/ident/child));

if ([**IS\_ERR**](https://elixir.bootlin.com/linux/latest/ident/IS_ERR)([**child**](https://elixir.bootlin.com/linux/latest/ident/child)))

goto [**out2**](https://elixir.bootlin.com/linux/latest/ident/out2);

[**dput**](https://elixir.bootlin.com/linux/latest/ident/dput)(**[path](https://elixir.bootlin.com/linux/latest/ident/path)**.[**dentry**](https://elixir.bootlin.com/linux/latest/ident/dentry));

[**path**](https://elixir.bootlin.com/linux/latest/ident/path).[**dentry**](https://elixir.bootlin.com/linux/latest/ident/dentry) = [**child**](https://elixir.bootlin.com/linux/latest/ident/child);

[**audit\_inode**](https://elixir.bootlin.com/linux/latest/ident/audit_inode)(nd->name, [**child**](https://elixir.bootlin.com/linux/latest/ident/child), 0);

*/\* Don't check for other permissions, the inode was just created \*/*

error = [**may\_open**](https://elixir.bootlin.com/linux/latest/ident/may_open)(&[**path**](https://elixir.bootlin.com/linux/latest/ident/path), 0, [**op**](https://elixir.bootlin.com/linux/latest/ident/op)->**[open\_flag](https://elixir.bootlin.com/linux/latest/ident/open_flag)**);

if (error)

goto [**out2**](https://elixir.bootlin.com/linux/latest/ident/out2);

[**file**](https://elixir.bootlin.com/linux/latest/ident/file)->f\_path.[**mnt**](https://elixir.bootlin.com/linux/latest/ident/mnt) = [**path**](https://elixir.bootlin.com/linux/latest/ident/path).[**mnt**](https://elixir.bootlin.com/linux/latest/ident/mnt);

error = [**finish\_open**](https://elixir.bootlin.com/linux/latest/ident/finish_open)([**file**](https://elixir.bootlin.com/linux/latest/ident/file), [**child**](https://elixir.bootlin.com/linux/latest/ident/child), NULL);

[**out2**](https://elixir.bootlin.com/linux/latest/ident/out2):

[**mnt\_drop\_write**](https://elixir.bootlin.com/linux/latest/ident/mnt_drop_write)(**[path](https://elixir.bootlin.com/linux/latest/ident/path)**.[**mnt**](https://elixir.bootlin.com/linux/latest/ident/mnt));

out:

[**path\_put**](https://elixir.bootlin.com/linux/latest/ident/path_put)(&[**path**](https://elixir.bootlin.com/linux/latest/ident/path));

[**return**](https://elixir.bootlin.com/linux/latest/ident/return) error;

}

static int [**do\_o\_path**](https://elixir.bootlin.com/linux/latest/ident/do_o_path)(struct [**nameidata**](https://elixir.bootlin.com/linux/latest/ident/nameidata) \*nd, unsigned flags, struct [**file**](https://elixir.bootlin.com/linux/latest/ident/file) \*[**file**](https://elixir.bootlin.com/linux/latest/ident/file))

{

struct [**path**](https://elixir.bootlin.com/linux/latest/ident/path) [**path**](https://elixir.bootlin.com/linux/latest/ident/path);

int error = [**path\_lookupat**](https://elixir.bootlin.com/linux/latest/ident/path_lookupat)(nd, flags, &[**path**](https://elixir.bootlin.com/linux/latest/ident/path));

if (!error) {

[**audit\_inode**](https://elixir.bootlin.com/linux/latest/ident/audit_inode)(nd->name, [**path**](https://elixir.bootlin.com/linux/latest/ident/path).[**dentry**](https://elixir.bootlin.com/linux/latest/ident/dentry), 0);

error = [**vfs\_open**](https://elixir.bootlin.com/linux/latest/ident/vfs_open)(&[**path**](https://elixir.bootlin.com/linux/latest/ident/path), [**file**](https://elixir.bootlin.com/linux/latest/ident/file));

[**path\_put**](https://elixir.bootlin.com/linux/latest/ident/path_put)(&[**path**](https://elixir.bootlin.com/linux/latest/ident/path));

}

[**return**](https://elixir.bootlin.com/linux/latest/ident/return) error;

}