

# Основы построения файловых систем



### Как уместить много разделов и ФС в одном дереве каталогов: точки монтирования

Все ФС, которые надо сделать видимыми пользовательским приложениям, «подсоединяются» к уже существующим каталогам в ФС, видной пользователю.

Точка монтирования с точки зрения ядра ОС – это отметка на каталоге «начиная отсюда, поиск имени делается от корня такой-то ФС».

```
$ ls -lh ~/testing/mount/  
total 0
```

```
$ mount -t ext4 ~/testing/fs-images/rpmbuild ~/testing/mount/
```

```
$ ls -lh ~/testing/mount/  
total 8.0K  
drwxrwxr-x 1 1002 1002 4.0K Sep 25 16:59 pstorage-fes  
drwxr-xr-x 1 1002 1002  83 Sep  6 21:11 rpmbuild
```

### Как уместить много разделов и ФС в одном дереве каталогов: точки монтирования

Все ФС, которые надо сделать видимыми пользовательским приложениям, «подсоединяются» к уже существующим каталогам в ФС, видной пользователю.

Точка монтирования с точки зрения ядра ОС – это отметка на каталоге «начиная отсюда, поиск имени делается от корня такой-то ФС».

```
$ ls -lh ~/testing/mount/  
total 0
```

```
$ mount -t ext4 ~/testing/fs-images/rpmbuild ~/testing/mount/
```

```
$ ls -lh ~/testing/mount/  
total 8.0K  
drwxrwxr-x 1 1002 1002 4.0K Sep 25 16:59 pstorage-fes  
drwxr-xr-x 1 1002 1002  83 Sep  6 21:11 rpmbuild
```

Посмотреть список точек монтирования можно так:

- \$ cat /proc/self/mounts

Монтировать ФС можно по требованию: <https://linux.die.net/man/5/auto.master>

## Ещё пример того, что объект ФС и его имя разделены

С помощью `link()` и `unlink()` можно создавать файлы, у которых есть несколько имён, или нет имён вообще.

Рабочий каталог тоже не привязан к пути:

```
artem@dev:~/testing/students$ pwd  
/home/artem/testing/students
```

```
artem@dev:~/testing/students$ ls -lh .  
total 40K  
-rw-r--r-- 1 artem artem 234 Sep 28 11:48 example  
-rw-r--r-- 1 artem artem 11K Sep 27 21:49 proc  
-rw-r--r-- 1 artem artem 1.1K Sep 27 21:49 proc.c  
-rw-r--r-- 1 artem artem 13K Sep 28 20:14 ps  
-rw-r--r-- 1 artem artem 1.6K Sep 28 20:13 ps.c
```

```
artem@dev:~/testing/students$ sshfs -o nonempty aanisimov@vzbuild ~/testings/students/
```

```
artem@dev:~/testing/students$ ls -lh .
```

???

```
artem@dev:~/testing/students$ ls -lh ~/testing/students/
```

???

### Ещё пример того, что объект ФС и его имя разделены

С помощью `link()` и `unlink()` можно создавать файлы, у которых есть несколько имён, или нет имён вообще.

Рабочий каталог тоже не привязан к пути:

```
artem@dev:~/testing/students$ pwd
/home/artem/testing/students
```

```
artem@dev:~/testing/students$ ls -lh .
total 40K
-rw-r--r-- 1 artem artem 234 Sep 28 11:48 example
-rw-r--r-- 1 artem artem 11K Sep 27 21:49 proc
-rw-r--r-- 1 artem artem 1.1K Sep 27 21:49 proc.c
-rw-r--r-- 1 artem artem 13K Sep 28 20:14 ps
-rw-r--r-- 1 artem artem 1.6K Sep 28 20:13 ps.c
```

```
artem@dev:~/testing/students$ sshfs -o nonempty aanisimov@vzbuild ~/testings/students/
```

```
artem@dev:~/testing/students$ ls -lh .
total 40K
-rw-r--r-- 1 artem artem 234 Sep 28 11:48 example
-rw-r--r-- 1 artem artem 11K Sep 27 21:49 proc
-rw-r--r-- 1 artem artem 1.1K Sep 27 21:49 proc.c
-rw-r--r-- 1 artem artem 13K Sep 28 20:14 ps
-rw-r--r-- 1 artem artem 1.6K Sep 28 20:13 ps.c
```

```
artem@dev:~/testing/students$ ls -lh ~/testing/students/
total 8.0K
drwxrwxr-x 1 1002 1002 4.0K Sep 25 16:59 pstorage-fes
drwxr-xr-x 1 1002 1002 83 Sep 6 21:11 rpmbuild
```

## Виртуальные FS в Linux

С точки зрения пользователя, ФС – это не набор данных на диске, представленный в виде дерева каталогов и файлов, а объект, который позволяет

- Найти файл или каталог по имени,
- Перечислить содержимое каталога,
- Читать и переписывать содержимое файла.

## Виртуальные FS в Linux: procfs

В Linux есть файловая система, каталоги в корне которой соответствуют исполняющимся процессам, а файлы внутри каждого каталога описывают состояние процесса.

```
artem@dev:~$ ls -lh /proc/self/
total 0

-r--r--r-- 1 artem artem 0 0ct  2 10:08 cmdline
lrwxrwxrwx 1 artem artem 0 0ct  2 10:08 cwd -> /home/artem
lrwxrwxrwx 1 artem artem 0 0ct  2 10:08 exe -> /bin/ls
dr-x----- 2 artem artem 0 0ct  2 10:08 fd
-r--r--r-- 1 artem artem 0 0ct  2 10:08 maps
-r--r--r-- 1 artem artem 0 0ct  2 10:08 stat
.....
```

Домашнее задание:

- `man 5 proc`,
- что находится в файле `/proc/PID/auxv` и как выглядит стек процесса сразу после `execve()`?
- напишите программу, которая спрячет первый аргумент командной строки из `/proc/PID/cmdline` (подсказка: `man prctl + операция PR_SET_MM`),
- напишите аналоги
  - `ps`
  - `lsuf`

## Виртуальные FS в Linux: FUSE

FUSE (Filesystem in USer space) – это способ создания драйверов файловых систем, которые исполняются как пользовательские процессы, а не часть ядра.

Драйверы FUSE работают как сервера, которые обслуживают один pipe. Они читают оттуда команды вроде “найти элемент каталога по имени”, “открыть/закрыть файл”, “прочитать/записать данные в файл”. Клиентом такого сервера является ядро ОС.

Пример: sshfs.

Преимущества FUSE:

- Позволяет легко экспериментировать,
- Реализации ФС могут иметь сложные зависимости, которые были бы нежелательны в ядре,
- Может использоваться непривилегированными пользователями.

Недостатки FUSE:

- Низкая производительность.

Домашнее задание:

- прочтите документацию по FUSE high level API,
- напишите драйвер для FUSE, который предоставляет ФС, состоящую из одного файла “hello”, из которого можно прочесть строку “hello, world!”; в каталоге, куда монтируется эта ФС, должны работать команды `ls` и `cat hello`.



## POSIX filesystem API

UNIX – многопользовательская ОС, поэтому требуется разделение доступа к файлам.

## POSIX filesystem API

UNIX – многопользовательская ОС, поэтому требуется разделение доступа к файлам.

Модель безопасности:

- имеется множество пользователей и групп, в которых пользователи состоят,
- каждый файл принадлежит одному пользователю и одной группе,
- файл указывает, какой доступ разрешён пользователю-владельцу, группе-владельцу и всем остальным.

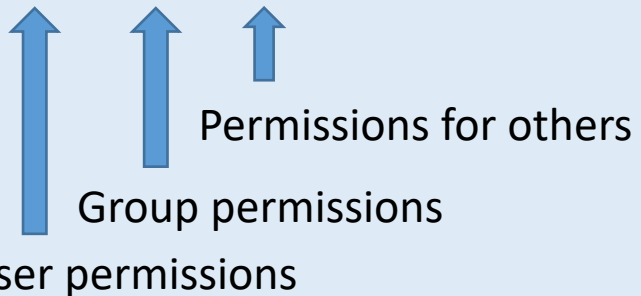
# POSIX filesystem API

UNIX – многопользовательская ОС, поэтому требуется разделение доступа к файлам.

Модель безопасности:

- имеется множество пользователей и групп, в которых пользователи состоят,
- каждый файл принадлежит одному пользователю и одной группе,
- файл указывает, какой доступ разрешён пользователю-владельцу, группе-владельцу и всем остальным.

```
$ ls -lh /usr/bin
...
-rwx  r-x  --x 1 root  root    31K Feb 22  2016 afm2pl
-rwx  r-x  --x 1 root  root    38K Feb 22  2016 afm2tfm
-rwx  r-x  --x 1 root  root   485K Feb 22  2016 aleph
```



The diagram illustrates the permission fields in the `ls` output. Three blue arrows point upwards from labels at the bottom to the first three fields of the last line (`aleph`):

- A tall arrow from **User permissions** points to the `rwx` field.
- A medium arrow from **Group permissions** points to the `r-x` field.
- A short arrow from **Permissions for others** points to the `--x` field.

# POSIX filesystem API

Имеются «права доступа», которые меняют то, как запускаются программы:

```
$ ls -lh /usr/bin
...
-rwsr-xr-x 1 root root 134K Jan 6 2016 sudo
-rwxrwxrwx 1 root root 4 Jan 6 2016 sudoedit -> sudo
-rwxr-xr-x 1 root root 47K Jan 6 2016 sudoreplay
```

## POSIX filesystem API

Имеются «права доступа», которые меняют то, как запускаются программы:

```
$ ls -lh /usr/bin
...
-rwsr-xr-x 1 root root 134K Jan 6 2016 sudo
-rwxrwxrwx 1 root root 4 Jan 6 2016 sudoedit -> sudo
-rwxr-xr-x 1 root root 47K Jan 6 2016 sudoreplay
```

При запуске файла с установленным флагом set-uid (соотв., set-gid) он будет запущен от имени пользователя-владельца (соотв., группы-владельца).

## POSIX filesystem API

Права доступа к файлу и файловому дескриптору разделены:

```
int fd = open("/path/to/a/file", O_RDWR | O_CREAT, S_IRUSR);  
write(fd, buffer, size);  
close(fd);
```

## POSIX filesystem API

Права доступа к файлу и файловому дескриптору разделены:

```
int fd = open("/path/to/a/file", O_RDWR | O_CREAT, S_IRUSR);  
write(fd, buffer, size);  
close(fd);
```

Где применяется:

- Простая привилегированная программа проверяет права доступа и передаёт файловый дескриптор (сложной) непривилегированной программе.
- Помогает в реализации binfmt-обработчиков для файлов с правами доступа --x--x--x:  
<https://lwn.net/Articles/679310/>
- См. также seccomp и seccomp filters: [https://www.kernel.org/doc/Documentation/prctl/seccomp\\_filter.txt](https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt)

### ФС как разделяемый ресурс: конфликты пользователей

Когда несколько потоков работают с одним регионом памяти, они должны брать блокировку, чтобы гарантировать согласованность данных.

Что происходит, когда несколько процессов работают с одним каталогом?



### ФС как разделяемый ресурс: конфликты пользователей

Когда несколько потоков работают с одним регионом памяти, они должны брать блокировку, чтобы гарантировать согласованность данных.

Что происходит, когда несколько процессов работают с одним каталогом?

1. Создание временного файла:

```
int fd = open("/tmp/tmp.1b42ac00de", O_RDWR|O_CREAT, 0);  
unlink("/tmp/tmp.1b42ac00de");  
... use fd to keep temp data ...
```

### ФС как разделяемый ресурс: конфликты пользователей

Когда несколько потоков работают с одним регионом памяти, они должны брать блокировку, чтобы гарантировать согласованность данных.

Что происходит, когда несколько процессов работают с одним каталогом?

1. Создание временного файла:

```
int fd = open("/tmp/tmp.1b42ac00de", O_RDWR|O_CREAT, 0);
unlink("/tmp/tmp.1b42ac00de");
... use fd to keep temp data ...
```

2. Чтение символической ссылки:

```
struct stat st;
lstat("/path/to/symlink", &st);
char *buf = malloc(st.st_size + 1);
readlink("/path/to/symlink", buf, st.st_size);
buf[st.st_size] = '\0';
```

## ФС как разделяемый ресурс: конфликты пользователей

Когда несколько потоков работают с одним регионом памяти, они должны брать блокировку, чтобы гарантировать согласованность данных.

Что происходит, когда несколько процессов работают с одним каталогом?

1. Создание временного файла:

```
int fd = open("/tmp/tmp.1b42ac00de", O_RDWR|O_CREAT, 0);
unlink("/tmp/tmp.1b42ac00de");
... use fd to keep temp data ...
```

2. Чтение символической ссылки:

```
struct stat st;
lstat("/path/to/symlink", &st);
char *buf = malloc(st.st_size + 1);
readlink("/path/to/symlink", buf, st.st_size);
buf[st.st_size] = '\0';
```

3. Создание двух файлов в одном каталоге:

```
int fd0 = open("/dir/a", O_RDWR|O_CREAT, S_IRUSR);
int fd1 = open("/dir/b", O_RDWR|O_CREAT, S_IRUSR);
```

## ФС как разделяемый ресурс: конфликты пользователей

Когда несколько потоков работают с одним регионом памяти, они должны брать блокировку, чтобы гарантировать согласованность данных.

Что происходит, когда несколько процессов работают с одним каталогом?

1. Создание временного файла:

```
int fd = open("/tmp/tmp.1b42ac00de", O_RDWR|O_CREAT, 0);
unlink("/tmp/tmp.1b42ac00de");
... use fd to keep temp data ...
```

2. Чтение символической ссылки:

```
struct stat st;
lstat("/path/to/symlink", &st);
char *buf = malloc(st.st_size + 1);
readlink("/path/to/symlink", buf, st.st_size);
buf[st.st_size] = '\0';
```

3. Создание двух файлов в одном каталоге:

```
int fd0 = open("/dir/a", O_RDWR|O_CREAT, S_IRUSR);
int fd1 = open("/dir/b", O_RDWR|O_CREAT, S_IRUSR);
```

1. В течение некоторого времени сторонний процесс может успеть открыть файл /tmp/tmp.1b42ac00de и подсматривать в данные чужого процесса.

2. В промежуток между вызовами lstat() и readlink() сторонний процесс может поменять значение ссылки (TOCTTOU: time of check to time of use).

3. В промежуток времени между вызовами open() сторонний процесс может переименовать каталог /dir и создать новый с тем же именем.

### ФС как разделяемый ресурс: конфликты пользователей

Одно из решений: контейнеризация приложений.

- Если у процессов А и В у каждого "свой" каталог /tmp, то они не могут видеть временные файлы друг друга.

### ФС как разделяемый ресурс: конфликты пользователей

Одно из решений: контейнеризация приложений.

- Если у процессов А и В у каждого "свой" каталог /tmp, то они не могут видеть временные файлы друг друга.
- В Linux есть "**filesystem namespaces**" – набор точек монтирования, видимых процессам, которые исполняются внутри filesystem namespace.
- В двух разных fs namespaces поверх каталога /tmp можно смонтировать разные экземпляры tmpfs.

### ФС как разделяемый ресурс: конфликты пользователей

Одно из решений: контейнеризация приложений.

- Если у процессов А и В у каждого "свой" каталог /tmp, то они не могут видеть временные файлы друг друга.
- В Linux есть "**filesystem namespaces**" – набор точек монтирования, видимых процессам, которые исполняются внутри filesystem namespace.
- В двух разных fs namespaces поверх каталога /tmp можно смонтировать разные экземпляры tmpfs.

См. также

- User namespaces,
- Pid namespaces,
- Network namespaces.

## Filesystem namespaces

Одно из решений: контейнеризация приложений.

- Если у процессов А и В у каждого "свой" каталог /tmp, то они не могут видеть временные файлы друг друга.
- В Linux есть "**filesystem namespaces**" – набор точек монтирования, видимых процессам, которые исполняются внутри filesystem namespace.
- В двух разных fs namespaces поверх каталога /tmp можно смонтировать разные экземпляры tmpfs.

Как заполнять filesystem namespace?

- Можно начать с пустой ФС,
- Добавить в неё read-only образы каталогов с бинарниками вроде /usr, /lib,
- Добавить специфичные для контейнера образы /tmp, /proc и прочих,
- Добавить каталоги с данными приложения, которое исполняется в контейнере.



## Filesystem namespaces

Одно из решений: контейнеризация приложений.

- Если у процессов А и В у каждого "свой" каталог /tmp, то они не могут видеть временные файлы друг друга.
- В Linux есть "**filesystem namespaces**" – набор точек монтирования, видимых процессам, которые исполняются внутри filesystem namespace.
- В двух разных fs namespaces поверх каталога /tmp можно смонтировать разные экземпляры tmpfs.

Как заполнять filesystem namespace?

- Можно начать с пустой ФС,
- Добавить в неё read-only образы каталогов с бинарниками вроде /usr, /lib,      • bind mounts
- Добавить специфичные для контейнера образы /tmp, /proc и прочих,
- Добавить каталоги с данными приложения, которое исполняется в      • см. "Kubernetes (persistent) volumes"
- контейнере.

## Bind-mounts

В Linux есть расширение понятия точек монтирования: каталоги, начиная с которых, поиск имени делается не от корня заданной ФС, а от другого каталога.

```
artem@dev:~/testing/bind-mount$ ls -lh src/
```

```
total 0
```

```
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 0
```

```
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 1
```

```
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 2
```

```
artem@dev:~/testing/bind-mount$ ls -lh dst/
```

```
total 0
```

```
artem@dev:~/testing/bind-mount$ sudo mount --bind src/ dst/
```

```
artem@dev:~/testing/bind-mount$ ls -lh dst/
```

```
total 0
```

```
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 0
```

```
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 1
```

```
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 2
```

## Bind-mounts

В Linux есть расширение понятия точек монтирования: каталоги, начиная с которых, поиск имени делается не от корня заданной ФС, а от другого каталога.

```
artem@dev:~/testing/bind-mount$ ls -lh src/
total 0
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 0
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 1
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 2

artem@dev:~/testing/bind-mount$ ls -lh dst/
total 0

artem@dev:~/testing/bind-mount$ sudo mount --bind src/ dst/

artem@dev:~/testing/bind-mount$ ls -lh dst/
total 0
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 0
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 1
-rw-r--r-- 1 artem artem 0 Oct  2 00:29 2
```

Bind mounts привносят много нетривиальных деталей:

- bind-mount можно делать на файлы
- <http://lwn.net/Articles/689856/>