

# 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, видимой 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$ lh -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$ lh -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
```

## 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/>

## 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:

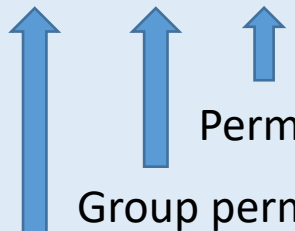
- имеется множество пользователей и групп, в которых пользователи состоят,
- каждый файл принадлежит одному пользователю и одной группе,
- файл указывает, какой доступ разрешён пользователю-владельцу, группе-владельцу и всем остальным.

# 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
```



Permissions for others

Group permissions

User permissions

## POSIX filesystem API

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

```
$ ls -lh /usr/bin
...
-rws  r-x  r-x 1 root  root  134K Jan  6  2016 sudo
-rwx  rwx  rwx 1 root  root    4 Jan  6  2016 sudoedit -> sudo
-rwx  r-x  r-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)

## Capabilities

В Linux была предпринята попытка сделать более гибкий контроль прав приложений, чем даёт SUID-root: process & file capabilities. Например:

```
# stat /usr/bin/ping
  File: '/usr/bin/ping'
  Size: 66176          Blocks: 136          IO Block: 4096   regular file
Device: 804h/2052d    Inode: 263776        Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (  0/   root)   Gid: (  0/   root)
```

```
# getcap /usr/bin/ping
/usr/bin/ping = cap_net_admin,cap_net_raw+p
```

## Capabilities

В Linux была предпринята попытка сделать более гибкий контроль прав приложений, чем даёт SUID-root: process & file capabilities. Например:

```
# stat /usr/bin/ping
  File: '/usr/bin/ping'
  Size: 66176          Blocks: 136          IO Block: 4096   regular file
Device: 804h/2052d    Inode: 263776       Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (  0/   root)   Gid: (  0/   root)
```

```
# getcap /usr/bin/ping
/usr/bin/ping = cap_net_admin,cap_net_raw+p
```

Попытка оказалась неудачной, так как слишком много прав спрятали за CAP\_NET\_ADMIN и CAP\_SYS\_ADMIN, который по факту оказался “root”.



## POSIX filesystem API

Есть более гибкое API для управления доступом: ACLs (Access Control Lists):

- к файлу привязывается список пользователей и действий, разрешённых тем пользователям;
- можно наследовать права доступа от родительского каталога.

См. [https://access.redhat.com/documentation/en-US/Red\\_Hat\\_Storage/2.0/html/Administration\\_Guide/ch09s05.html](https://access.redhat.com/documentation/en-US/Red_Hat_Storage/2.0/html/Administration_Guide/ch09s05.html) и [https://linux.die.net/man/5/nfs4\\_acl](https://linux.die.net/man/5/nfs4_acl) .

### 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);
```

## 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);
```

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.

### Что ФС не должна делать, или наблюдение о стоимости дизайн-решений

- Интерпретировать/изменять содержимое файлов
- Интерпретировать/изменять имена файлов

### Что ФС не должна делать

- Интерпретировать/изменять содержимое файлов
- Интерпретировать/изменять имена файлов

### И что делают Windows (VFAT, NTFS) и Mac (HFS)

Case-insensitive & case-preserving FS



### Case-insensitivity: почему так делать не надо

Необходимо сравнивать строки без учёта регистра. Для этого требуется:

- Знание всех языков из Unicode,
- Разбираться с неоднозначностью записи букв: «á» может быть одним символом, а может быть объединением символов «а» и «'»,
- Правило преобразования большой малых букв в заглавные зависит не от алфавита, а от региона в стране,
- В новых версиях Unicode правила нормализации и приведения к заглавным буквам меняются.

### Case-insensitivity: почему так делать не надо

Необходимо сравнивать строки без учёта регистра. Для этого требуется:

- Знание всех языков из Unicode,
- Разбираться с неоднозначностью записи букв: «á» может быть одним символом, а может быть объединением символов «а» и «'»,
- Правило преобразования большой малых букв в заглавные зависит не от алфавита, а от региона в стране,
- В новых версиях Unicode правила нормализации и приведения к заглавным буквам меняются.
- В Windows при создании раздела NTFS в него записываются таблицы нормализации и преобразования регистра.

### Case-insensitivity: почему так делать не надо

Необходимо сравнивать строки без учёта регистра. Для этого требуется:

- Знание всех языков из Unicode,
  - Разбираться с неоднозначностью записи букв: «á» может быть одним символом, а может быть объединением символов «а» и «'»,
  - Правило преобразования большой малых букв в заглавные зависит не от алфавита, а от региона в стране,
  - В новых версиях Unicode правила нормализации и приведения к заглавным буквам меняются.
- 
- **Правильное решение:** не вмешиваться в именование файлов – пусть пользовательские программы заботятся об этом, если им надо.

Дополнение: Как работать с Unicode-строками, будто это null-terminated ASCII-строки: UTF-8

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
1	7	U+0000	U+007F	0xxxxxxx					
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx				
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx			
4	21	U+10000	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
5	26	U+200000	U+3FFFFFFF	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
6	31	U+4000000	U+7FFFFFFF	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

- Достоинства:
- Запись английского языка не меняется,
  - Европейские языки требуют двух байт,
  - При ошибках потока однозначно находятся позиции начала символов.

- Домашнее задание: напишите преобразователи в utf-8 и из него:
- `std::vector<uint8_t> to_utf8(const std::vector<uint32_t> &x)`
  - `std::vector<uint32_t> from_utf8(const std::vector<uint8_t> &x)`