

# Основы построения файловых систем



## 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
```



Group permissions

User permissions

Permissions for others

## POSIX filesystem API

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

```
$ ls -lh /usr/bin
-rwsr-xr-x 1 root root 134K Jan 5 2016 sudo
-rwxrwxrwx 1 root root 4 Jan 5 2016 sudoedit -> sudo
-rwxr-xr-x 1 root root 47K Jan 5 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 5 2016 sudo
-rwxrwxrwx 1 root root 4 Jan 5 2016 sudoedit -> sudo
-rwxr-xr-x 1 root root 47K Jan 5 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, 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, 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://www.kernel.org/doc/Documentation/binfmt\\_misc.txt](https://www.kernel.org/doc/Documentation/binfmt_misc.txt)



### 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) .

### Что ФС не должна делать

- 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.

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.

### 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.

С каталогами похожая ситуация:

```
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
-rwxr-xr-x 1 artem artem  11K Sep 27 21:49 proc
-rw-r--r-- 1 artem artem  1.1K Sep 27 21:49 proc.c
-rwxr-xr-x 1 artem artem  13K Sep 27 20:14 ps
-rw-r--r-- 1 artem artem  1.6K Sep 27 20:13 ps.c

artem@dev:~/testing/students$ sshfs -o nonempty anisimov@vzbuild: ~/testing/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.

С каталогами похожая ситуация:

```
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
-rwxr-xr-x 1 artem artem  11K Sep 27 21:49 proc
-rw-r--r-- 1 artem artem 1.1K Sep 27 21:49 proc.c
-rwxr-xr-x 1 artem artem  13K Sep 27 20:14 ps
-rw-r--r-- 1 artem artem 1.6K Sep 27 20:13 ps.c

artem@dev:~/testing/students$ sshfs -o nonempty anisimov@vzbuild: ~/testing/students/

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

## Bind-mounts

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

```
$ ls -lh ./src/  
total 4.0K  
-rw-r--r-- 1 artem artem 6 Sep 28 11:46 hello  
  
$ ls -lh ./dst/  
total 0  
  
$ sudo mount --bind ./src/ ./dst/  
artem@dev:~/testing/students$ ls -lh ./dst/  
total 4.0K  
-rw-r--r-- 1 artem artem 6 Sep 28 11:46 hello
```

## Bind-mounts

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

```
$ ls -lh ./src/
total 4.0K
-rw-r--r-- 1 artem artem 6 Sep 28 11:46 hello

$ ls -lh ./dst/
total 0

$ sudo mount --bind ./src/ ./dst/
artem@dev:~/testing/students$ ls -lh ./dst/
total 4.0K
-rw-r--r-- 1 artem artem 6 Sep 28 11:46 hello
```

Bind mounts привносят много нетривиальных деталей: <http://lwn.net/Articles/689856/>

POSIX API	Windows API
open(const char *path, int mode, int flags)	HANDLE WINAPI CreateFile( _In_ LPCTSTR lpFileName, _In_ DWORD dwDesiredAccess, _In_ DWORD dwShareMode, _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes, _In_ DWORD dwCreationDisposition, _In_ DWORD dwFlagsAndAttributes, _In_opt_ HANDLE hTemplateFile );
read(int fd, void *buf, size_t count)	BOOL WINAPI ReadFile( _In_ HANDLE hFile, _Out_ LPVOID lpBuffer, _In_ DWORD nNumberOfBytesToRead, _Out_opt_ LPDWORD lpNumberOfBytesRead, _Inout_opt_ LPOVERLAPPED lpOverlapped );
write(int fd, const void *buf, size_t count)	BOOL WINAPI WriteFile( _In_ HANDLE hFile, _In_ LPCVOID lpBuffer, _In_ DWORD nNumberOfBytesToWrite, _Out_opt_ LPDWORD lpNumberOfBytesWritten, _Inout_opt_ LPOVERLAPPED lpOverlapped );
close(int fd)	BOOL WINAPI CloseHandle( _In_ HANDLE hObject );

## POSIX API

Имя объекта в ФС отделено от него

- Можно делать hardlinks,
- Можно создавать файлы без имени.

## Windows API

Имя и файл (каталог) существуют только в паре

- Нельзя удалить открытый файл.

## POSIX API

```
open(const char *path, int mode, int flags)
```

```
read(int fd, void *buf, size_t count)
```

```
write(int fd, const void *buf, size_t count)
```

```
close(int fd)
```

## Windows API

```
HANDLE WINAPI CreateFile(
    _In_      LPCTSTR          lpFileName,
    _In_      DWORD            dwDesiredAccess,
    _In_      DWORD            dwShareMode,
    _In_opt_  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    _In_      DWORD            dwCreationDisposition,
    _In_      DWORD            dwFlagsAndAttributes,
    _In_opt_  HANDLE           hTemplateFile
);
```

```
BOOL WINAPI ReadFile(
    _In_      HANDLE          hFile,
    _Out_     LPVOID          lpBuffer,
    _In_      DWORD           nNumberOfBytesToRead,
    _Out_opt_ LPDWORD         lpNumberOfBytesRead,
    _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

```
BOOL WINAPI WriteFile(
    _In_      HANDLE          hFile,
    _In_      LPCVOID         lpBuffer,
    _In_      DWORD           nNumberOfBytesToWrite,
    _Out_opt_ LPDWORD         lpNumberOfBytesWritten,
    _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

```
BOOL WINAPI CloseHandle(
    _In_ HANDLE hObject
);
```



## Синхронный ввод-вывод

Диск, если начал операцию, не прерывает её до тех пор, пока она не завершится.

API для работы с файлами сохранили это же свойство – они не отдают управление, пока не завершатся.

```
char buf[128 * 1024];
int fd_src = open("source_file", O_RDONLY);
int fd_dst = open("destination_file", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);

for (;;) {
    int r = read(fd_src, buf, sizeof(buf));
    write(fd_dst, buf, sizeof(buf));

    if (r < sizeof (buf))
        break;
}

close(fd_dst);
close(fd_src);
```

Синхронный ввод-вывод

```
for (;;) {  
  int r = read(fd_src, buf, sizeof(buf));  
  write(fd_dst, buf, sizeof(buf));  
}
```





