Основы построения файловых систем

Дополнение 1. 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+3FFFFFF	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)

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

Домашнее задание:

- man 5 proc,
- напишите аналоги
  - ps
  - lsof

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
-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 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
-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 aanisimov@vzbuild ~/testings/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 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
artem@dev:~/testing/bind-mount$
```

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

Bind mounts привносят много нетривиальных деталей:

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

Memory-mapped files

```
int fd = open(“file.txt”, O_RDONLY);
char *str = mmap(NULL, length, PROT_READ, MAP_PRIVATE, fd, 0);

/* work with @str as if it were an array */
printf(“%s\n”, str);

munmap(str, length);
```

Как это работает?

Виртуальная память: зачем это надо?

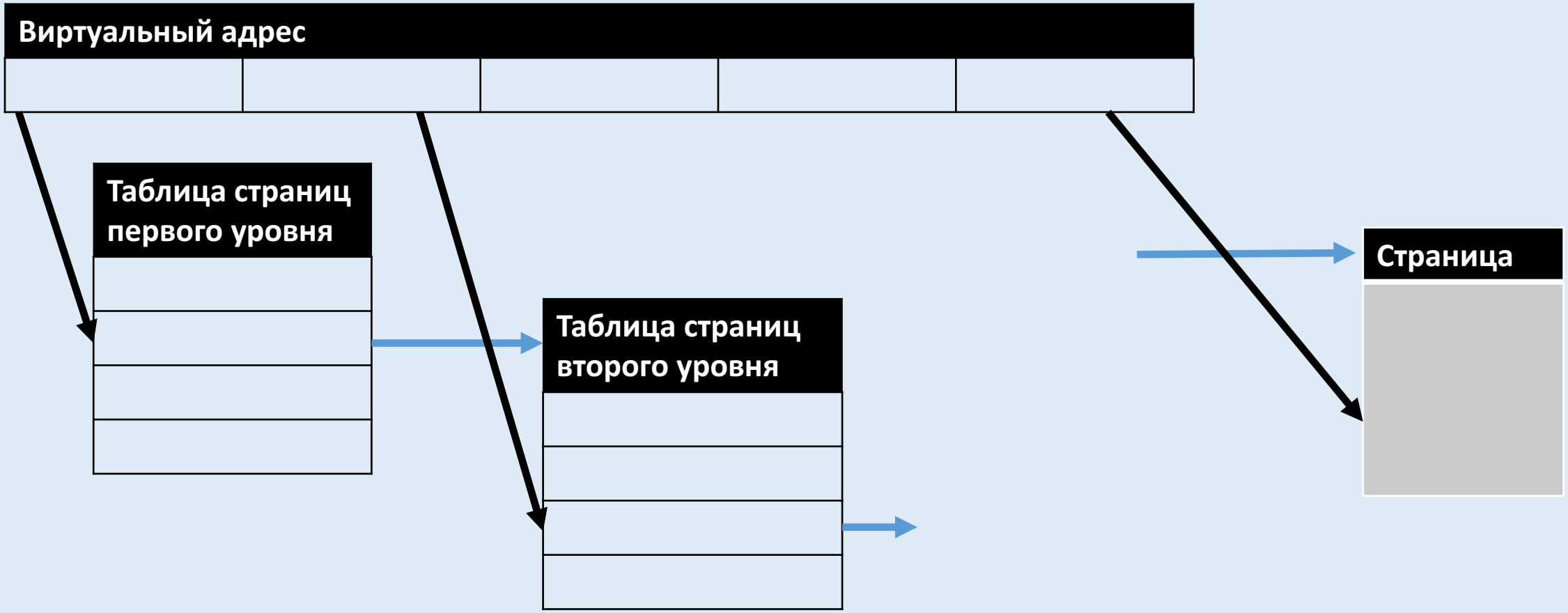
Процессы не имеют доступа к физической памяти.

Вместо этого, ОС предоставляют процессам линейное адресное пространство,
которое может произвольно отображаться на физическую память.

Задачи, которые решает введение виртуального адресного пространства:

1. Возможность предоставить каждому процессу единообразное адресное пространство:
   процесс просто считает, что ему доступны все адреса в диапазоне [0, MAX_ADDR),
2. Изоляция процессов,
3. Возможность прозрачно разделять часть памяти между процессами (shared libraries, text segments, etc.),
4. Возможность «незаметно» для процесса заполнять/выгружать его части из памяти:
   memory-mapped files, swapping,

Виртуальная память с точки зрения CPU



- Таблицы разрешается заполнять частично, чтобы не тратить много памяти.
- Поиск по таблицам требует много обращений к памяти, поэтому результаты преобразований адресов кешируются в TLB (Translation Look-aside Buffer)

Виртуальная память с точки зрения ОС

Для операционной системы память процесса представляется как набор VMA (Virtual Memory Area).

Каждая VMA указывает

- диапазон адресов,
- права доступа (и флаги вроде copy-on-write),
- правило, как подгружать страницы из данной VMA.

Memory-mapped files: проблемы

Если файл виден как массив в памяти, то чтение и запись делаются очень просто.

Но как

1. увеличивать размер файла?
2. обрабатывать ошибки чтения из файла?
3. обрабатывать ошибки записи в файл?

Memory-mapped files: проблемы

Если файл виден как массив в памяти, то чтение и запись делаются очень просто.

Но как
1. увеличивать размер файла?
2. обрабатывать ошибки чтения из файла?
3. обрабатывать ошибки записи в файл?

Ответ: никак.

До недавнего времени ошибки при отложенной записи (writeback) можно было легко потерять:
- https://lwn.net/Articles/718734/
- http://stackoverflow.com/q/42434872/398670

Page cache и отложенная запись (writeback)

Аналогичные проблемы с записью есть и в POSIX API:

```
int fd = open(“file.txt”, O_RDWR);
pwrite(fd, buf, size, 0);
fsync(fd);
close(fd);
```

Вызов pwrite() не записывает данные в файл, а только помещает их в page cache.
Данные будут записаны на диск только после вызова fsync() или когда ОС решит сбросить page cache на диск.

- ошибки записи будут возвращены из fsync()
- close() тоже может завершаться с ошибкой.

Page cache и отложенная запись (writeback)

fsync() и fdatasync()

- могут сказать, что записать данные не удалось,
- не указывают диапазон страниц, которые не удалось записать.

Как с этим бороться?

Page cache и отложенная запись (writeback)

fsync() и fdatasync()

- могут сказать, что записать данные не удалось,
- не указывают диапазон страниц, которые не удалось записать.

Как с этим бороться?

Упорядочивать записи в файл:

1. записать новые данные,
2. fsync(),
3. записать заголовок, который ссылается на новые данные,
4. fsync().

Page cache и отложенная запись (writeback)

fsync() и fdatasync()

- могут сказать, что записать данные не удалось,
- не указывают диапазон страниц, которые не удалось записать.

Как с этим бороться?

Упорядочивать записи в файл:

1. записать новые данные,
2. fsync(),
3. записать заголовок, который ссылается на новые данные,
4. fsync().

Как быть с перезаписями?

1. append-only files (только append и punch holes),
2. следить за использованием областей и перезаписывать только те, которые не используются.

Master-slave репликация для append-only файлов

Пусть у нас есть файловый сервер, которые предоставляет следующие операции:

- прочесть данные из файла,
- дописать данные в конец файла,
- превратить часть файла в дырку.

Как для такого файлового сервера добавить возможность асинхронной репликации? Файл на реплике всегда должен быть в согласованном состоянии.

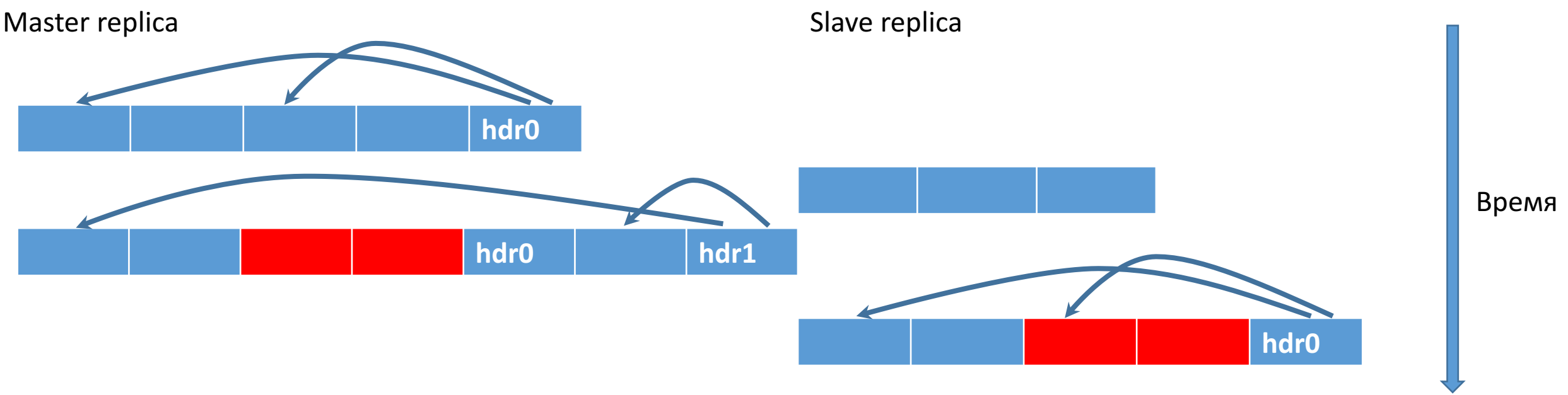
Master-slave репликация для append-only файлов

Пусть у нас есть файловый сервер, которые предоставляет следующие операции:

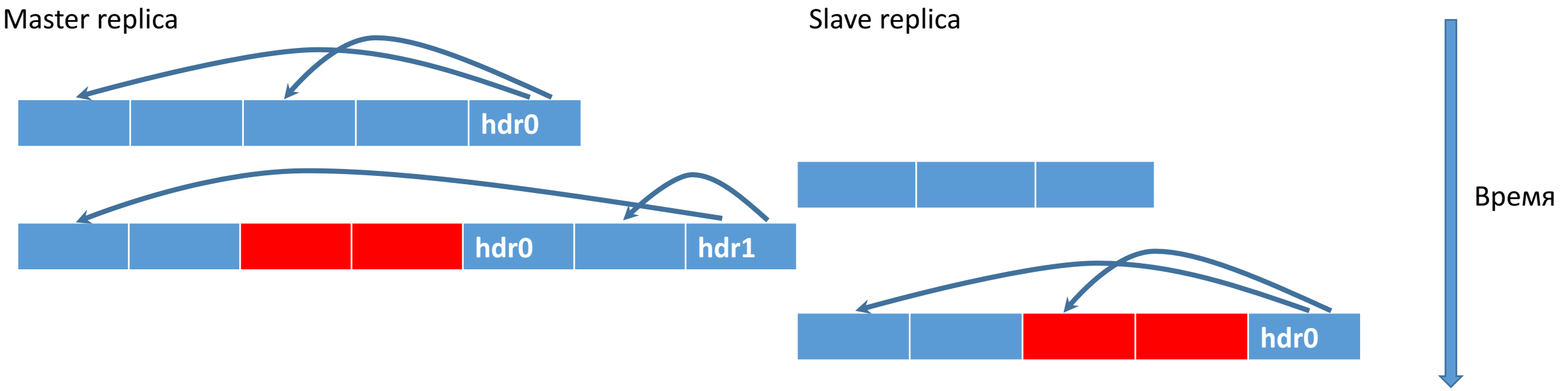
- прочесть данные из файла,
- дописать данные в конец файла,
- превратить часть файла в дырку.

Как для такого файлового сервера добавить возможность асинхронной репликации? Файл на реплике всегда должен быть в согласованном состоянии.

В чём проблема:



Master-slave репликация для append-only файлов



Вывод: исполнять punch_holes сразу при получении запроса нельзя, их надо журналировать и исполнять позже.

Домашнее задание: придумайте механизм журналирования дырок для master-slave репликации append-only файлов.

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	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);

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.

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