

Лекция 22

Файловая система

Просмотр каталогов

```
#include <sys/types.h>
```

```
#include <dirent.h>
```

```
DIR *opendir(const char *name);
```

```
int closedir(DIR *dirp);
```

```
struct dirent *readdir(DIR *dirp);
```

Сложности

- Файловая система может изменяться одновременно несколькими процессами
- Недопустимо блокировать каталоги файловой системы на длительное время
- Формат хранения данных в каталоге оптимизирован для большого количества файлов (ext4 – B-tree)

Telldir/seekdir

- Позволяют получить текущую “позицию” в каталоге и вернуться к ранее полученной “позиции” в каталоге
- “Позиция” - на самом деле какой-то хеш от имени
- В стандарте POSIX есть оговорка: только для “the same directory stream”, то есть закрывать каталог в промежутке между telldir и seekdir нельзя
- В современном Linux этой оговорки нет в man, но есть в документации на glibc
- Следует придерживаться консервативного правила: **seekdir после closedir и opendir не работает**

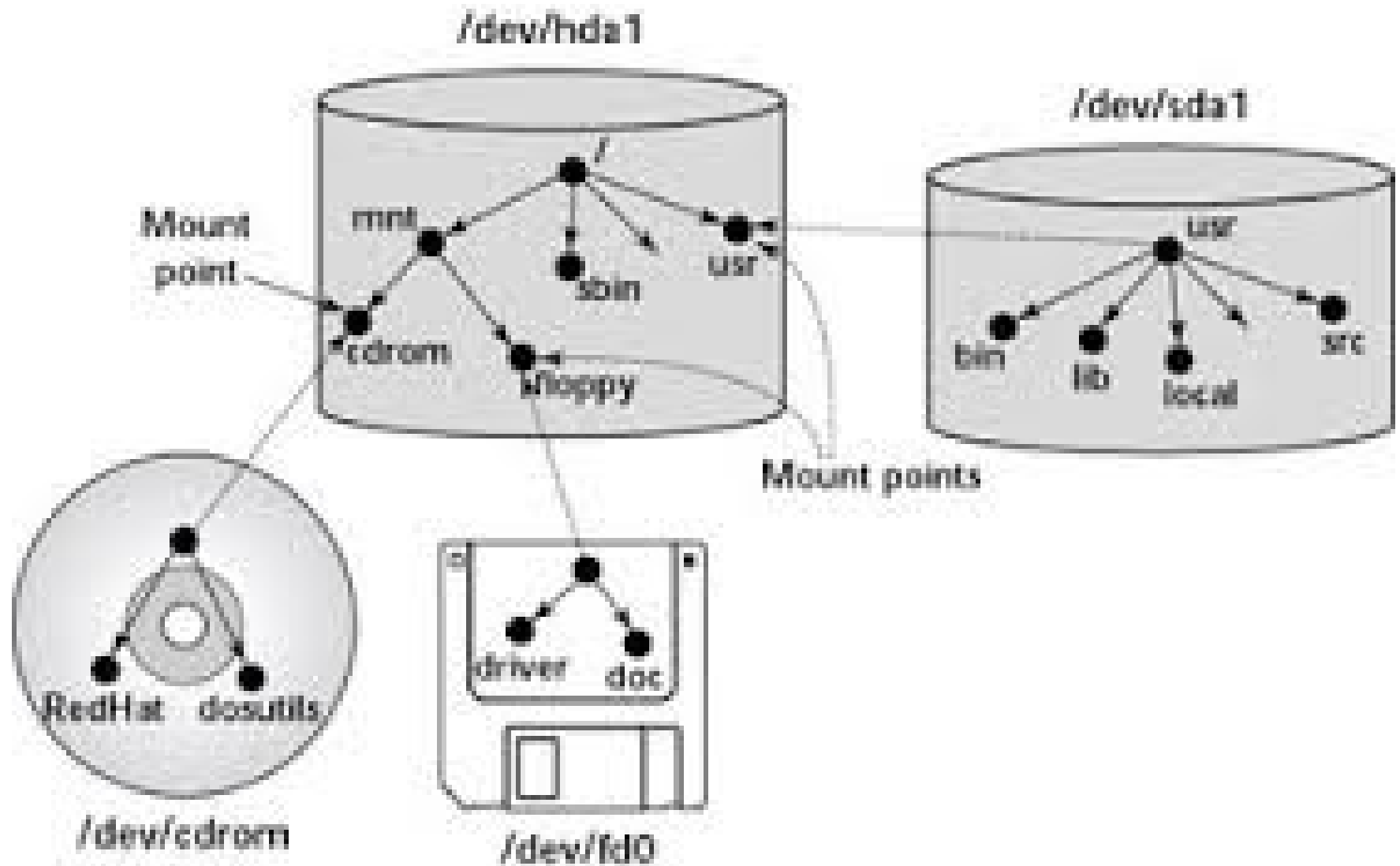
Требования к readdir

- Если файл существовал и не был удален за время сканирования каталога, информация о нем должна быть получена readdir
- Информация о любом файле должна быть получена не более одного раза

readdir

- Readdir возвращает указатель на структуру с информацией о текущей записи
- Readdir может использовать одну и ту же область памяти каждый раз
- Нельзя предполагать, что вызов readdir не испортит память по указателю, полученному предыдущим вызовом readdir
- Каталоги могут меняться одновременно с чтением, при использовании readdir + (l)stat нельзя предполагать, что файл еще будет существовать к моменту (l)stat

Монтирование



Пример монтирования

\$ mount

```
/dev/md0 on / type ext4 (rw,relatime)
```

```
/dev/md1 on /home type ext4 (rw,noatime,nodiratime)
```

Homepa st_dev:st_ino

```
2304:2 /
```

```
2305:2 /home
```

```
2305:2 /home/.
```

```
2304:2 /home/..
```


Файловые системы /proc и /sys

- Ядро предоставляет доступ к информации о состоянии системы с помощью этих псевдофайловых систем
- Структуры данных этих файловых систем существуют только в памяти ядра
- /proc более «историческая»
- /sys более «структурированная»

/proc

- Информация о процессах /proc/\${PID}:
 - fd — открытые файловые дескрипторы
 - exe — путь к исполняемому файлу
 - cwd — текущий рабочий каталог
 - maps — карта памяти
- Информация о системе
 - interrupts — прерывания
 - modules — загруженные модули ядра

Модификация параметров

- `/proc/sys/kernel/core_pattern` — шаблон для имени core-файла

```
cat /proc/sys/kernel/core_pattern
```

```
echo 'mycore' > /proc/sys/kernel/core_pattern
```

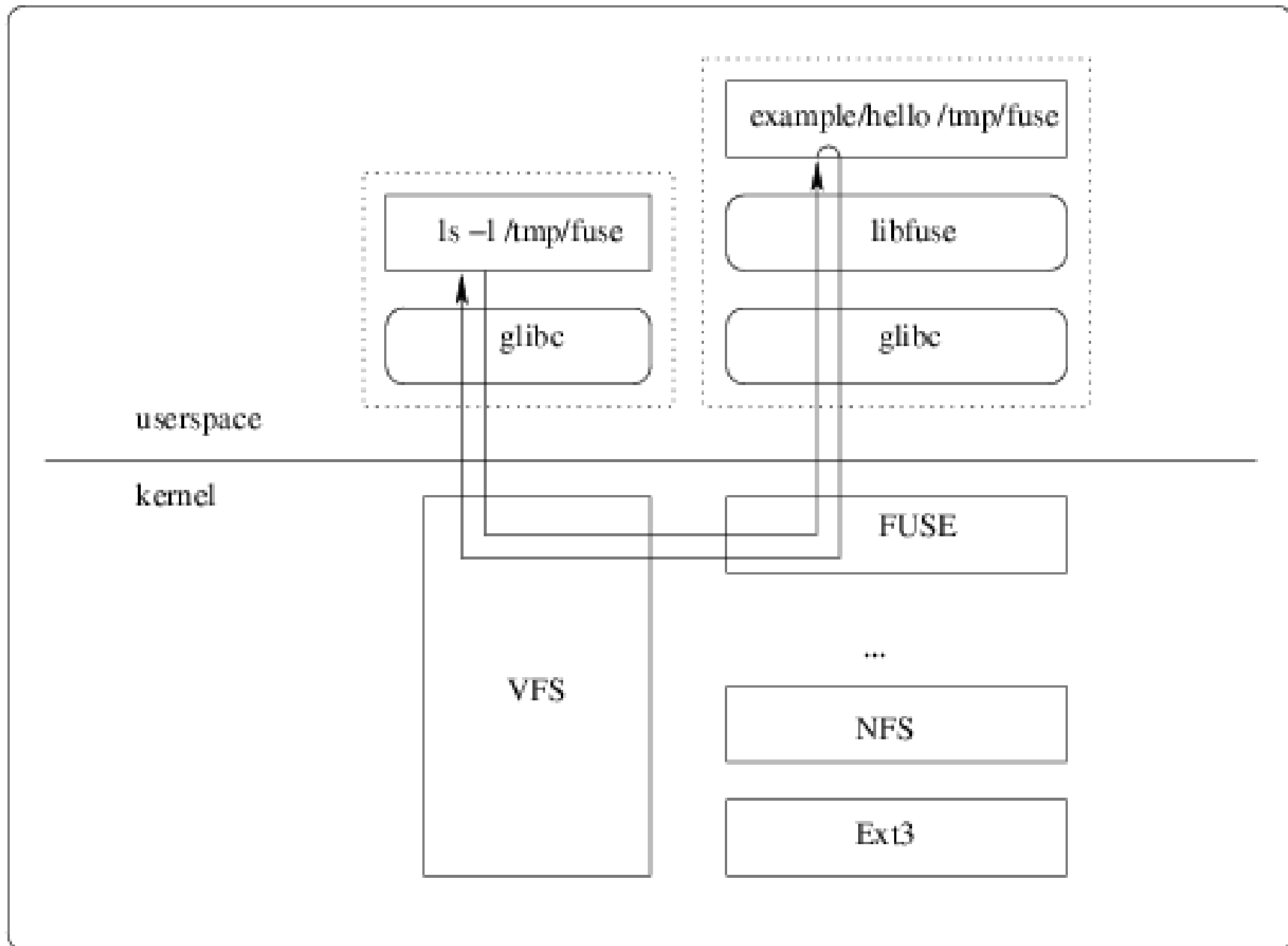
FUSE (Filesystem in userspace)

- Позволяет реализовать драйвер файловой системы в пользовательском процессе
 - Надежность: ошибка в драйвере не приводит к краху ядра
 - Гибкость: можно менять без перезагрузки, права обычного пользователя
 - Безопасность: смонтировавший пользователь может ограничить доступ
- Применения: fuse-sshfs, ntfs

Архитектура FUSE

- Файловая система fuse — модуль ядра Linux
 - Запросы к VFS собираются в пакеты данных и пересылаются пользовательскому процессу через `/dev/fuse`
 - Ответы отправляются обратно в VFS
- Библиотека `libfuse` для драйверов ФС
- Привилегированная утилита монтирования `fusermount`

FUSE



Управление памятью

- С точки зрения процесса
 - Адресное пространство процесса
 - Управление адресным пространством
 - Отображаемые файлы в память
 - Динамические (разделяемые) библиотеки
- С точки зрения ядра
 - Управление виртуальной адресацией
 - Управление страничным/буферным кешем

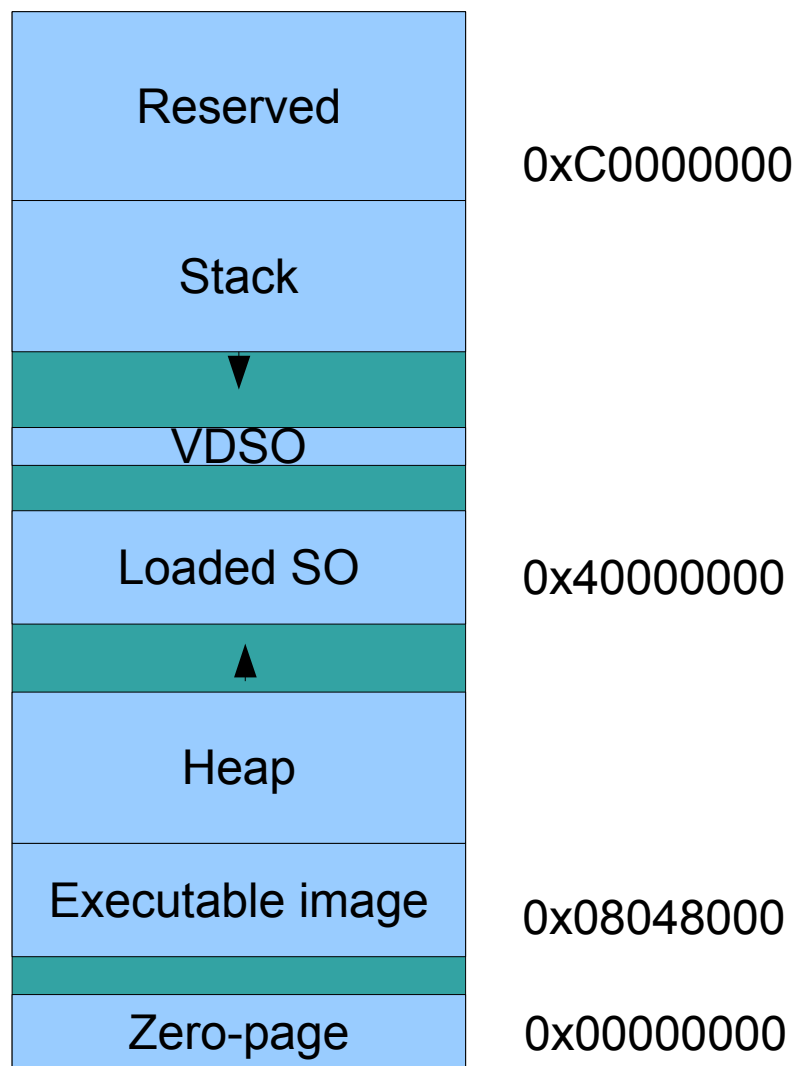
Адресное пространство процесса

- Каждый процесс работает в своем изолированном виртуальном адресном пространстве
- Иллюзия того, что процесс монопольно владеет всей памятью
- Пример: x86 — 32-битное адресное пространство, $2^{32} = 4\text{GiB}$
- X64 — 48-битное адресное пространство, $2^{48} = 256\text{TiB}$
- Процессор x86 в 32-битном режиме может работать с $> 4\text{GiB}$ ОЗУ, но не более 4GiB на процесс

Адресное пространство x86

- Указатели — 32-битные
- Диапазон адресов: 0x00000000 — 0xffffffff
- Как правило, ОС не дает использовать все 4 GiB:
 - Linux: 3GiB доступны, 1GiB зарезервирован
 - Win32: 2GiB / 2GiB
- Попытка обращения в зарезерв. область — segmentation fault
- В зарезервированный 1GiB (не доступный из user-space) каждого процесса отображается память ядра — ускорение переключения user->kernel

Адресное пространство процесса



- Нулевая страница — защита от обращений по указателю NULL
- Стек расширяется ВНИЗ автоматически
- Куча растет вверх по запросу
- Текущее состояние карты памяти:
`/proc/${PID}/maps`

Адресное пространство

- VDSO — спец. разделяемая библиотека — ускорение частых системных вызовов (time, gettimeofday, etc)
- Исполняемый образ — ELF-файл, отображенный на память. Состоит из секций:
 - .text — секция кода, read-only, executable — содержит инструкции программы и константные данные
 - .data — секция данных, read-write
 - .bss — секция данных, инициализированных 0
- Каждый SO-файл (разделяемая библиотека) — ELF-файл, отображаемый в память

/proc/\${PID}/maps

45e55000-45e74000 r-xp 00000000 08:02 1508434 /usr/lib/ld-2.17.so

45e74000-45e75000 r--p 0001e000 08:02 1508434 /usr/lib/ld-2.17.so

- Диапазон виртуальных адресов отображения
- Права: rwx, p — private COW mapping, s — shared
- Смещение в файле
- Major:Minor Inode
- Путь к файлу

/proc/\${PID}/status

- Статистика работы процесса, в т. ч. по памяти

VmPeak:	4300	kB	// макс. Размер VM
VmSize:	4300	kB	// текущий размер VM
VmLck:	0	kB	// locked in memory
VmPin:	0	kB	// pinned in memory
VmHWM:	456	kB	// макс. RSS
VmRSS:	456	kB	// resident set size
VmData:	156	kB	// размер данных
VmStk:	136	kB	// размер стека
VmExe:	48	kB	// размер исп. файла
VmLib:	1884	kB	// размер SO-библиотек
VmPTE:	24	kB	// размер таблиц страниц
VmSwap:	0	kB	// использование swap

Статистика использования памяти

- Virtual Memory Size — суммарный размер отображенных страниц виртуальной памяти
- Resident Set Size — размер страниц, находящихся в оперативной памяти
- Страницы могут находиться:
 - В ОЗУ
 - В swap-файле
 - В файле (исполняемого файла или SO)
 - Нигде (overcommit)

Ограничения адресного пространства

- Команда `ulimit` — установка ограничений процесса

core file size	(blocks, -c)	0
data seg size	(kbytes, -d)	unlimited
scheduling priority	(-e)	0
file size	(blocks, -f)	unlimited
pending signals	(-i)	57326
max locked memory	(kbytes, -l)	32
max memory size	(kbytes, -m)	unlimited
open files	(-n)	1024
pipe size	(512 bytes, -p)	8
POSIX message queues	(bytes, -q)	819200
real-time priority	(-r)	0
stack size	(kbytes, -s)	8192
cpu time	(seconds, -t)	unlimited
max user processes	(-u)	1024
virtual memory	(kbytes, -v)	unlimited
file locks	(-x)	unlimited

Ограничения адресного пространства

- Системные вызовы `setrlimit/getrlimit`
- Жесткий лимит (hard limit) — нельзя превышать
- Мягкий лимит (soft limit) — процесс может увеличивать и уменьшать
- `RLIMIT_AS` — лимит адресного пространства
- `RLIMIT_STACK` — лимит размера стека

Типы страниц в памяти

- Выгружаемые (страница может быть выгружена в область подкачки)
- Невыгружаемые (locked) — должны находиться в ОЗУ
- Процесс может пометить часть страниц как невыгружаемые (системный вызов `mlock`)
- Непривилегированный — макс. 32 KiB
- Все страницы ядра — невыгружаемые