История создания UNIX

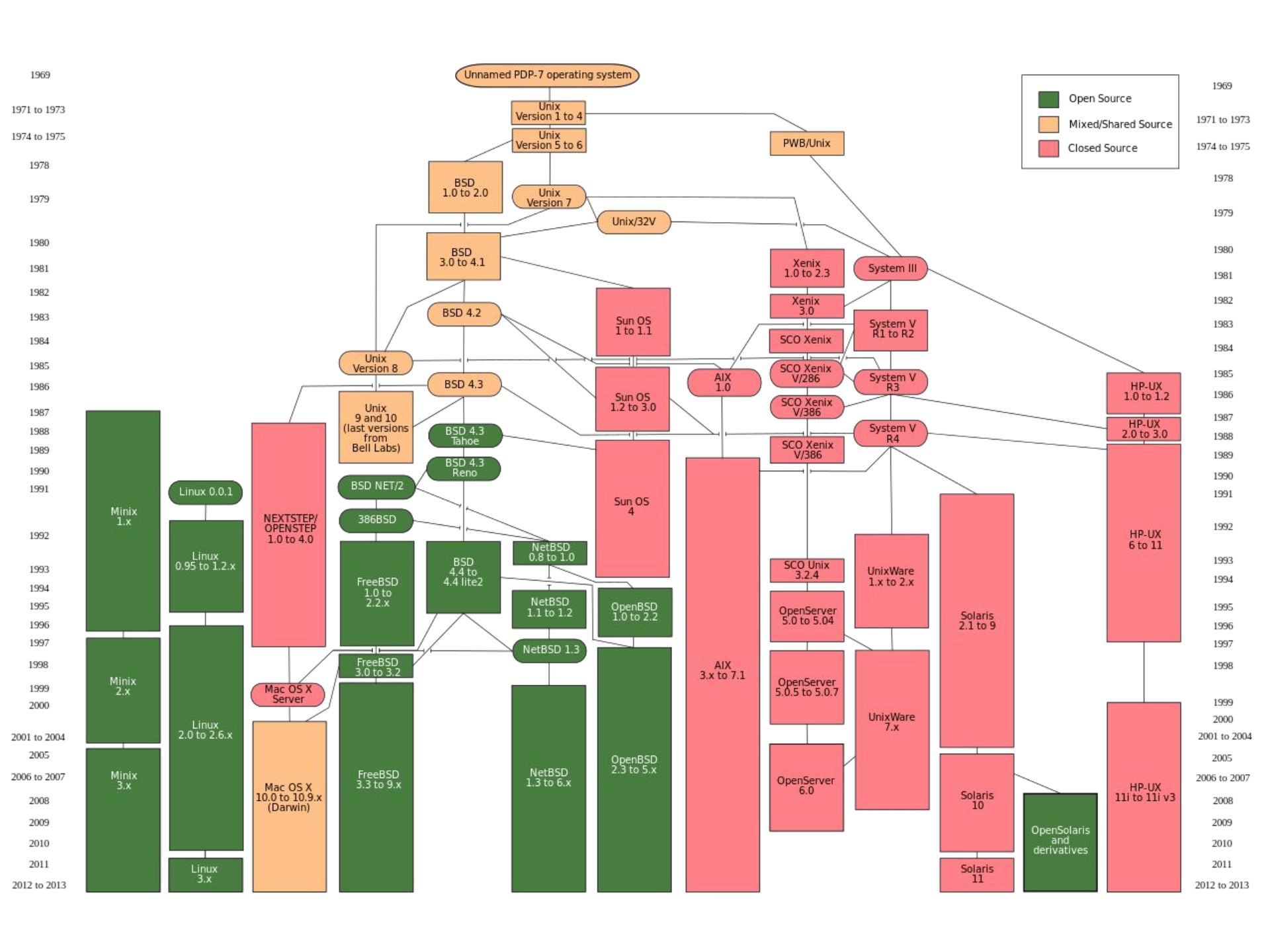
1. Создание CTSS (Compatible Time Sharing System) в МТИ (1961 г.).
2. Создание MULTICS (Multiplexed Information and Computer Service), язык EOL (PL/1), МТИ + Bell Labs + General Electric, 1963 г.
3. Разработка усеченного варианта MULTICS на ассемблере для PDP-7 (UNICS – Uniplexed Information and Computer Service) – Кен Томпсон (1.01.1970).
4. Создание языка высокого уровня B и переработка Unix на этом языке – Томпсон.

Создание языка C (наследник B) – Ритчи.
6. Переписывание UNIX на C – Томпсон и Ритчи.
7. Статья Томпсона и Ритчи об ОС UNIX, 1974 г.
8. Версия 6 UNIX – 8200 строк C + 900 строк ассемблера –1974 г.
9. Первая переносимая версия UNIX (версия 7) –18000 строк C + 2110 строк ассемблера –1976 г.
10. Выпуск коммерческой версии UNIX фирмой AT&T (System III) – 1984 г., а затем UNIX System V.
11. Развитие UNIX Калифорнийским университетом в Беркли – 1BSD (Fist Berkeley Software Distribution), затем 2BSD, 3BSD, 4BSD.
12. Широкое распространение UNIX – Xenix, Minix, AIX, Sun OS, Solaris, Linux.

Дуг Макилрой, изобретатель каналов UNIX и один из основателей традиции UNIX, обобщил философию следующим образом:
«Философия UNIX гласит:
Пишите программы, которые делают что-то одно и делают это хорошо.
Пишите программы, которые бы работали вместе.
Пишите программы, которые бы поддерживали текстовые потоки, поскольку это универсальный интерфейс».

«UNIX прост. Но надо быть гением, чтобы понять его простоту» — Деннис Ритчи.

«UNIX не предназначен для ограждения своих пользователей от глупостей, поскольку это оградило бы их и от умных вещей» — Дуг Гвин.


1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2007
2008
2009
2010
2011
2012 to 2013
Open Source
Mixed/Shared Source
Closed Source
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
Unix Version 7
PWB/Unix
BSD 1.0 to 2.0
Unix/32V
BSD 3.0 to 4.1
BSD 4.2
BSD 4.3
Unix Version 8
Unix 9 and 10 (last versions from Bell Labs)
BSD 4.3 Tahoe
BSD 4.3 Reno
BSD NET/2
386BSD
FreeBSD 1.0 to 2.2.x
FreeBSD 3.0 to 3.2
FreeBSD 3.3 to 9.x
BSD 4.4 to 4.4 lite2
NetBSD 0.8 to 1.0
NetBSD 1.1 to 1.2
NetBSD 1.3
NetBSD 1.3 to 6.x
OpenBSD 1.0 to 2.2
OpenBSD 2.3 to 5.x
Sun OS 1 to 1.1
Sun OS 1.2 to 3.0
Sun OS 4
Xenix 1.0 to 2.3
Xenix 3.0
SCO Xenix
SCO Xenix V/286
SCO Xenix V/386
SCO Unix 3.2.4
OpenServer 5.0 to 5.04
OpenServer 5.0.5 to 5.0.7
OpenServer 6.0
System III
System V R1 to R2
System V R3
System V R4
AIX 1.0
AIX 3.x to 7.1
UnixWare 1.x to 2.x
UnixWare 7.x
Solaris 2.1 to 9
Solaris 10
Solaris 11
OpenSolaris and derivatives
HP-UX 1.0 to 1.2
HP-UX 2.0 to 3.0
HP-UX 6 to 11
HP-UX 11i to 11i v3
Minix 1.x
Minix 2.x
Minix 3.x
Linux 0.0.1
Linux 0.95 to 1.2.x
Linux 2.0 to 2.6.x
Linux 3.x
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Mac OS X Server
Mac OS X 10.0 to 10.9.x (Darwin)

Общая характеристика системы UNIX

ОС UNIX – интерактивная система, разработанная программистами и для программистов. Основные требования: простота, элегантность, непротиворечивость, мощь и гибкость.

Общие черты Unix независимо от версии:

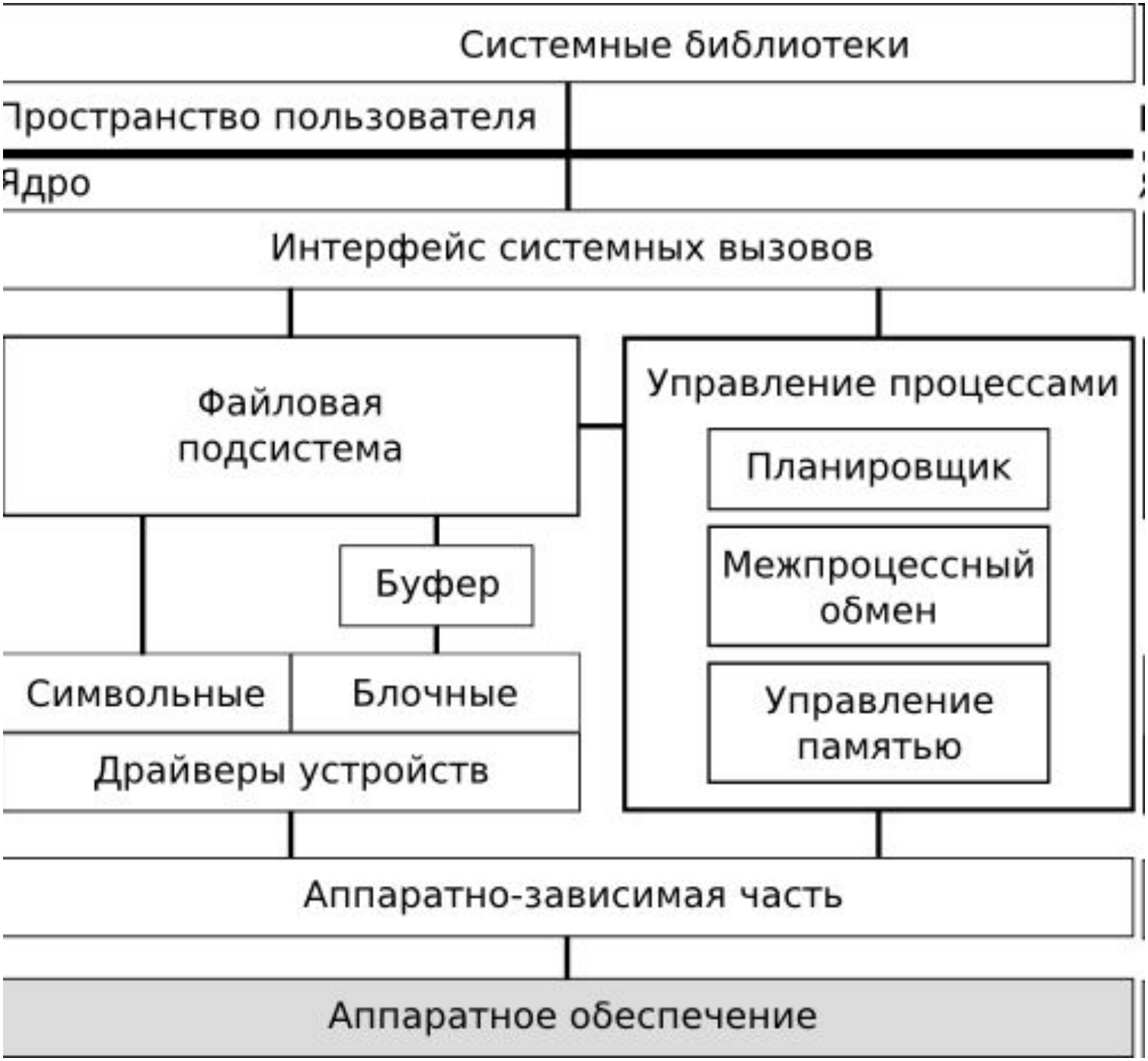
1. Многопользовательский режим со средствами защиты от несанкционированных пользователей.
2. Реализация мультипрограммной работы в режиме разделения времени, основанная на использовании алгоритмов вытесняющей многозадачности.
3. Использование механизмов виртуальной памяти и свопинга для повышения уровня мультипрограммирования.
4. Унификация ввода-вывода на основе расширенного использования понятия файл.
5. Иерархическая файловая система, образующая единый граф каталогов независимо от количества физических устройств, используемых для размещения файлов.
6. Переносимость системы за счет написания ее основной части на языке С.
7. Разнообразные средства взаимодействия процессов, в том числе через сеть.
8. Кэширование дисков для уменьшения среднего времени доступа к файлам.

Интерфейс системы UNIX

Структура ядра системы Unix

Ядро операционной системы

- Ядро ОС – низкоуровневая программа компьютера. Для большинства устройств, ядро – единственная программа, имеющая доступ. Доступ к устройствам осуществляется с помощью драйверов.
- Ядро реализует файловую систему, управляет памятью, контролирует выполнение программ, управляет доступом к сети.
- Ядро создается при инсталляции системы и хранится в специальном файле /kernel
- При загрузке системы можно изменить параметры ядра:
  - boot [-опции] [ядро]
- При необходимости можно загрузить отдельные модули ядра:
  - kldload [модуль]
- выгрузка
  - kldunload [модуль]
- Список модулей хранится в специальном каталоге, например, **/modules**


Системные библиотеки
Пространство пользователя
Ядро
Интерфейс системных вызовов
Файловая подсистема
Управление процессами
Планировщик
Межпроцессный обмен
Управление памятью
Буфер
Символьные
Блочные
Драйверы устройств
Аппаратно-зависимая часть
Аппаратное обеспечение

Оболочка системы UNIX

Система поддерживает графическое окружение X Windows, но многие программисты предпочитают интерфейс командной строки, создавая множество консольных окон и действуя так, как если бы у них было несколько алфавитно-цифровых терминалов, на каждом из которых работала бы оболочка (shell).

Существует много различных оболочек: sh, ksh, bash и др. После запуска оболочка печатает на экране символ приглашения к вводу (% или $) и ждет, когда пользователь введет командную строку.

Примеры командных строк:

1) cp file1 file2 (копировать файл file1, копия – file2)

2) head –20 file (печатать первые 20 строк файла file)

3) sort < in > out (программе sort взять в качестве входного файла in и направить вывод в файл out)

4) sort < in > temp; head -30 < temp; rm temp

5) sort < in | head -30 (канал)

6) sort < x | head & (фоновый процесс)

Файлы, содержащие команды оболочки,называются *сценариями оболочки. В них можно использовать конструкции if, for, while, case.*

Утилиты системы Unix

Кроме оболочки пользовательский интерфейс содержит большое число обслуживающих программ (утилит):

1. Программы (команды) управления файлами и каталогами.
2. Фильтры.
3. Средства разработки программ (текстовые редакторы, компиляторы).
4. Текстовые процессоры.
5. Системное администрирование.
6. Разное.

Процессы в ОС UNIX

Процесс в ОС создается при запуске приложения со стороны пользователя или самой ОС. Для каждого процесса ОС характерны совокупность набора команд процессора и ассоциированных ресурсов – адресное пространство, стеки, используемые файлы и устройства ввода-вывода и т. п.

Различают:

- **независимые процессы** – используют ресурсы, но не обмениваются информацией;
- **взаимодействующие процессы** – обмениваются информацией, либо их выполнение синхронизировано.

Процессы взаимодействуют с помощью специальных механизмов:

- сигналы;
- программные каналы;
- разделяемая память;
- семафоры;
- сообщения;
- общие файлы.

Атрибуты процесса

- Каждый процесс характеризуется набором атрибутов. К их числу относятся:
  - PID – идентификатор процесса
  - PPID – идентификатор родительского процесса
  - UID, GID – идентификаторы пользователя и группы
  - TT – управляющий терминал (процессы не связанные с управляющими терминалами называются демонами)
  - SID – идентификатор сессии, устанавливается равным PID лидера сессии;
  - NICE – приоритет процесса (относительный приоритет)
  - TIME – процессорное время.

Атрибуты процесса

Таблица, содержащая список процессов имеет примерно следующий вид:

USER	PID	%CPU	%MEM	VSZ	RSS	TT	STAT	STARTED	TIME	COMMAND
dima	1731	0.0	1.6	1080	932	p0	R+	3:15PM	0:00.00	-bash (bash)
root	1	0.0	0.4	552	212	??	ILs	Tue12PM	0:00.04	/sbin/init --
root	2	0.0	0.0	0	0	??	DL	Tue12PM	0:00.31	(pagedaemon)
root	3	0.0	0.0	0	0	??	DL	Tue12PM	0:00.00	(vmdaemon)
root	4	0.0	0.0	0	0	??	DL	Tue12PM	0:01.24	(bufdaemon)
root	5	0.0	0.0	0	0	??	DL	Tue12PM	0:01.81	(vnlru)
root	6	0.0	0.0	0	0	??	DL	Tue12PM	1:35.73	(syncer)
root	60	0.0	0.4	448	248	??	Ss	Tue12PM	0:21.35	/sbin/natd -u -m -
root	76	0.0	0.9	944	544	??	Is	Tue12PM	0:01.17	/usr/sbin/syslogd
root	87	0.0	1.1	1076	620	??	Is	Tue12PM	0:00.02	/usr/sbin/inetd -w
root	89	0.0	1.0	996	592	??	Is	Tue12PM	0:01.39	/usr/sbin/cron
root	91	0.0	2.4	2740	1404	??	Is	Tue12PM	0:04.09	/usr/sbin/sshd
root	94	0.0	2.8	2788	1664	??	Ss	Tue12PM	0:14.07	sendmail: acceptin
smmsp	97	0.0	2.6	2660	1564	??	Is	Tue12PM	0:00.27	sendmail: Queue ru
drweb	217	0.0	3.6	2652	2132	??	Is	Tue12PM	0:00.00	/usr/local/drweb/d
drweb	222	0.0	1.1	1380	640	??	Ss	Tue12PM	0:06.06	/usr/local/sbin/dr
root	227	0.0	0.9	948	532	v1	Is+	Tue12PM	0:00.02	/usr/libexec/getty

У каждого пользователя системы может быть одновременно несколько активных процессов, кроме того существуют десятки фоновых процессов (демонов). Типичный демон – ***cron daemon,*** предназначенный для планирования и запуска процессов. Системный вызов ***fork*** создает точную копию исходного процесса, называемого **родительским процессом**. Новый процесс называется потомком. У процессов собственные образы памяти. Открытые файлы и др.ресурсы используются совместно.

```
pid = fork ( );  /* если fork завершился успешно, pid > 0 в родит. процессе */

if (pid < 0) {

handle_error ( );  /* fork потерпел неудачу (например, память переполнена)*/

} else if (pid > 0) {

                    /* здесь располагается родительская программа */

} else {

                    /* здесь располагается программа-потомок */

}
```

Механизм создания нового процесса : для процесса-потомка создается новая ячейка в таблице процессов, которая заполняется по большей мере из соответствующей ячейки родительского процесса. Процесс-потомок получает PID, затем настраивается его карта памяти. Кроме того, процессу-потомку предоставляется совместный доступ к файлам родительского процесса. Затем настраиваются регистры процесса-потомка, после чего он готов к запуску.

В принципе, следует создать полную копию адресного пространства, так как семантика системного вызова fork говорит, что никакая область памяти не используется совместно родителем и потомком. Однако копирование памяти является дорогим удовольствием, поэтому все системы UNIX слегка жульничают. Они выделяют процессу–потомку новые таблицы страниц, но эти таблицы указывают на страницы родительского процесса, помеченные как доступные только для чтения. Когда Потомок пытается писать в такую страницу, происходит прерывание. При этом ядро выделяет процессу-потомку новую копию этой страницы, к которой этот процесс получает также и доступ записи.

Таким образом, копируются только те страницы, в которые потомок пишет новые данные. Такой механизм называется копированием при записи. При этом сохраняется память, так как страницы с программой не копируются.

После того как процесс-потомок начинает работу, его программа (копия оболочки) выполняет системный вызов exec, задавая имя команды в качестве параметра. При этом ядро находит и проверяет исполняемый файл, копирует в ядро аргументы и строки окружения, а также освобождает старое адресное пространство и его таблицы страниц.

Процессы взаимодействуют с помощью каналов. Синхронизация процессов достигается путем блокировки процесса при попытке прочитать данные из пустого канала. Например, когда оболочка выполняет строку **sort < f | head** она создаст два процесса, **sort** и **head,** и устанавливает между ними канал. Если канал переполняется, система приостанавливает работу **sort**, пока **head** не удалит хоть сколько-нибудь данных. **Процессы могут взаимодействовать также при помощи программных прерываний посылкой сигналов**.

Для управления процессами используются системные вызовы.

Системный вызов	Описание
pid = fork ( )	Создать дочерний процесс, идентичный родительскому
pid = waitpid (pid, &statloc, opts)	Ждать завершения дочернего процесса
s = execve (name, argv, envp)	Заменить образ памяти процесса
exit (status)	Завершить выполнение процесса и вернуть статус
s = sigaction (sig, &act, &oldact)	Определить действие, выполняемое при приходе сигнала
s = sigreturn (&context)	Вернуть управление после обработки сигнала
s =sigprocmask (how, &set, &old)	Исследовать или изменить маску сигнала
s = sigpending (set)	Получить или установить блокированные сигналы s =
sigsushtnd (sigmask)	Заменить маску сигнала и приостановить процесс
s = kill (pid, sig)	Послать сигнал процессу
s = pause ( )	Приостановить выполнение процесса до след. сигнала


Создание процесса
Созданный
Памяти недостаточно
Памяти достаточно
Готовый к выполнению, выгруженный
Загрузка
Выгрузка
Готовый к выполнению, загруженный
Выполнение
Вытеснение
Выполняющийся
Пробуждение
Пробуждение
Засыпание
Освобождение
Спящий, выгруженный
Выгрузка
Спящий, загруженный
Зомби
Завершение

Каждый запущенный процесс в любой момент времени находится в одном из следующих состояний (которое называют еще статусом процесса):
Активен (R=Running) – процесс находится в очереди на выполнение, то есть либо выполняется в данный момент, либо ожидает выделения ему очередного кванта времени центрального процессора.
«Спит» (S=Sleeping) – процесс находится в состоянии прерываемого ожидания, то есть ожидает какого-то события, сигнала или освобождения нужного ресурса.
Находится в состоянии непрерываемого ожидания (D=Direct) – процесс ожидает определенного («прямого») сигнала от аппаратной части и не реагирует на другие сигналы;
Приостановлен (T) – процесс находится в режиме трассировки (обычно такое состояние возникает при отладке программ).
«Зомби» (Z=Zombie) – это процесс, выполнение которого завершилось, но относящиеся к нему структуры ядра по каким-то причинам не освобождены. Одной из причин их появления в системе может быть следующая ситуация. Обычно освобождение структур ядра, относящихся к процессу, выполняет процесс-родитель после получения от потомка сигнала о завершении. Но бывают случаи, когда родительский процесс завершается раньше дочернего. Процессы, не имеющие родителя, называются "сиротами". "Сироты" автоматически усыновляются процессом init, который и принимает сигналы об их завершении. Если процесс-родитель или init по каким-то причинам не может принять сигнал о завершении дочернего процесса, то процесс-потомок превращается в "зомби" и получает статус Z. Процессы-зомби не занимают процессорного времени (т. е. их выполнение прекращается), но соответствующие им структуры ядра не освобождаются. В некотором смысле это «мертвые» процессы. Уничтожение таких процессов — одна из обязанностей системного администратора. (появление данных процессов говорит о том, что в системе что-то не в порядке, и скорее всего не в порядке с аппаратной частью).
Особый вид процессов - демоны. Данный вид процессов работает в фоне (подобно службам в Windows), без терминала и выполняет задачи для других процессов. Данный вид процессов на серверных системах является основным.

Реализация процессов в системе Unix

Ядро поддерживает две ключевые структуры данных, относящиеся к процессам: **таблицу процессов** – *proc* -(резидентная) и **структуру пользователя** - *user* -(выгружается на диск, когда процесс отсутствует в памяти).

Таблица процессов содержит:

1. **Параметры планирования**. Приоритеты процессов, процессорное время, потребленное за последний учитываемый период, количество времени, проведенное процессом в режиме ожидания.
2. **Образ памяти**. Указатели на сегменты программы, данных и стека или на таблицы страниц. Когда процесса нет в памяти здесь содержится информация о его месте на диске.
3. **Сигналы.** Маски, характеризующие сигналы (игнорирование, перехват, блокирование)
4. **Разное.** Текущее состояние процесса, ожидаемые события, PID процесса, идентификатор пользователя и др.

Структура пользователя включает:

1. **Машинные регистры.**
2. **Информацию о текущем системном вызове (параметры и результаты).**
3. **Таблицу дескрипторов файлов.**
4. **Учетную информацию. Данные о процессорном времени, использованном в пользовательском и системном режимах.**
5. **Стек ядра для использования процессом в режиме ядра.**

Планирование в системе UNIX

1. **Низкоуровневый алгоритм** выбирает процесс, готовый к работе из очереди, имеющей высший приоритет (у процессов ядра – отрицательный, у процессов пользователя – положительный). Время кванта – 100 мс.

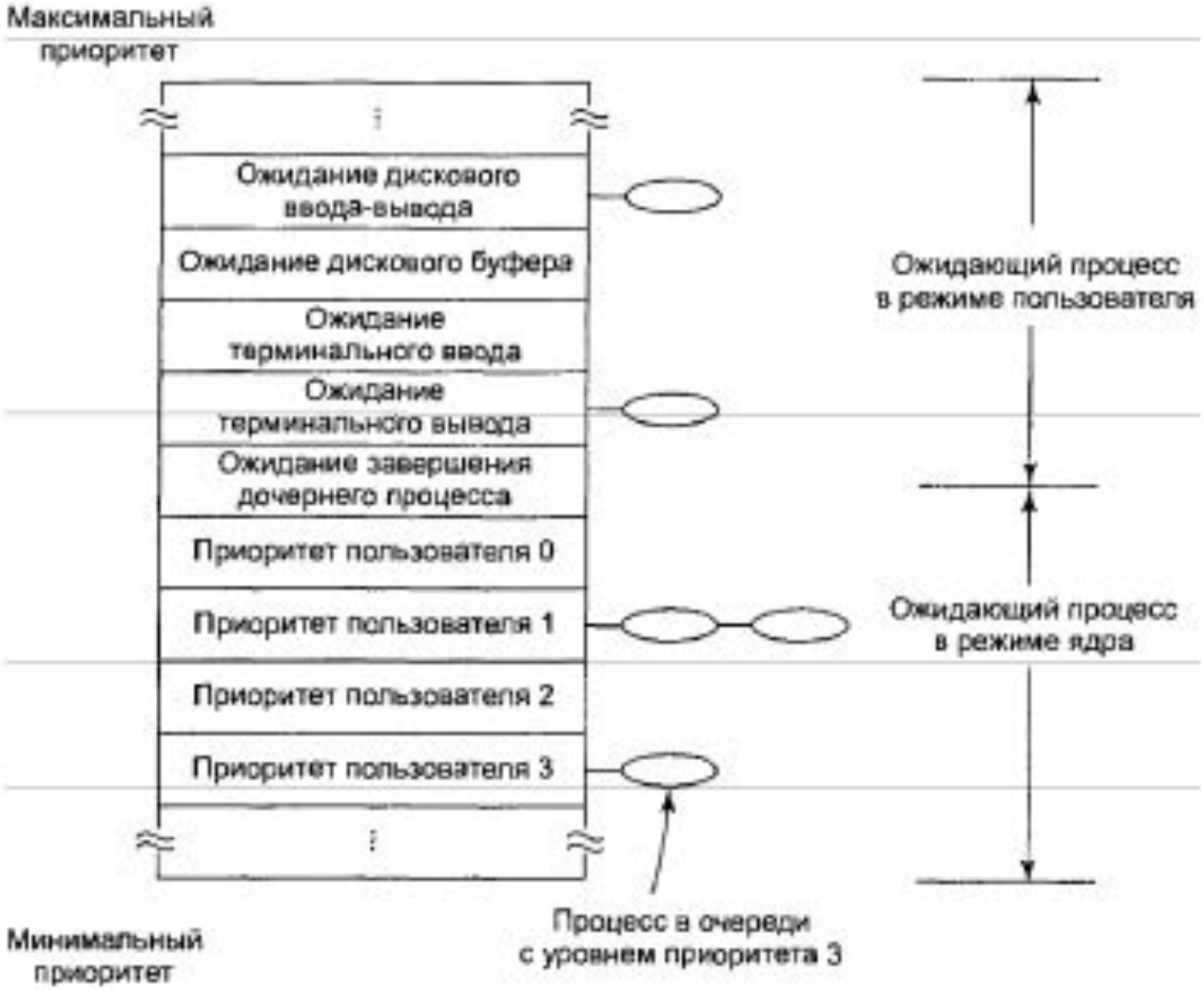
Priority = CPU_usage + nice + base

CPU_usage - “тики” таймера, при которых работал процесс;
CPU_usage (i) = CPU_usage (i – 1) / 2;
nice = от – 20 до + 20 (по умолчанию = 0);
base –назначается ОС (прошиты жестко и отрицательны для свопинга, дискового ввода-вывода и др.)

Priority пересчитывается каждую секунду.

На основе сосчитанного нового приоритета каждый процесс прикрепляется к соответствующей очереди . Для получения номера очереди приоритет, как правило, делится на некую константу.

2. **Высокоуровневый алгоритм** перемещает процессы из памяти на диск и обратно.


Максимальный приоритет
Ожидание дискового ввода-вывода
Ожидание дискового буфера
Ожидание терминального ввода
Ожидание терминального вывода
Ожидание завершения дочернего процесса
Приоритет пользователя 0
Приоритет пользователя 1
Приоритет пользователя 2
Приоритет пользователя 3
Ожидающий процесс в режиме пользователя
Ожидающий процесс в режиме ядра
Процесс в очереди с уровнем приоритета 3
Минимальный приоритет

Команды управления процессами

- Существует ряд команд, позволяющих просматривать и управлять процессами в системе:
  - ps – выводит информацию о выполняющихся процессах;
  - top – выводит и динамически обновляет список наиболее активных процессов;
  - nice – явно устанавливает приоритет процесса;
  - renice – корректирует приоритет процесса;
  - kill – завершение работы заданного процесса;
  - killall – завершение работы всех процессов, соответствующих заданному имени.

Средства администрирования

- Для управления операционной системой в UNIX часто используются конфигурационные файлы. Такие файлы определяют параметры запуска многих системных процессов.
- Для размещения конфигурационных файлов, как правило, используется каталог **/etc**.
  - adduser.conf – определяет параметры пользователя
  - crontab – задает таблицу расписаний
  - fstab – определяет таблицу разделов
  - ftpusers – определяет параметры пользователей ftp
  - hosts – определяет список соответствий имен и ip-адресов
  - hosts.allow – определяет список разрешенных хостов
  - rc.conf – определяет конфигурацию сетевых подключений
  - и др.
- Многие файлы представляют собой сценарии, обрабатываемые оболочками.

Учетные записи пользователей

- Для упорядочивания работы с пользователями, хранения информации о их персональных настройках используются **учетные записи** пользователей.
- **Группа пользователей** – именованное объединение нескольких учетных записей. Группа может быть использована для разграничения доступа к данным.

Имена групп и пользователей в текстовом виде • пользователей. используются для удобства самих Система вместо имени использует :идентификаторы

  - UID – идентификатор пользователя;
  - GID – идентификатор группы.

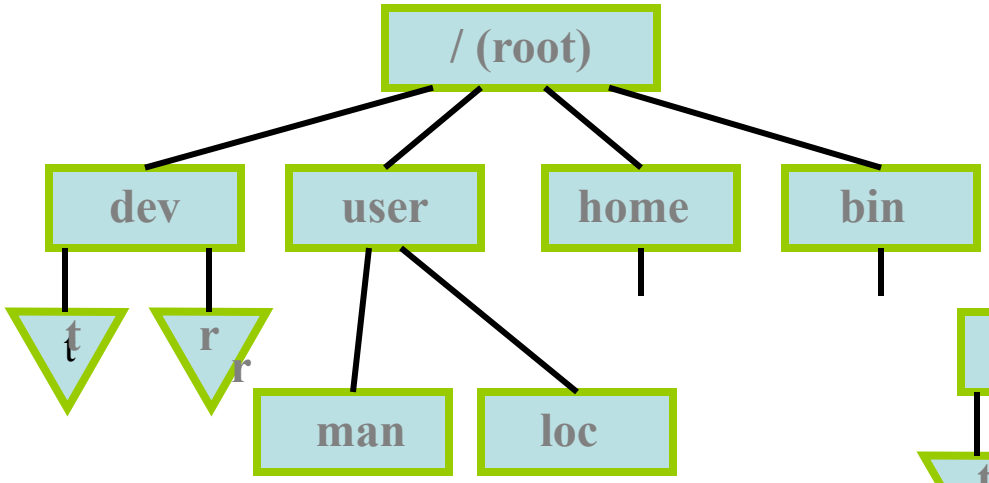
Хранение информации об учетных записях

- Информация об учетных записях хранится в нескольких структурах данных:
  - /etc/passwd – файл, содержащий основную информацию обо всех учетных записях:
    - logname:*:UID:GID:GECOS:HOME:SHELL
    - ivlev:*:1038:1038:ivlev:/home/ivlev:/usr/local/bin/bash
  - /etc/master.passwd – файл хранящий информацию из /etc/passwd и, кроме того, хэшированные значения паролей и ряд других сведений;
  - /etc/pwd.db /etc/spwd.db –специальные файлы баз данных для хранения информации подобно /etc/passwd
  - начальный каталог – содержит полное имя каталога для хранения пользовательских данных

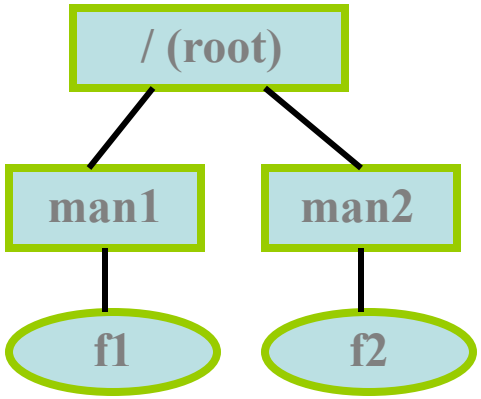
Монтирование файловой системы

- Доступ к разделу на носителе информации обеспечивается **монтированием** раздела в общую файловую систему.
- Монтирование обозначается определением файла устройства и точкой монтирования.
- При старте системы программа **mount** запускается стартовым скриптом и автоматически монтирует системы указанные в файле /etc/fstab.
- Команда монтирования раздела:
  - mount файл_устройства точка_монтирования

Монтирование



Файловая система 1

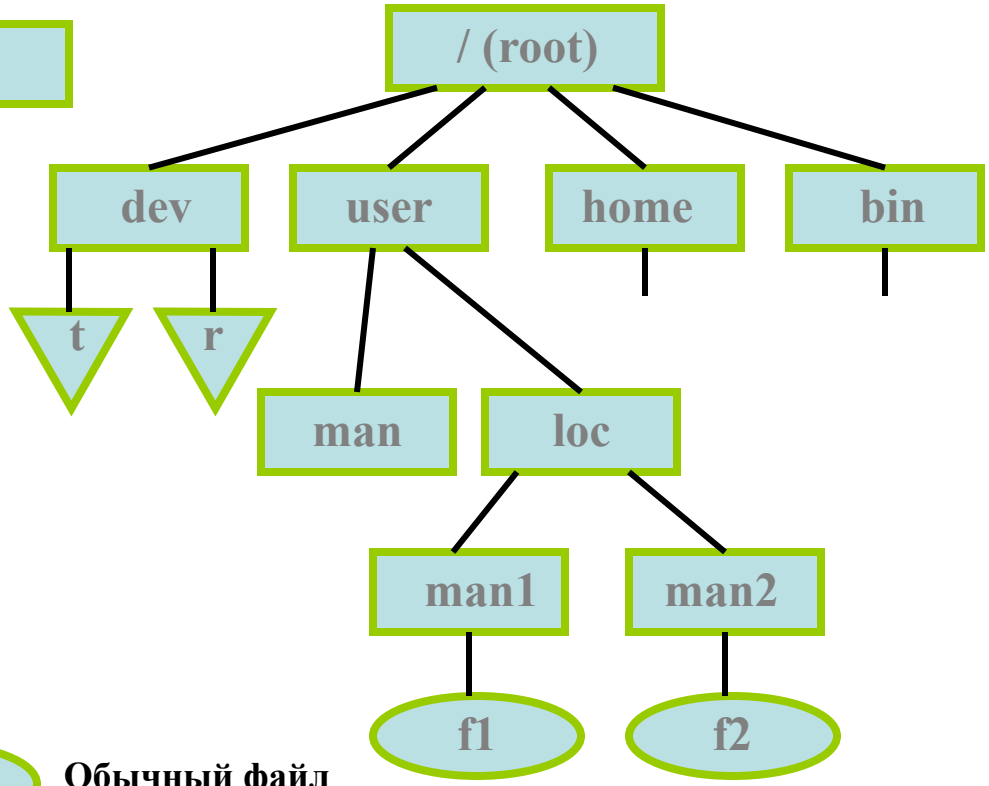


Файловая система 2

Обычный файл

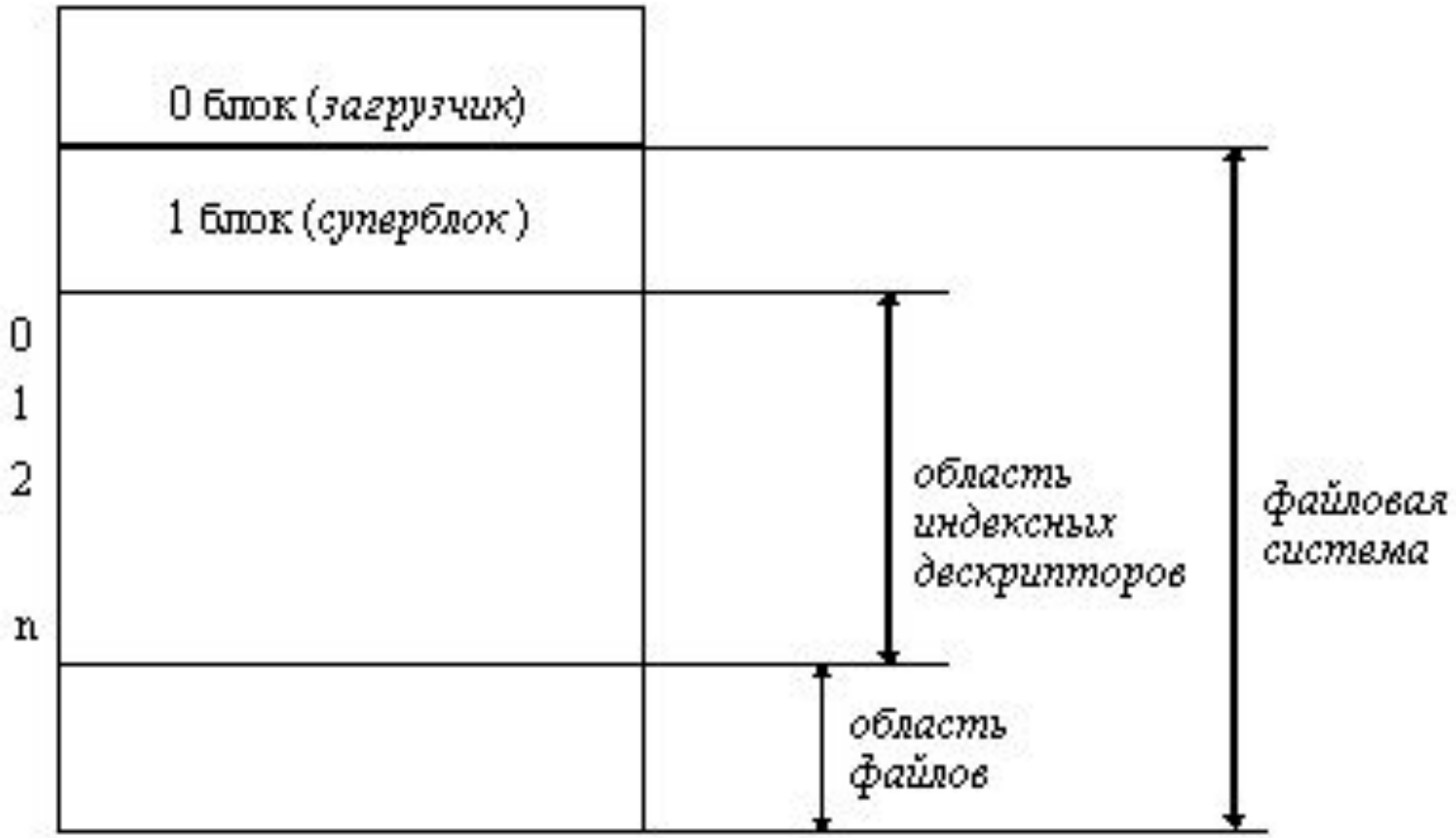
Каталог

Специальный файл-устройство



Общая файловая система после монтирования

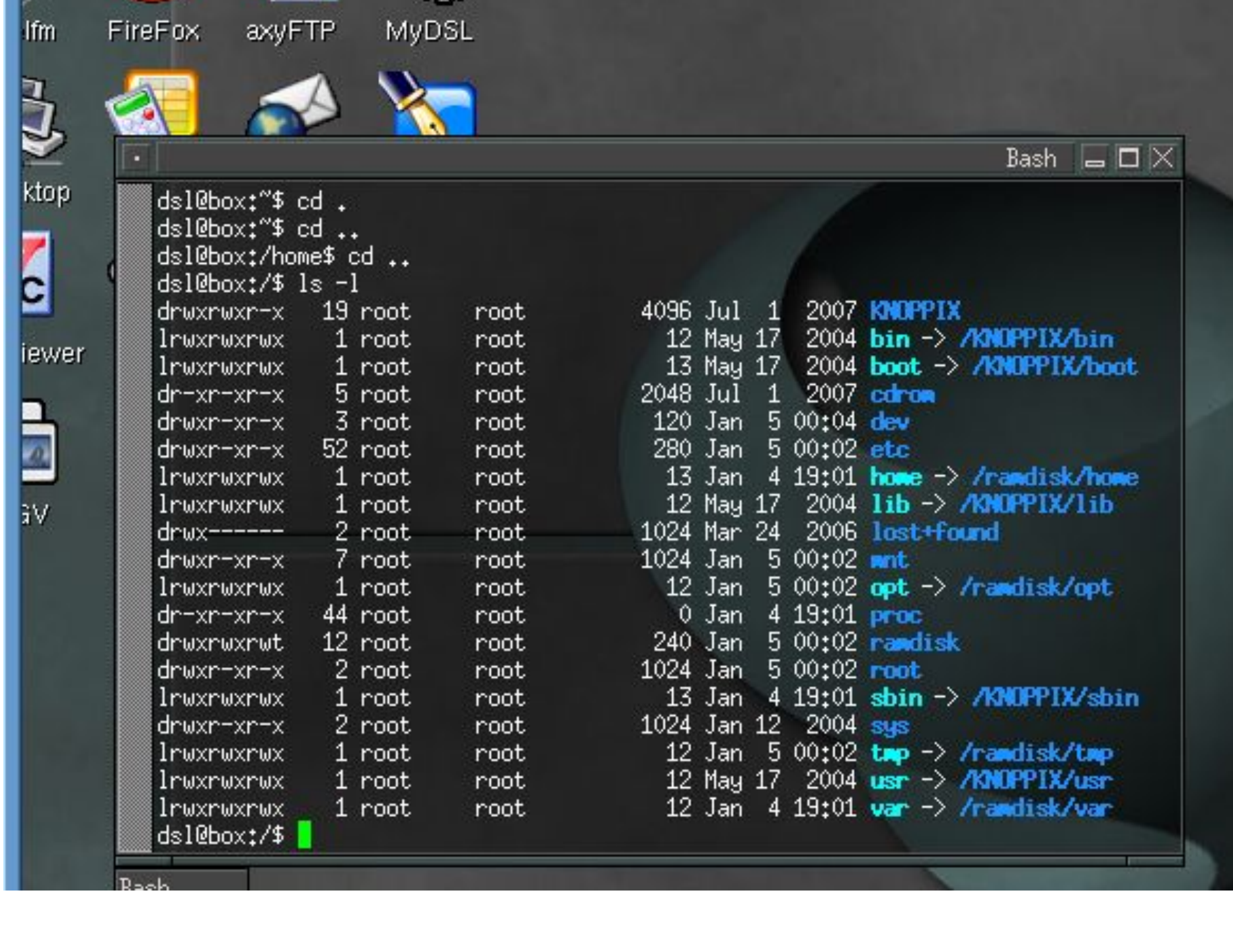
Физическая организация S5 и ufs



Расположение файловой системы s5 на диске

Все дисковое пространство, отведенное под файловую систему, делится на четыре области:

1. *загрузочный блок (boot),* в котором хранится загрузчик ОС;
2. *суперблок (superblock)* - содержит самую общую информацию о файловой системе: размер файловой системы, размер области индексных дескрипторов, число индексных дескрипторов, список свободных блоков и список свободных индексных дескрипторов, а также другую административную информацию;
3. *область индексных дескрипторов*, порядок расположения индексных дескрипторов в которой соответствует их номерам;
4. *область данных*, в которой расположены как обычные файлы, так и файлы-каталоги. Специальные файлы представлены в файловой системе только записями в соответствующих каталогах и индексными дескрипторами специального формата, но места в области данных не занимают.


lfm
FireFox
axyFTP
MyDSL
ktop
iewer
gV
Bash
dsl@box:~$ cd .
dsl@box:~$ cd ..
dsl@box:/home$ cd ..
dsl@box:/$ ls -l
drwxrwxr-x 19 root root 4096 Jul 1 2007 KNOPPIX
lrwxrwxrwx 1 root root 12 May 17 2004 bin -> /KNOPPIX/bin
lrwxrwxrwx 1 root root 13 May 17 2004 boot -> /KNOPPIX/boot
dr-xr-xr-x 5 root root 2048 Jul 1 2007 cdrom
drwxr-xr-x 3 root root 120 Jan 5 00:04 dev
drwxr-xr-x 52 root root 280 Jan 5 00:02 etc
lrwxrwxrwx 1 root root 13 Jan 4 19:01 home -> /ramdisk/home
lrwxrwxrwx 1 root root 12 May 17 2004 lib -> /KNOPPIX/lib
drwx------ 2 root root 1024 Mar 24 2006 lost+found
drwxr-xr-x 7 root root 1024 Jan 5 00:02 mnt
lrwxrwxrwx 1 root root 12 Jan 5 00:02 opt -> /ramdisk/opt
dr-xr-xr-x 44 root root 0 Jan 4 19:01 proc
drwxrwxrwt 12 root root 240 Jan 5 00:02 ramdisk
drwxr-xr-x 2 root root 1024 Jan 5 00:02 root
lrwxrwxrwx 1 root root 13 Jan 4 19:01 sbin -> /KNOPPIX/sbin
drwxr-xr-x 2 root root 1024 Jan 12 2004 sys
lrwxrwxrwx 1 root root 12 Jan 5 00:02 tmp -> /ramdisk/tmp
lrwxrwxrwx 1 root root 12 May 17 2004 usr -> /KNOPPIX/usr
lrwxrwxrwx 1 root root 12 Jan 4 19:01 var -> /ramdisk/var
dsl@box:/$

Структура *индексного дескриптора* (i-node)

1. идентификатор владельца и группы владельца файла;
2. тип файла:

   - (дефис) — обычный файл;

   d — каталог;

   c — символьное устройство;

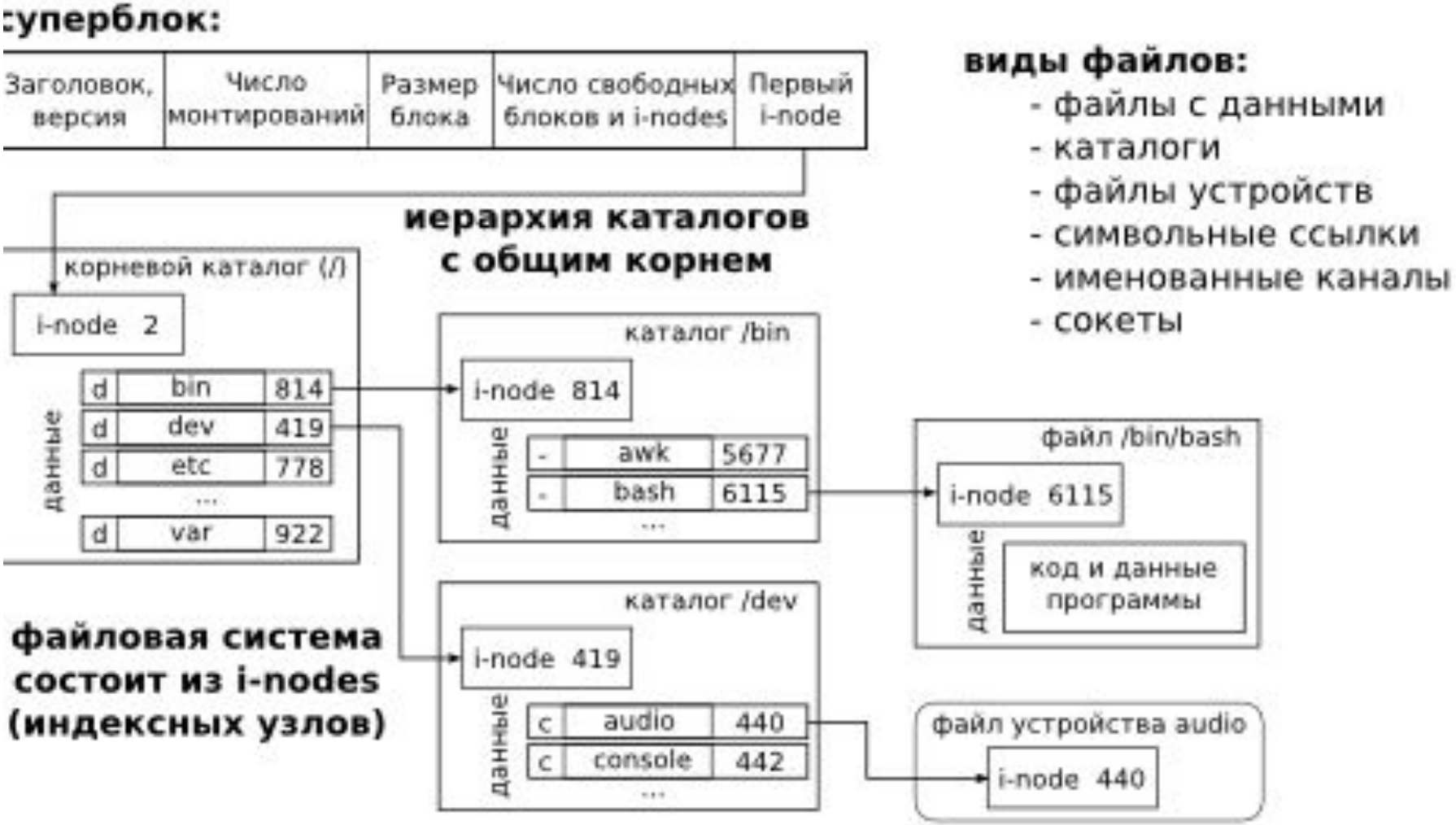
   b — блочное устройство;

   p — именованный канал / конвейер (named pipe);

   s — сокет (socket);

   l — символическая ссылка.
3. права доступа к файлу;
4. временные характеристики: время последней модификации файла, время последнего обращения к файлу, время последней модификации индексного дескриптора;
5. число ссылок на данный индексный дескриптор:
   - для обычных файлов равно количеству псевдонимов файла;
   - для каталогов равно количеству входящих в него файлов +2.
6. адресная информация (перечень номеров блоков);
7. размер файла в байтах.

Файловая система UNIX



Копирование индексного дескриптора входит в процедуру открытия файла. При открытии файла ядро выполняет следующие действия:

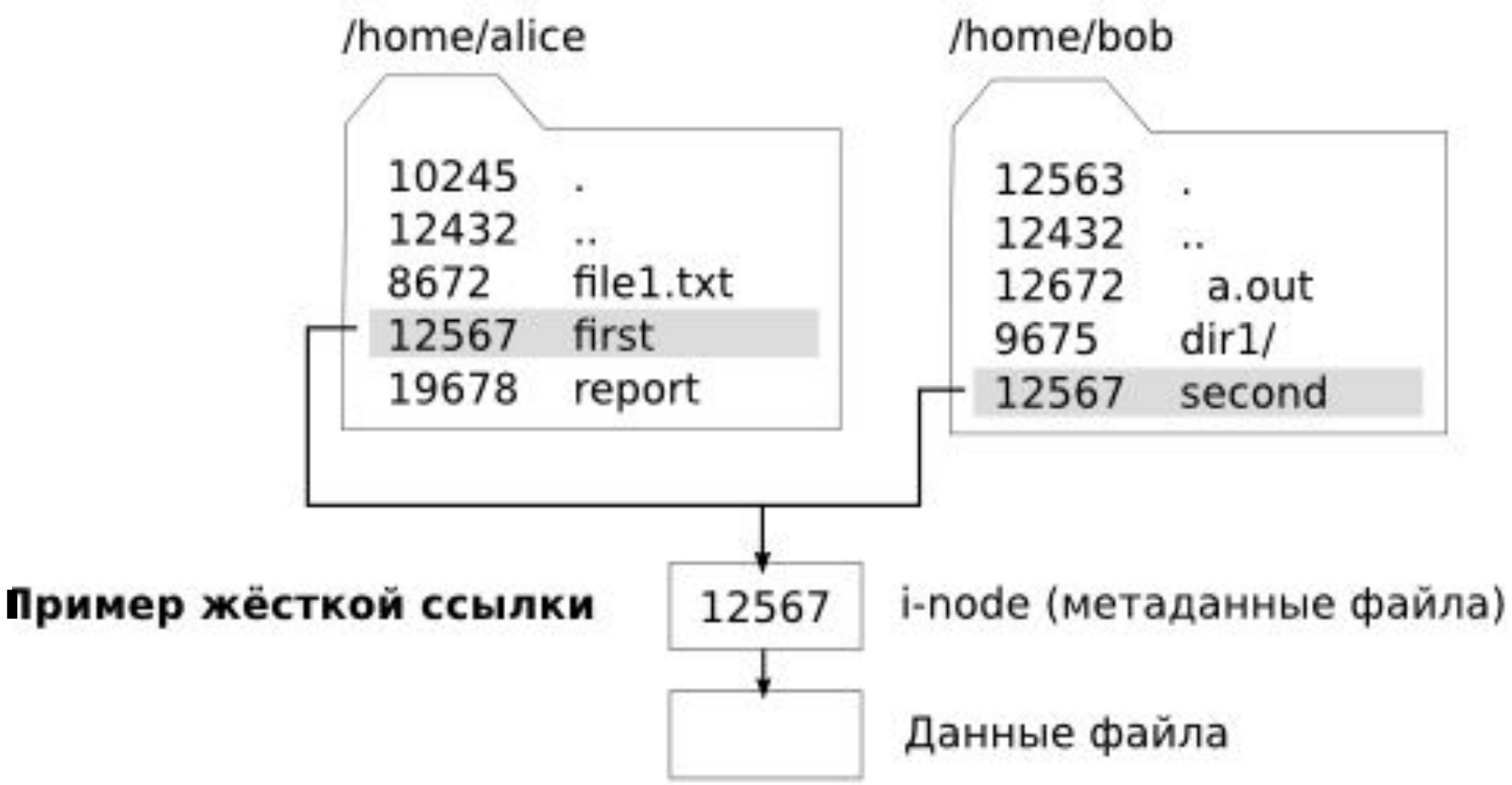
- Проверяет, существует ли файл; если не существует, то можно ли его создать. Если существует, то разрешен ли к нему доступ требуемого вида.
- Копирует индексный дескриптор с диска в оперативную память; если с указанным файлом уже ведется работа, то новая копия индексного дескриптора не создается.
- Создает в области ядра структуру, предназначенную для отображения текущего состояния операции обмена данными с указанным файлом. Эта структура, называемая file, содержит данные о типе операции (чтение, запись или чтение и запись), о числе считанных или записанных байтов, указатель на байт файла, с которым проводится операция.
- Делает отметку в контексте процесса, выдавшего системный вызов на операцию с данным файлом.

Стандарт POSIX (**Portable Operating System Interface**) требует реализовать поддержку двух типов связей - жестких и символических.

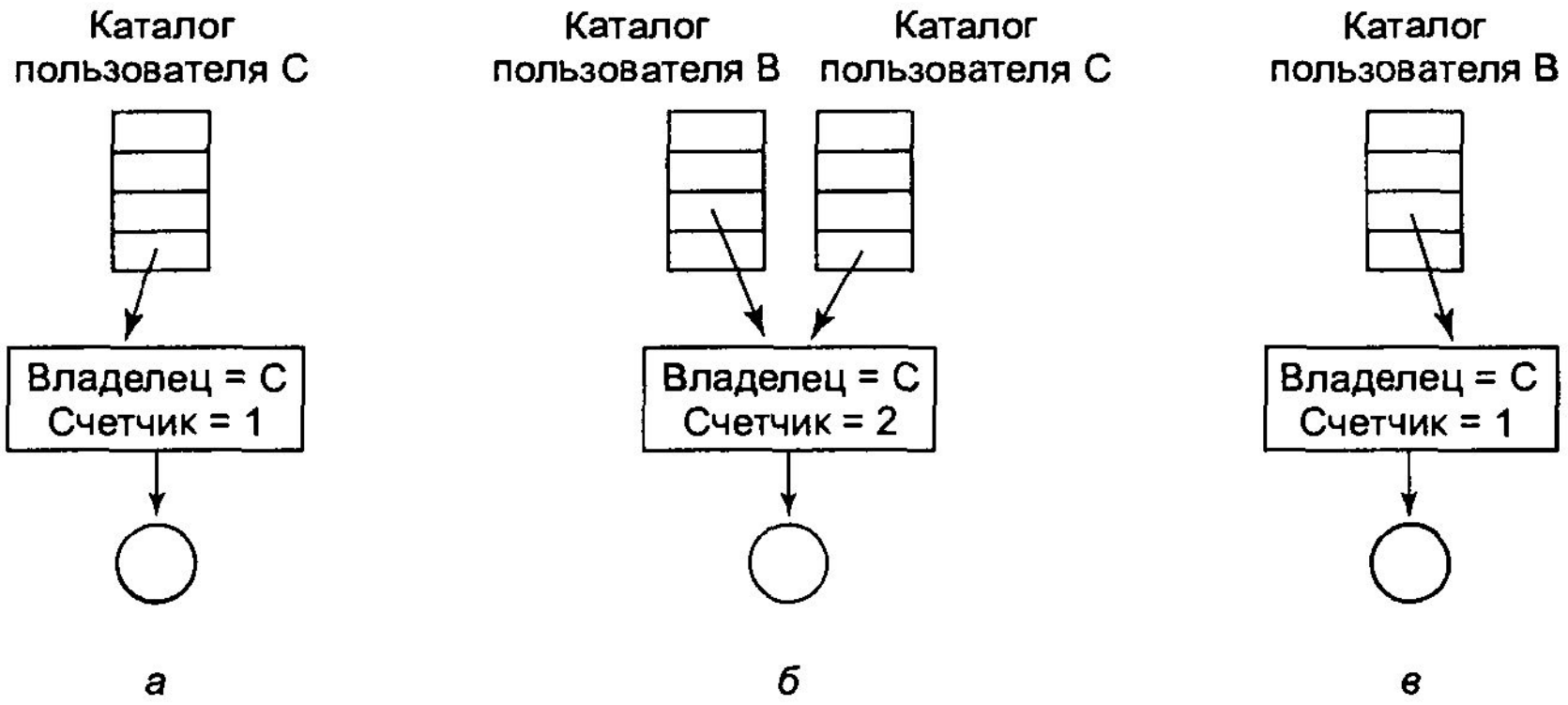
Жесткой связью (hard link) считается элемент каталога, указывающий непосредственно на некоторый *индексный дескриптор*. Жесткие связи очень эффективны, но они могут создаваться только в пределах одной физической файловой системы. Когда создается такая связь, связываемый файл должен уже существовать. Каталоги не могут связываться жесткой связью.

Символическая связь (symbolic link) - это специальный файл, который содержит путь к другому файлу. Указание на то, что данный элемент каталога является символической связью, находится в индексном дескрипторе. Поэтому обычные команды доступа к файлу вместо получения данных из физического файла, берут их из файла, имя которого приведено в связи. Этот путь может указывать на что угодно: это может быть каталог, он может даже находиться в другой физической файловой системе, более того, указанного файла может и вовсе не быть.

Имя файла является указателем на индексный дескриптор (i-node), который содержит атрибуты файла и массив адресов дисковых блоков, в которых находятся данные файла. Однако файл может иметь несколько имен. Дескриптор содержит счетчик числа этих имен. Создание жесткой связи - это создание еще одного имени, ссылающегося на тот же самый индексный дескриптор.



Пример жёсткой ссылки

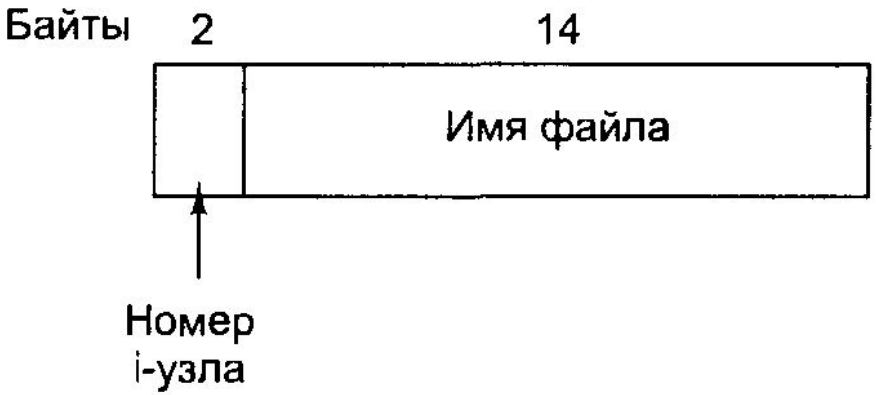


Ситуация до связывания (а); после создания связи (*б*); владелец удалил свой файл (*в*)

Удаление файла представляет собой уменьшение на 1 счетчика его имен в индексном дескрипторе. Физически файл удаляется системой, если он закрыт и если счетчик имен равен нулю. Его i-node объявляется свободным.

Символическая ссылка имеет ряд преимуществ по сравнению с жёсткой ссылкой: она может использоваться для связи файлов в разных файловых системах (ведь номера индексных узлов уникальны только в рамках одной файловой системы), а также более прозрачно удаление файлов – ссылка может удаляться совершенно независимо от основного файла.

Структура каталога



Корневой каталог/

bin	6
sys	3
etc	11

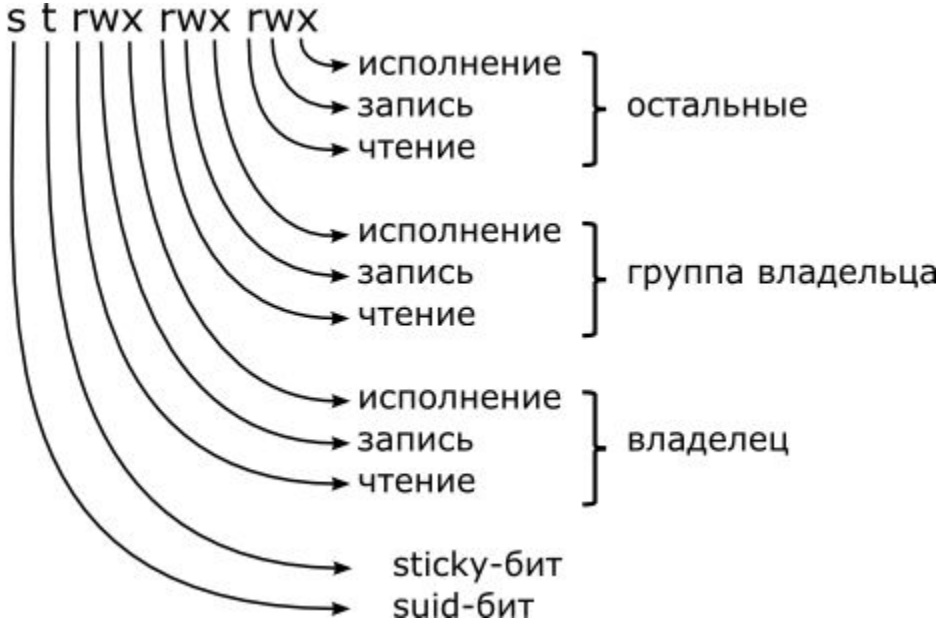
Для каталога число связей - число его подкаталогов. Текущий каталог обозначается точкой (.); родительский каталог, которому принадлежит текущий, обозначается двумя точками (..)

Права доступа к файлу

- Для управления доступом к файлу используются специальные атрибуты, определяющие права доступа – биты доступа.
- Класс доступа задается числовым идентификатором, определяющимся следующим образом:
  - Для каждой категории задается трехзначное двоичное число:
    - старший разряд определяет право на чтение;
    - второй разряд определяет право на запись;
    - младший разряд – на выполнение данного файла.
  - Категории определяются следующим образом:
    - первая категория – владелец файла
    - вторая категория – группа владельца
    - третья категория –остальные пользователи

Владелец			**Группа**			**Остальные**			**Значение**
R	**W**	**X**	**R**	**W**	**X**	**R**	**W**	**X**	
1	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**750**

- В UNIX существует три основных права доступа: чтение, запись и исполнение. Право исполнения трактуется по-разному для разных типов файлов.
- Для простых файлов оно определяет возможность запуска содержащейся в нём программы.
- Для директории право исполнения означает возможность "войти" в неё.

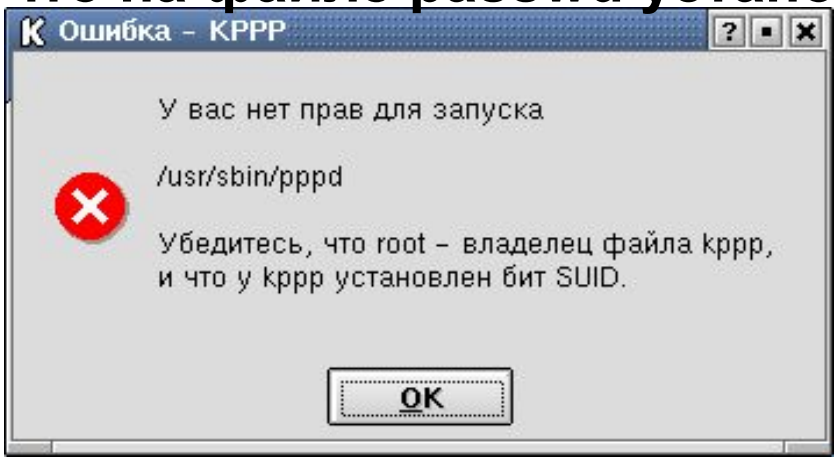


Sticky bit

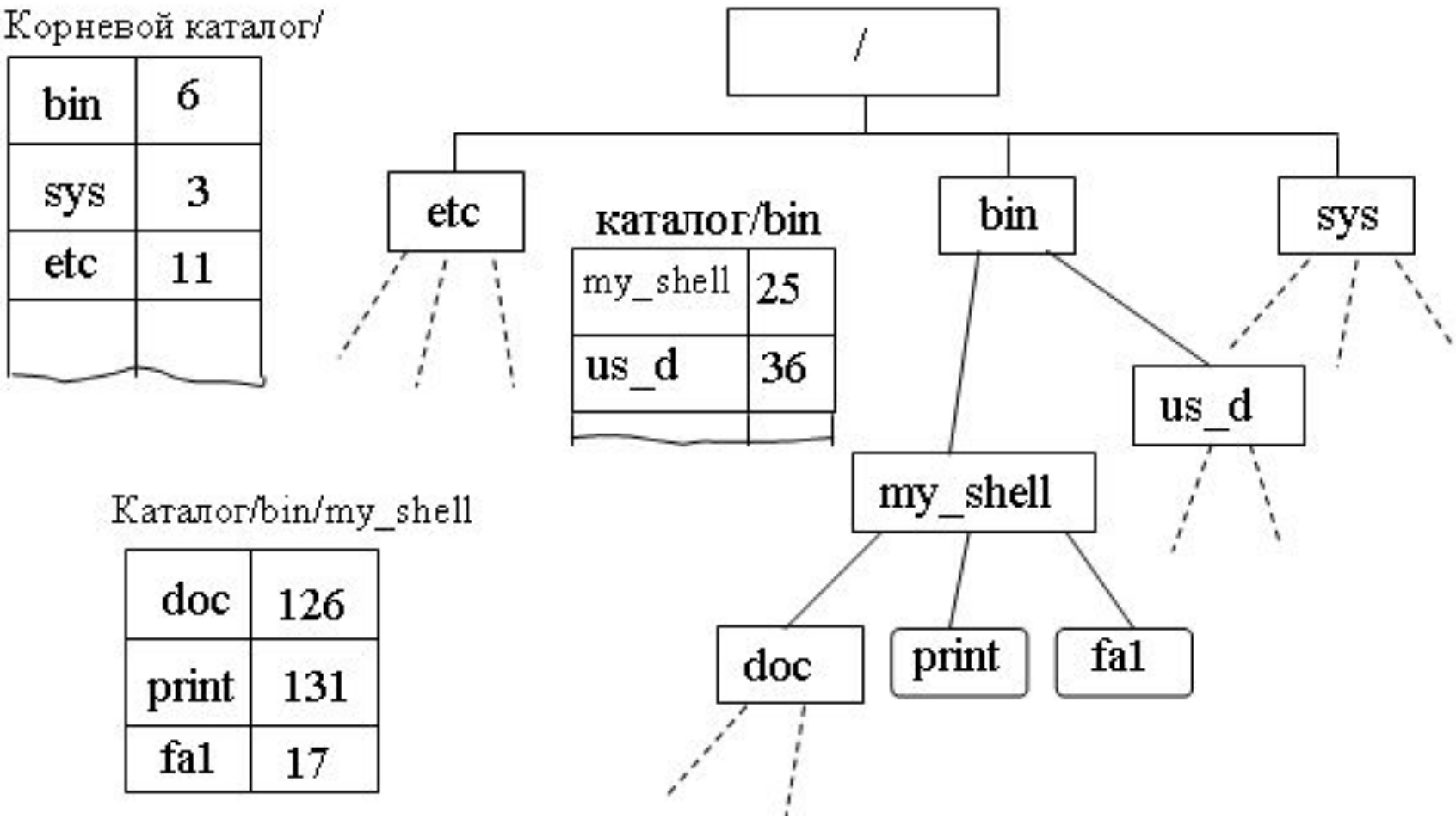
- Помимо комбинации из этих девяти прав доступа, каждый файл может иметь дополнительные флаги доступа: sticky-бит (t), специфичный для директорий, и suid-бит (s), применяемый для исполняемых файлов.
- Сегодня sticky bit используется в основном для директорий, чтобы защитить в них файлы. Из такой директории пользователь может удалить только те файлы, владельцем которых он является. Примером может служить директория /tmp, в которой запись открыта для всех пользователей, но нежелательно удаление чужих файлов.

Suid-бит

- Программа с установленным битом suid является «потенциально опасной».
- Если установлены права доступа SUID и файл исполняемый, то при запуске на выполнение файл получает не права запустившего его, а права владельца файла.
  Например:
  В системе пользователям разрешено самостоятельно изменять пароль при помощи утилиты passwd, обычно находится /usr/bin/passwd. Владельцем passwd является пользователь root поскольку утилита работатет с файлом /etc/passwd который может модифицировать только пользователь root. Что бы непривилигированый пользователь мог работать с системным файлом (т.е. изменить свой пароль) на утилите passwd должен быть устанвлен бит SUID.
  -r-s--x--x 1 root root 15 Mar 2004 passwd s -(буква) в правах означает, что на файле passwd установлен бит SUID



Поиск файла */bin/ my_shell/print*



1. просматривается корневой каталог с целью поиска первого составляющего символьного имени – это *bin.* Определяется номер индексного дескриптора каталога – это *6*, адрес корневого каталога системе известен;
2. из области индексных дескрипторов считывается дескриптор №6, начальный адрес дескриптора определяется на основании известных системе номера начального сектора номера индексного дескриптора и размера индексного дескриптора. Из индексного дескриптора 6 определяется физический адрес каталога bin.
3. просматривается каталог bin, целью поиска имени my_shell и определяется его номер – это *25*;
4. считывается индексный дескриптор 25, определяется физический адрес каталога */bin/ my_shell/print*;
5. просматривая каталог /bin/ my_shell/print, определяется номер индексного дескриптора файла print – это 131;
6. из индексного дескриптора131 определяются номера блоков данных и другие характеристики искомого файла.

Атрибуты файлов

Аналогично файловой системе FAT, имеющей атрибуты файлов (архивный, системный файл, скрытый), в файловой системе UNIX также имеются свои собственные, но используются они очень редко и могут быть установлены только для каталогов и обычных файлов.

- Команда **lsattr** выводит список (LiSt) атрибутов
- Команда **chattr** изменяет (CHange) их.

Доступны следующие атрибуты:

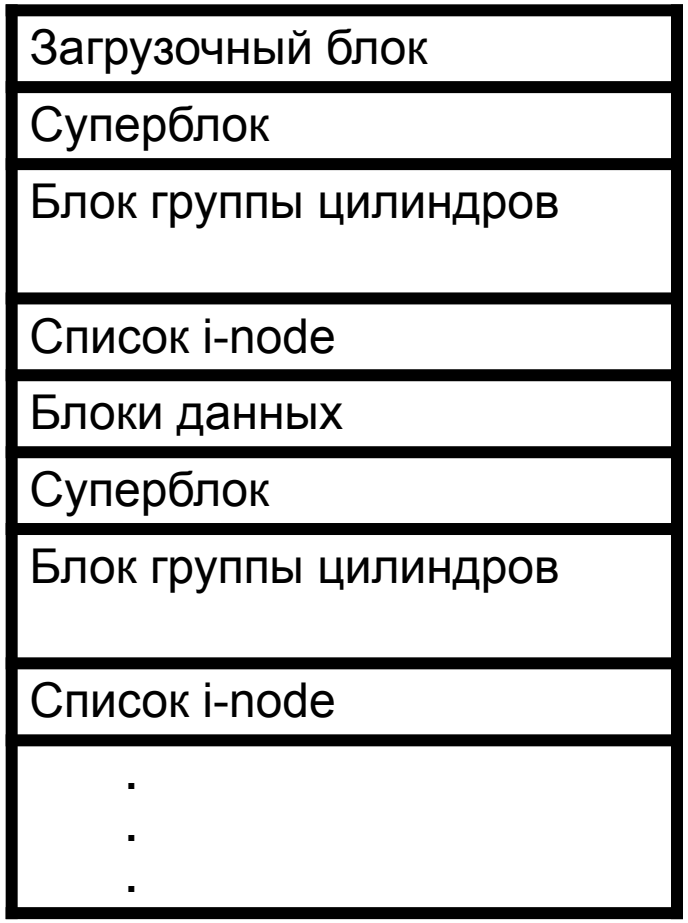
1. **A** («no *A*ccess time»): если для файла или каталога установлен этот атрибут, то, всякий раз при обращении к нему для чтения или записи, у него не будет обновляться время последнего доступа. Это может быть полезно, например, для файлов и каталогов, к которым очень часто обращаются для чтения, особенно из-за того, что это единственный параметр в inode, который изменяется при открытии файла только для чтения.
2. **a** («*a*ppend only»): если для файла установлен этот атрибут, и этот файл открыт для записи, то единственной доступной операцией будет добавление данных к его предыдущему содержимому. Для каталога это означает, что вы сможете только добавить файлы, но не сможете переименовать или удалить ни одного из существующих файлов. Только `root` может установить или снять этот атрибут.

3. `d` («no *d*ump»): **dump** - это стандартная утилита UNIX для резервного копирования. Она делает дамп любой файловой системы, для которой счетчик дампов в файле `/etc/fstab` (5-е поле) равен 1. Но если этот атрибут установлен для файла или каталога, то он, в отличие от других, будет пропущен при снятии дампа. Обратите внимание, что при установке его для каталогов, это также распространяется на все их подкаталоги и файлы.
4. `i` («*i*mmutable»): файл или каталог с установленным этим атрибутом вообще не может быть изменен: он не может быть переименован, на него не может быть создана ссылка и он не может быть удален. Только `root` может установить или снять этот атрибут. Обратите внимание, что это также предотвращает изменение времени последнего доступа, поэтому вам нет необходимости устанавливать атрибут `A`, если установлен `i`.
5. `s` («*s*ecure deletion»): когда удаляется файл или каталог с этим атрибутом, блоки, которые он занимал на диске перезаписываются нулями.
6. `S` («*S*ynchronous mode»): если для файла или каталога установлен этот атрибут, все его изменения синхронизируются и немедленно записываются на диск.

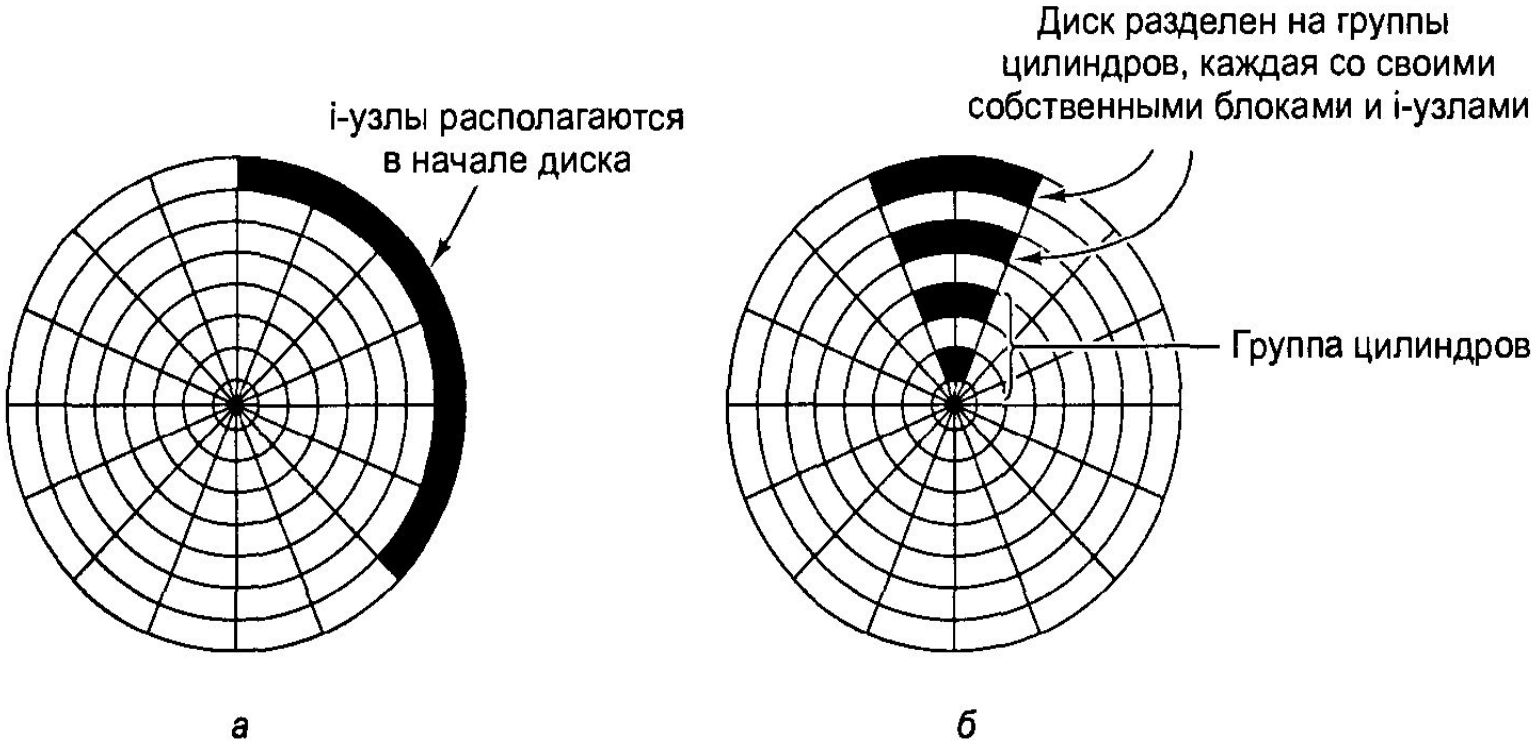
Например, вы можете установить атрибут `i` на жизненно важные системные файлы, чтобы избежать неприятных сюрпризов.

Физическая организация файловой системы

ufs



Таблицы **inodes** содержатся в блоке группы цилиндров, наряду с картами свободных/занятых **inodes** и карты свободных/занятых блоков данных). Это имеет целью размещение **inodes** и относящихся к ним блоков данных максимально близко друг к другу, что влечет за собой повышение производительности (за счет минимизации перемещений головок).

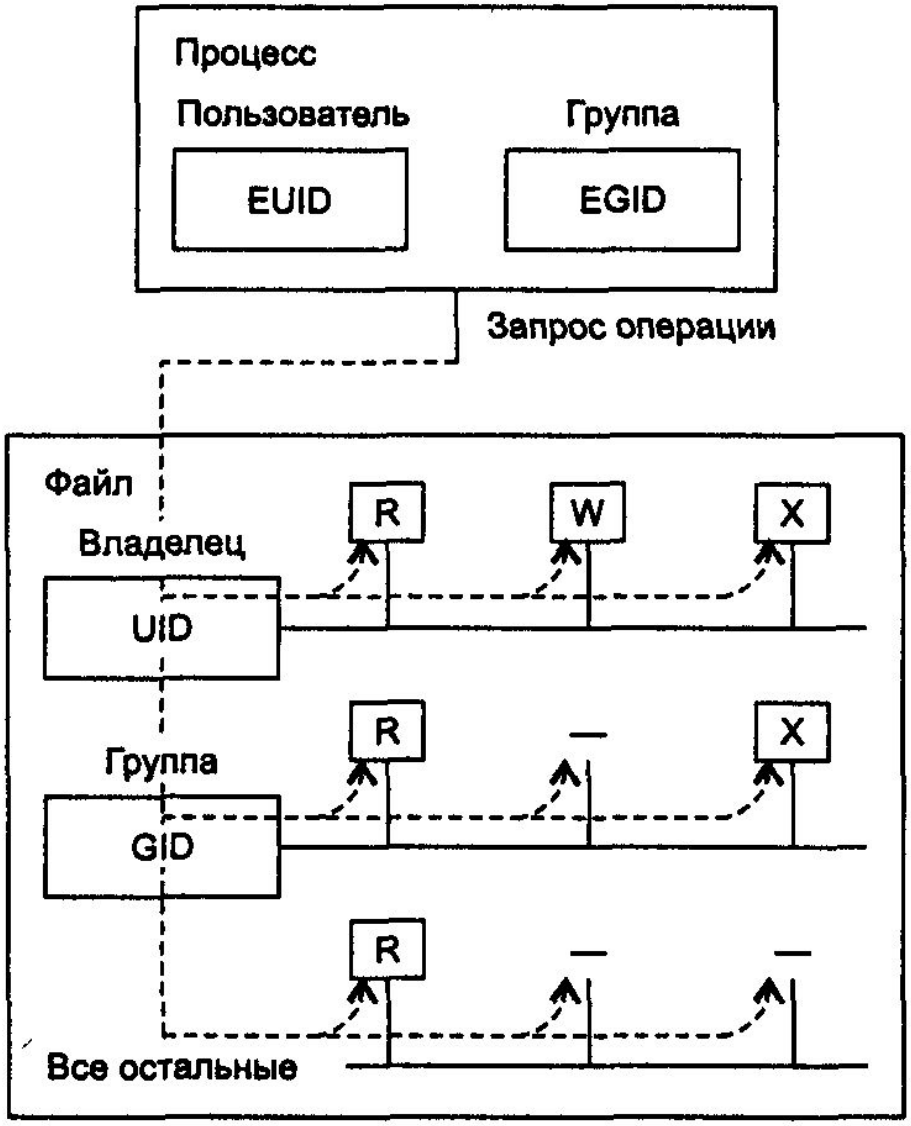


I-узлы, размещенные в начале диска (*а*); диск, разделенный на группы цилиндров, каждая со своими собственными блоками и i-узлами (*б*)

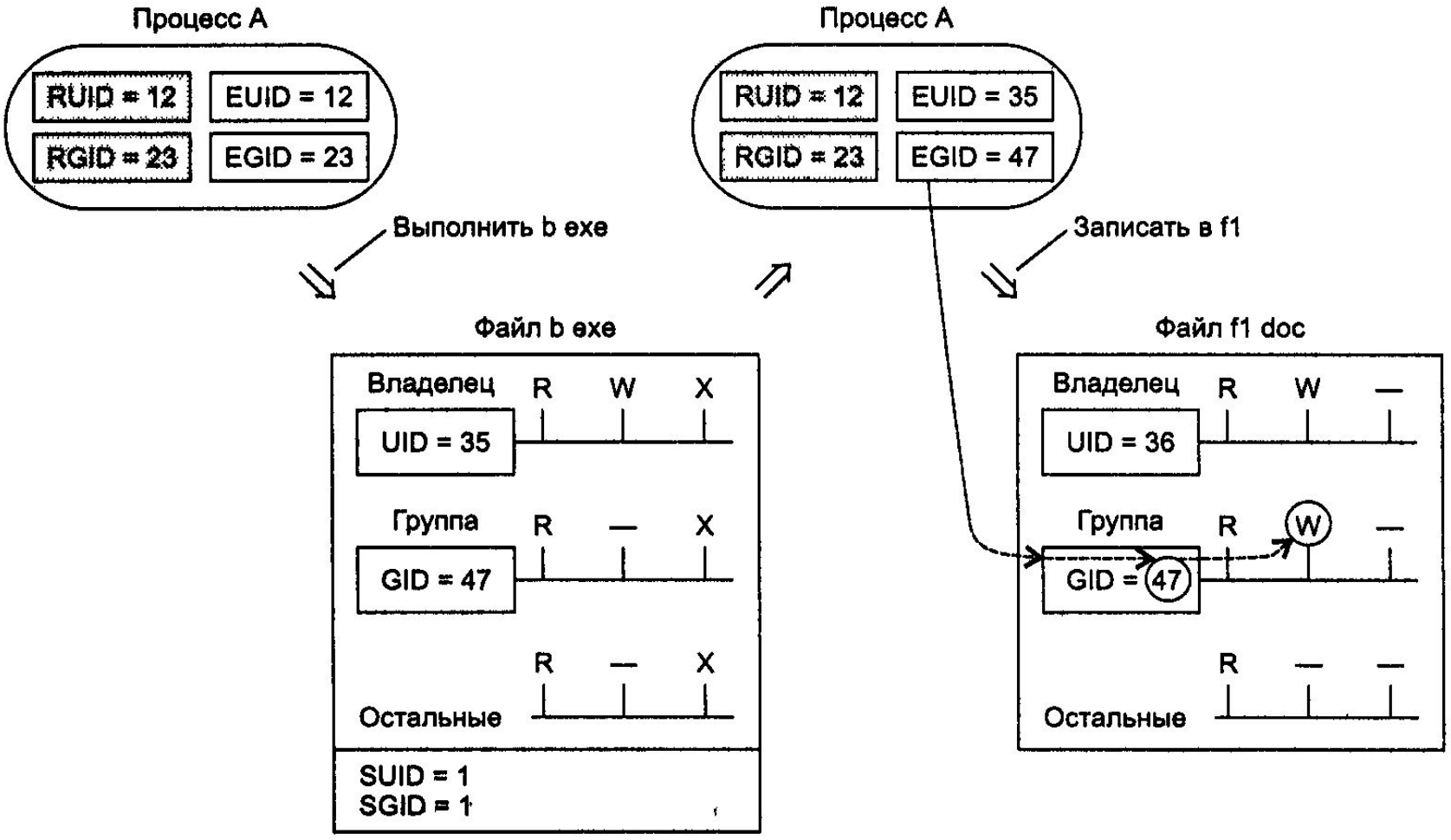
Средства управления доступом

- **С каждым процессом UNIX связаны два идентификатора: пользователя, от имени которого был создан этот процесс, и группы, к которой принадлежит данный пользователь. Эти идентификаторы носят название *реальных идентификаторов пользователя:* Real User ID, RUID и *реальных идентификаторов группы:* Real Group ID, RGID.**
- **При проверке прав доступа к файлу используются так называемые *эффективные идентификаторы пользователя:* Effective User ID, EUID и *эффективные идентификаторы группы:* Effective Group ID, EGID.**
- **Файл имеет два признака разрешения смены идентификатора — Set User ID on execution (SUID) и Set Group ID on execution (SGID), которые разрешают смену идентификаторов пользователя и группы при выполнении данного файла.**

Проверка прав доступа в UNIX



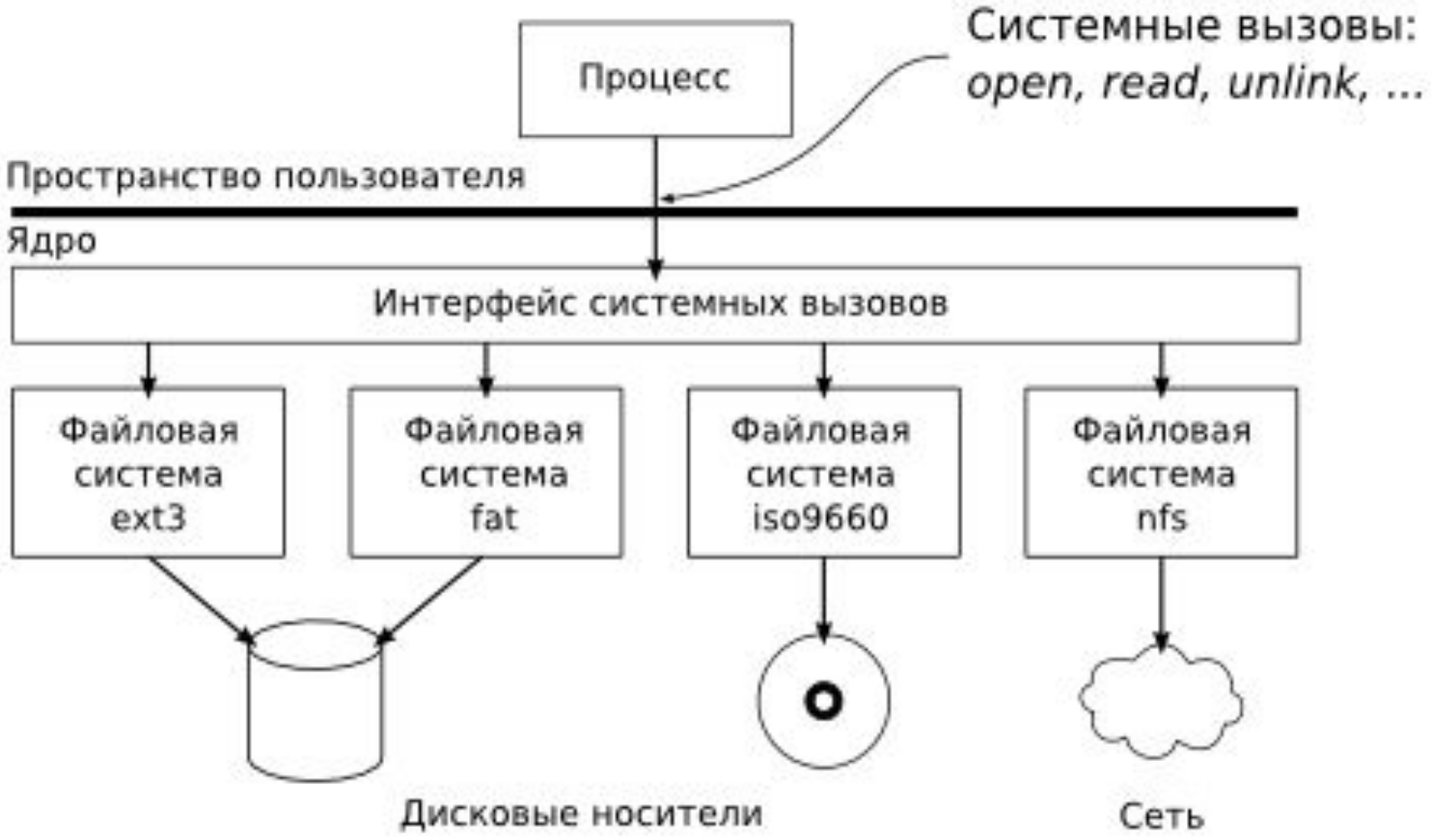
Смена эффективных идентификаторов процесса



Система ввода-вывода

- Основу системы ввода-вывода ОС UNIX составляют драйверы внешних устройств и средства буферизации данных. ОС UNIX использует два различных интерфейса с внешними устройствами: байт-ориентированный и блок-ориентированный.
- Любой запрос на ввод-вывод к блок-ориентированному устройству преобразуется в запрос к подсистеме буферизации, которая представляет собой буферный пул и комплекс программ управления этим пулом.
- Буферный пул состоит из буферов, находящихся в области ядра. Размер отдельного буфера равен размеру блока данных на диске.

Виртуальная файловая система




Виртуальная файловая система
Кэш
Файловая система 1
Файловая система 2
Обычный файл
Блочный специальный файл
Символьный специальный файл
Сетевой сокет
Планировщик ввода-вывода
Планировщик ввода-вывода
Дисциплина линии (не обязательна)
Драйверы протоколов
Драйвер блочного устройства
Драйвер блочного устройства
Драйвер символьного устройства
Драйвер сетевого устройства

Специальные файлы как универсальный интерфейс

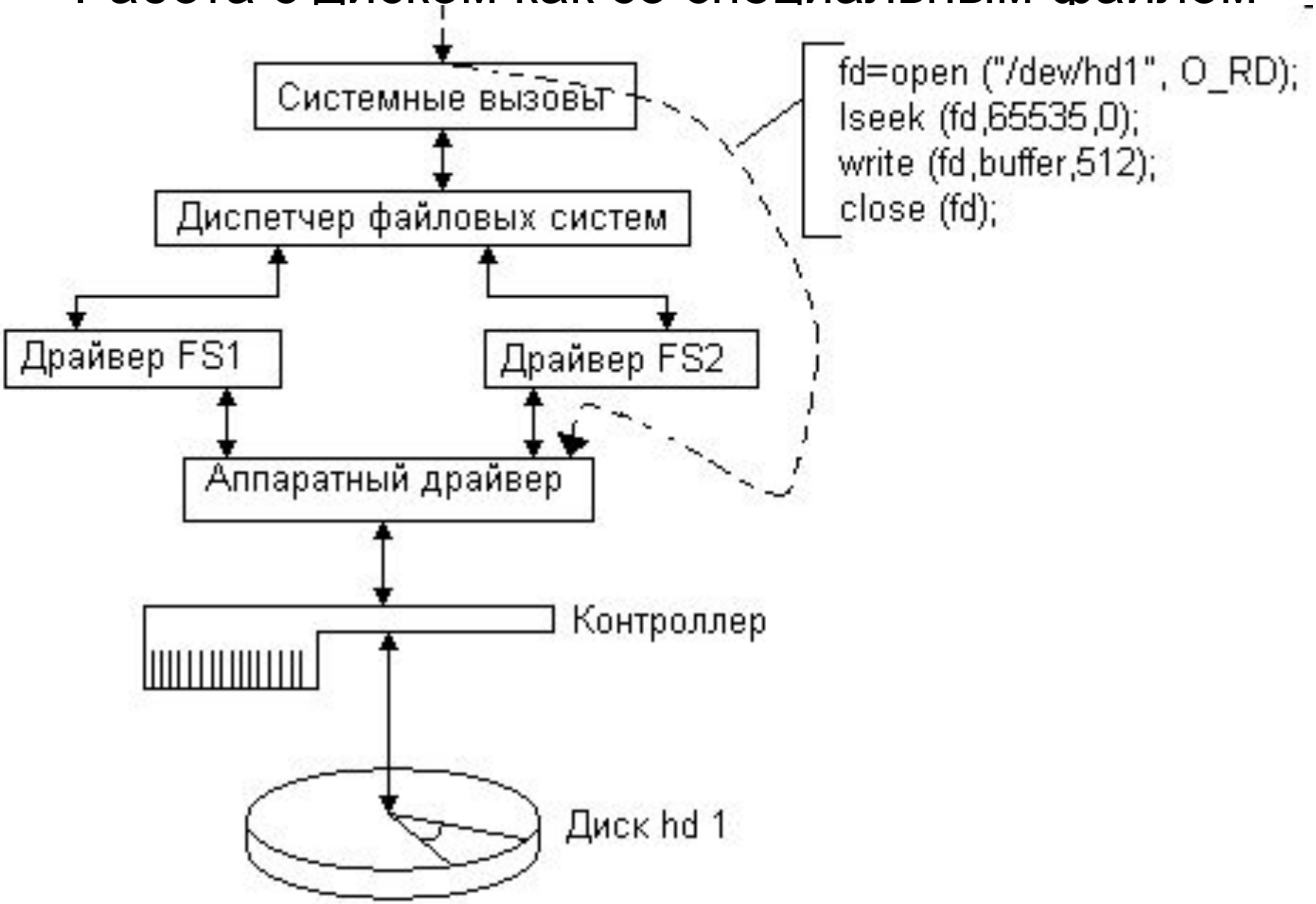
- Специальный файл (СФ), называемый также виртуальным файлом, связан с некоторым устройством ввода-вывода и представляет его для остальной части операционной системы и прикладных процессов в виде неструктурированного набора байт, то есть в виде обычного файла. Однако в отличие от обычного файла СФ не хранит статичные данные, а является интерфейсом к одному из аппаратных драйверов ОС.
- Со специальным файлом можно работать так же, как и с обычным, то есть открывать, считывать из него или же записывать в него определенное количество байт, а после завершения операции закрывать.

Основные специальные файлы	
/dev/console	Системная консоль, т. е. монитор и клавиатура, физически подключенные к компьютеру
/dev/hd	Жесткие диски с IDE-интерфейсом. Устройство /dev/hda1 соответствует первому разделу на первом жестком диске (/dev/hda), т. е. на диске, подключенном как Primary Master
/dev/sd	Жесткие диски с SCSI-интерфейсом
/dev/fd	Файлы дисководов для гибких дисков. Первому дисководу соответствует /dev/fd0, второму /dev/fd1
/dev/tty	Файлы поддержки пользовательских консолей. Название сохранилось с тех пор, когда к системе UNIX подключались телетайпы в качестве терминалов. В Unix эти файлы устройств обеспечивают работу виртуальных консолей.
/dev/pty	Файлы поддержки псевдо-терминалов. Применяются для удаленных рабочих сессий с использованием telnet
/dev/cua	Специальные устройства для работы с модемами
/dev/null	Это устройство - просто черная дыра. Все, что записывается в /dev/null, навсегда потеряно.

- Для этого используются системные вызовы для работы с обычными файлами: open, create, read, write и close. Кроме того, имеется несколько системных вызовов, используемых только при работе с СФ, например вызов loctl, с помощью которого можно передать команду контроллеру устройства.
- Для устройств прямого доступа имеет смысл также указатель текущего положения в файле, которым можно управлять с помощью системного вызова lseek.

- Файловый интерфейс, оперирующий только с неструктурированным потоком байт, оказывается полезным и для устройств со сложной организацией информации.
- Прикладной программист может воспользоваться им для создания собственного интерфейса к какому-либо устройству, обходя лежащие над аппаратным драйвером данного устройства слои высокоуровневых драйверов

Работа с диском как со специальным файлом



- В UNIX специальные файлы традиционно помещаются в каталог /dev, хотя ничто не мешает созданию их в любом каталоге файловой системы. При появлении нового устройства и соответственно нового драйвера администратор системы может создать новую запись.
- Связь специального файла с драйвером устанавливается за счет информации, находящейся в индексном дескрипторе специального файла.

1. В индексном дескрипторе хранится признак того, что файл является специальным, причем этот признак позволяет различить класс соответствующего устройству драйвера, то есть он определяет, является ли драйвер байт-ориентированным или блок-ориентированным.
2. В индексном дескрипторе хранится адресная информация, позволяющая выбрать нужный драйвер и нужное устройство. Эта информация заменяет стандартную адресную информацию обычного файла, состоящую из номеров блоков файла на диске.

$ ls -l /dev/sd*

brw-rw---T 1 root disk 8, 0 Окт 18 11:15 /dev/sda

brw-rw---T 1 root disk 8, 1 Окт 18 11:16 /dev/sda1

$ ls -l /dev/ | grep 1

crw-rw-rw- 1 root root 1, 7 Авг 31 16:04 full

crw-rw---- 1 root root 1, 11 Авг 31 16:04 kmsg

brw-rw---- 1 root disk 1, 1 Авг 31 16:04 ram1

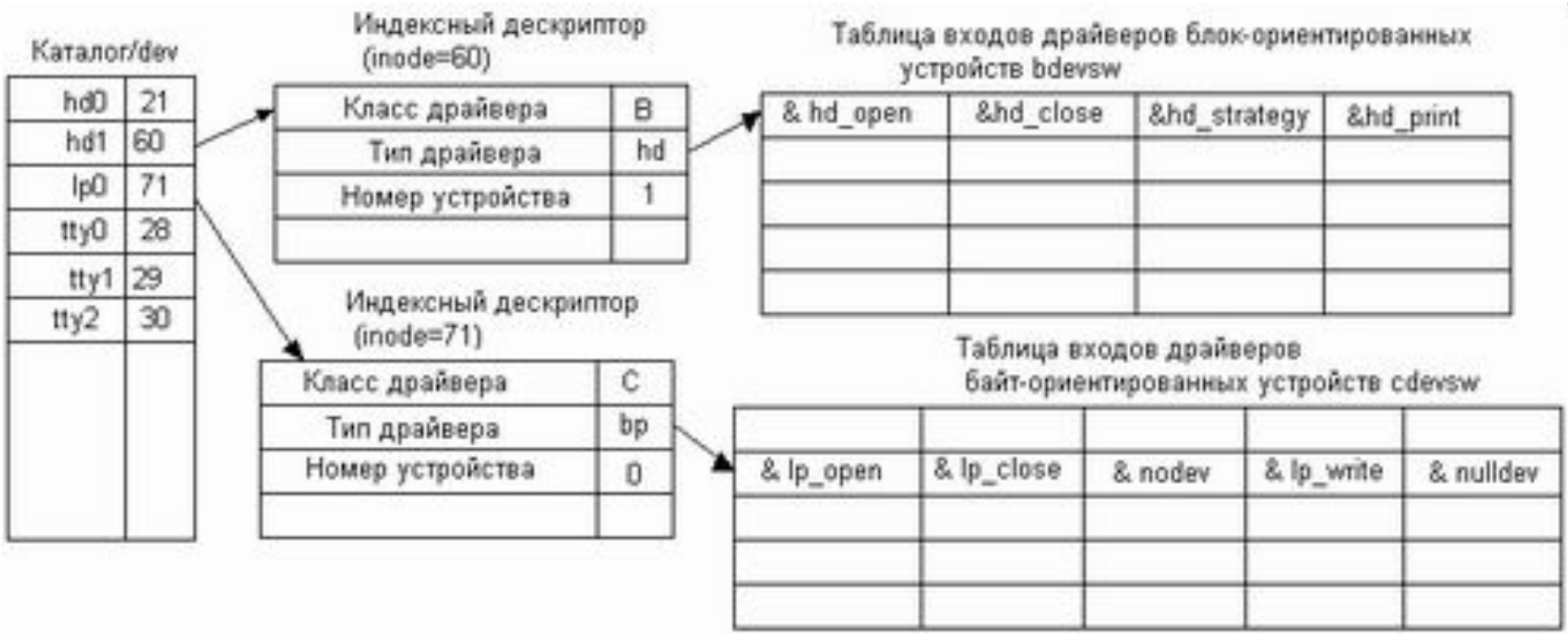
brw-rw---- 1 root disk 1, 10 Авг 31 16:04 ram10

- Адресная информация специального файла состоит из двух элементов:
- major — номер драйвера;
- minor — номер устройства.
- Значение major (номер драйвера) определяет выбор драйвера, обслуживающего данный специальный файл, а значение minor (номер устройства) передается драйверу в качестве параметра вызова и указывает ему на одно из нескольких однотипных устройств, которыми драйвер может управлять. Например, для дисковых драйверов номер устройства задает не только диск, но и раздел на диске.
-

- Например, следующая команда создает блок-ориентированный специальный файл для представления третьего раздела на втором диске четвертого SCSI-контроллера:
- mknod /dev/dsk/scsi b 32 33
- аргумент b определяет создание специального файла для блок-ориентированного драйвера, аргумент 32 определяет номер драйвера, который будет вызываться при открытии устройства /dev/dsk/scsi, а аргумент 33 декодируется самим драйвером (в нем закодированы данные о том, что нужно управлять третьим разделом на втором диске четвертого SCSI-контроллера).

- ОС UNIX использует для хранения информации об установленных аппаратных драйверах две системные таблицы:
- bdevsw — таблица блок-ориентированных драйверов;
- cdevsw — таблица байт-ориентированных драйверов.
- Номер драйвера (major) является индексом соответствующей таблицы.
- Таблицы bdevsw и cdevsw содержат адреса программных секций драйверов, причем одна строка таблицы описывает один драйвер. Такая организация логической связи между ядром UNIX и драйверами позволяет легко настраивать систему на новую конфигурацию внешних устройств путем модификации таблиц bdevsw и cdevsw.

Организация связи ядра UNIX с драйверами



Структура драйвера UNIX

Драйвер блок-ориентированного устройства состоит из следующих функций:

- open — выполняет процедуру логического открытия устройства;
- close — выполняет процедуру логического закрытия устройства;
- strategy — читает или записывает блок;
- print — выводит сообщение об ошибке;
- size — возвращает размер раздела, который представляет данное устройство.

- Указатели на эти функции (то есть их адреса) составляют строку в таблице bdevsw, описывающую один драйвер системы. Ядро UNIX вызывает нужную функцию драйвера, передавая ей параметры, необходимые для работы. Например, при вызове функции open ей передается номер устройства (minor), режим открытия (для чтения, для записи, для чтения и записи и т. д.), а также указатель на идентификаторы безопасности процесса, открывающего файл.

- Процедуры обработки прерываний драйвера в таблице bdevsw не указываются, их адреса помещаются в специальную системную структуру — таблицу прерываний. В UNIX все обработчики прерываний, в том числе и обработчики прерываний аппаратных драйверов, состоят из двух процедур, называемых соответственно top_half — верхняя часть обработчика прерываний и bottom_half — нижняя часть обработчика прерываний.

- В обязанности верхней части входит быстрая реакция на событие в устройстве, вызвавшее генерирование сигнала прерывания. При обработке верхних половин все прерывания с более низкими приоритетами блокируются аппаратно. Верхняя половина отвечает также за постановку в очередь на выполнение нижней половины обработчика прерываний драйвера, который выполняет менее срочную и более трудоемкую работу.
- Нижние половины обработчиков прерываний драйверов UNIX выполняются с низким уровнем приоритета, так что любые запросы прерываний устройств могут прервать их обработку.

- В качестве примера можно рассмотреть обработчик прерывания от сетевой карты, сообщающего, что принят ethernet-пакет. Он обязан сделать две вещи:

1. Взять пакет из буфера сетевой карты и сигнализировать сетевой карте, что пакет получен операционной системой. Это нужно сделать немедленно по получении прерывания, через миллисекунду в буфере будут уже совсем другие данные;
2. Поместить этот пакет в какие-либо структуры ядра, выяснить, к какому протоколу он относится, передать его в соответствующие функции обработки. Это нужно сделать как можно быстрее, чтобы обеспечить максимальную производительность сетевой подсистемы, но не обязательно немедленно.

- Соответственно, первое выполняет верхняя половина, а второе — нижняя.

Драйвер **байт-ориентированного** устройства состоит из следующих стандартных функций:

- open — открывает устройство;
- close — закрывает устройство;
- read — читает данные из устройства;
- write — записывает данные в устройство;
- ioctl — управляет вводом-выводом;
- poll — опрашивает устройство для выяснения, не произошло ли некоторое событие.

- Функции чтения и записи данных выполняют обмен заданной последовательности байт из буфера в области пользователя с контроллером символьного устройства.
- С помощью функции **ioctl** обычно устанавливается режим работы устройства, например задаются параметры COM-порта, такие как разрядность символов, режим проверки четности и т. п.

- Например, функция записи осуществляет передачу данных из пользовательского буфера процесса, выдавшего запрос на обмен, в системный буфер, организованный в виде очереди байт. Передача байт идет до тех пор, пока системный буфер не заполнится до некоторого, заранее определенного в драйвере уровня. Затем функция записи драйвера приостанавливается, выполнив системную функцию sleep, переводящую процесс, в рамках которого работает функция записи write, в состояние ожидания.
- Аналогично организована работа драйвера при чтении данных с устройства.

Дисковый кэш

- Запросы к блок-ориентированным внешним устройствам с прямым доступом (типичными представителями которых являются диски) перехватываются промежуточным программным слоем — подсистемой буферизации, называемой также **дисковым кэшем.**
- Дисковый кэш располагается между слоем драйверов файловых систем и блок-ориентированными драйверами.
- При поступлении запроса на чтение некоторого блока диспетчер дискового кэша просматривает свой буферный пул, находящийся в системной области оперативной памяти, и если требуемый блок имеется в кэше, то диспетчер копирует его в буфер запрашивающего процесса. Операция ввода-вывода считается выполненной, хотя физического обмена с устройством не происходило, при этом выигрыш во времени доступа к файлу очевиден.

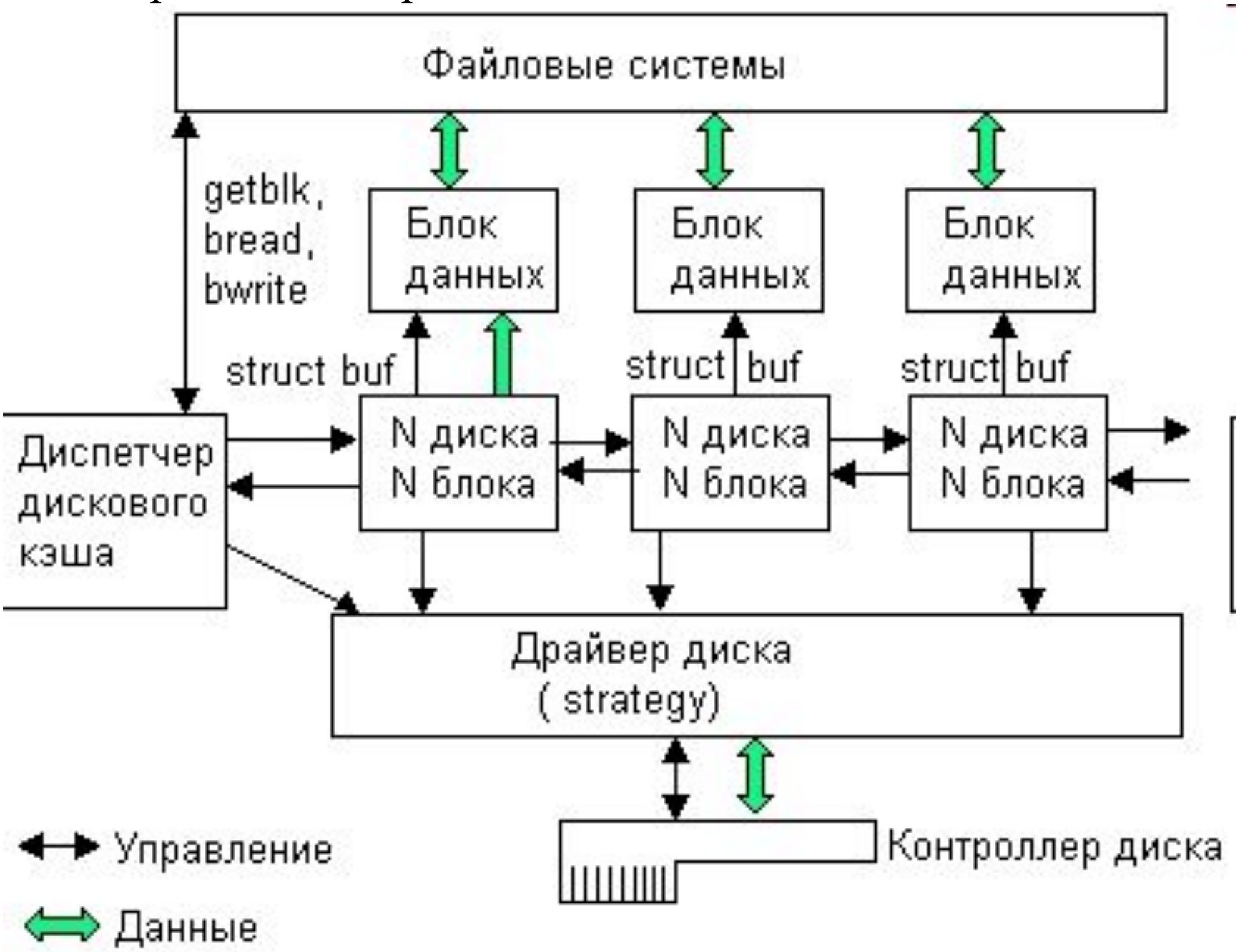
- Дисковый кэш обычно занимает достаточно большую часть оперативной системной памяти, чтобы максимально повысить вероятность попадания в кэш при выполнении дисковых операций.
- Доля оперативной памяти, отводимой под дисковый кэш, зависит от специфики функций, выполняемых компьютером, — например, сервер приложений выделяет под дисковый кэш меньше памяти, чем файловый сервер, чтобы обеспечить более высокую скорость работы приложений за счет предоставления им большего объема оперативной памяти.

Существуют два способа организации дискового кэша.

1. Способ, который можно назвать традиционным, основан на автономном диспетчере кэша, обслуживающем набор буферов системной памяти и самостоятельно организующим загрузку блока диска в буфер при необходимости, не обращаясь за помощью к другим подсистемам ОС.

2. Способ основан на использовании возможностей подсистемы виртуальной памяти. При этом способе функции диспетчера дискового кэша значительно сокращаются, так как большую часть работы выполняет подсистема виртуальной памяти, а значит, уменьшается объем ядра ОС и повышается его надежность.

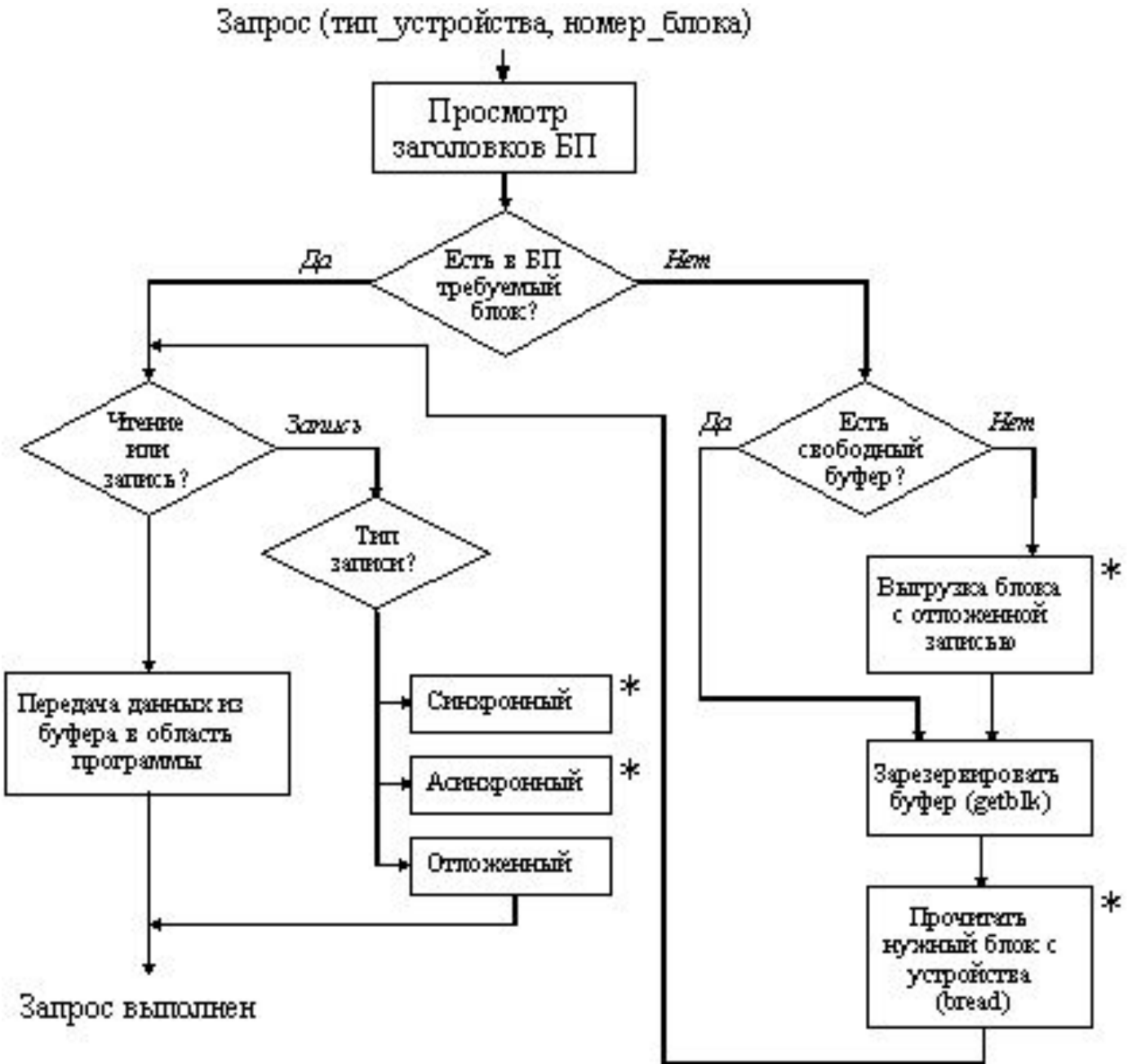
Организация традиционного дискового кэша



С каждым буфером связана специальная структура - заголовок буфера, в котором содержится следующая информация:

1. Данные о состоянии буфера:
   - занят/свободен,
   - чтение/запись,
   - признак отложенной записи,
   - ошибка ввода-вывода.
2. Данные об устройстве - источнике информации, находящейся в этом буфере:
   - тип устройства,
   - номер устройства,
   - номер блока на устройстве.
3. Адрес буфера.
4. Ссылка на следующий буфер в очереди свободных буферов, назначенных для ввода-вывода какому-либо устройству.

Схема выполнения запросов подсистемой буферизации



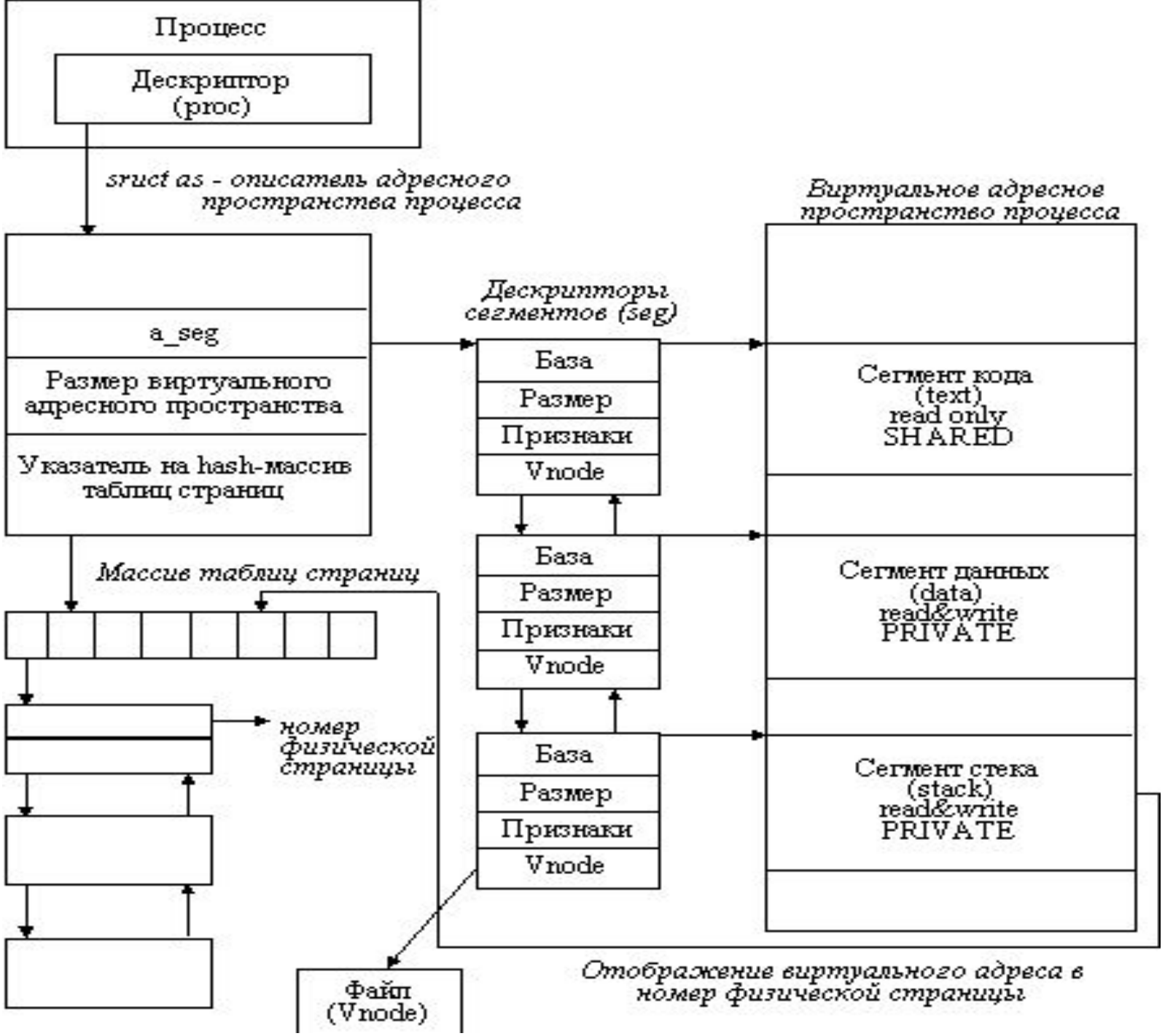
- Функция bwrite - синхронная запись. Процесс, выдавший запрос, ожидает результат выполнения операции ввода-вывода.
- Функция bawrite - асинхронная запись. Процесс не дожидается завершения операции ввода-вывода.
- Функция bdwrite - отложенная запись. При этом передача данных из системного буфера не производится, а в заголовке буфера делается отметка о том, что буфер заполнен и может быть выгружен, если потребуется освободить буфер.

Управление памятью ОС UNIX

- В UNIX реализована сегментно-страничная модель памяти. Наряду с механизмом управления страницами используется и механизм свопинга, когда на диск выталкиваются все страницы какого-либо процесса. Свопинг применяется в "предаварийных" ситуациях, когда размер свободной оперативной памяти уменьшается до некоторого заданного порога, так что работа всей системы очень затрудняется.

Структуры, описывающие виртуальное адресное пространство отдельного процесса

- В дескрипторе процесса proc содержится указатель на структуру as, с помощью которой описываются все виртуальные сегменты, которыми обладает данный процесс. Элемент a_seg в структуре as указывает на первый дескриптор сегмента процесса.
- Каждый дескриптор сегмента (структура seg) описывает один виртуальный сегмент процесса. Дескрипторы сегментов процесса связаны в двунаправленный список. Дескриптор сегмента содержит базовый адрес начала сегмента в виртуальном адресном пространстве процесса, размер сегмента, а также указатели на операции, которые допускаются над этим сегментом (дублирование, освобождение, отображение и т.д.).


Процесс
Дескриптор (proc)
sruct as - описатель адресного пространства процесса
Виртуальное адресное пространство процесса
a_seg
Размер виртуального адресного пространства
Указатель на hash-массив таблиц страниц
Дескрипторы сегментов (seg)
База
Размер
Признаки
Vnode
Сегмент кода (text) read only SHARED
Сегмент данных (data) read&write PRIVATE
Сегмент стека (stack) read&write PRIVATE
Массив таблиц страниц
номер физической страницы
Файл (Vnode)
Отображение виртуального адреса в номер физической страницы

Имеются следующие типы виртуальных сегментов:

- *Текст (text)* - содержит коды команд исполняемого модуля процесса. Он обычно обозначается "только для чтения", так чтобы ни сам процесс, ни другие процессы не могли изменить его кодовую часть. Текстовый сегмент может разделяться многими процессами, например, всеми пользователями, работающими с одним редактором.
- *Данные (data)* - содержит данные, используемые и модифицируемые процессом во время выполнения. К сегменту данных обычно разрешается иметь доступ для чтения и записи. В отличие от текстового сегмента, сегмент данных никогда не разделяется другими процессами.
- *Стек (stack)* - содержит стек процесса. Он помечается доступным для чтения и записи и не может разделяться другими процессами.

- Есть еще два типа сегментов:
- *Разделяемая память (shared memory)* - область памяти, доступная для чтения и записи нескольким процессам.
- *Отображенный файл (mapped file)* - сегменты отображенного файла используются для того, чтобы отобразить части файлов в адресное пространство процесса, и использовать стандартные механизмы ОС управления виртуальной памятью для ускорения доступа к файлам.

Ядро системы не выгружается на диск, остальная часть памяти доступна для страниц пользователей.

Кроме того Unix поддерживает динамически загружаемые модули, в основном драйверы устройств. Они могут быть большего размера и каждому из них должен быть выделен непрерывный участок памяти ядра.

Виртуальный адрес страницы состоит из 5 частей:

1. Глобальный каталог
2. Верхний каталог
3. Средний каталог
4. Страница
5. Смещение

В системе Unix используется 4-х уровневая схема таблиц страниц. Поля каталогов используются как индекс в соответствующем каталоге страниц. Глобальный каталог указывает на верхний каталог, верхний каталог указывает на средний и средний указывает на конкретную страницу, которую необходимо использовать в данный момент времени.

Unix различает 4 разных типа страниц:

1) Неиспользуемые страницы – страницы, которые не могут вытесняться в подкачку

2) Подкачиваемые страницы – страницы, которые должны быть записаны в область подкачки перед тем, как их можно будет использовать.

3) Синхронизируемые страницы – страницы, которые должны быть записаны на диск в том случае, если они были помечены как грязные (в них производилась запись).

4) Отбрасываемые страницы – страницы, которые могут быть использованы немедленно.

- Во время загрузки процесс init запускает страничные демоны kswapd и настраивает их на периодическое срабатывание. При каждом пробуждении поток kswapd проверяет, есть ли достаточное количество свободных страниц. Для этого он сравнивает нижний и верхний пределы с текущим уровнем использования памяти в каждой области памяти. Если памяти достаточно, то он отправляется обратно спать, хотя он может быть разбужен, если внезапно понадобятся дополнительные страницы. Если доступной памяти в одной из зон становится меньше нижнего предела, то kswapd инициирует алгоритм востребования страниц PFRA.

При каждом выполнении алгоритма PFRA (Page Frame Reclaming algorithm), он сначала пытается востребовать легкодоступные страницы, после чего переходит к труднодоступным. Востребованность страниц рассматривается в следующей очередности:

1) Отбрасываемые страницы и страницы, на которые нет ссылок.

2) Страницы с резервным хранением, на которые не было ссылок в последнее время

3) Совместно используемые страницы, которые не используются активно пользователями.

4) Обычные пользовательские страницы.

Также в системе существует еще один фоновый поток управления памятью. Данный поток либо пробуждается периодически для записи на диск грязных страниц, либо явным образом вызывается ядром системы для записи грязных страниц из кэша страниц на диск.

Процесс загрузки ОС Unix

1. **BIOS**

BIOS отвечает за базовый ввод/вывод данных с устройств/на устройства.

Делает проверки целостности устройств. За тестирование работоспособности электроники отвечает POST (Power-on self-test, «тест на адекватность себя самого», выполняющийся как этап презагрузки), который управляется BIOS.

Ищет, загружает и выполняет программу-загрузчик ОС

Берет загрузчик с жесткого диска или другого носителя.

Итог: BIOS загружает и выполняет загрузочную запись (MBR).

2. MBR

MBR — это главная загрузочная запись, хранящаяся на жестком диске

Она размещена в 1-м секторе загрузочного диска, например /dev/hda или /dev/sda

MBR состоит из трех компонентов:

1) главная загрузочная информация, находящаяся в первых 446 байтах;
2) информация о таблице разделов — в следующих 64 байтах;
3) последние 2 байта нужны для проверки корректности MBR .

Она содержит информацию о GRUB (или LILO).

Итог: MBR загружает и выполняет загрузчик GRUB.

3. GRUB

GRUB — Grand Unified Bootloader.

Если в системе установлено более, чем одно ядро, есть возможность выбирать, которое из них должен выполняться.

GRUB отображает заставку, и, подождав несколько секунд интерактивного воздействия пользователя, если он не нажал ни одной клавиши, он загружает ядро, установленное по умолчанию в файле конфигурации grub.

Конфигурационный файл Grub обычно расположен по пути /boot/grub/grub.conf (так же /etc/grub.conf может быть символьной ссылкой на него).

Конфигурационный файл содержит путь к ядру и образу initrd

Итог: GRUB загружает и выполняет образы ядра и initrd.

4. Ядро или Kernel
Ядро монтирует файловую систему в соответствии с настройкой «root=» в фале grub.conf
Выполняет программу /sbin/init
Поскольку init — это первый процесс, запущенный ядром Unux, он имеет идентификатор процесса (PID) №1.
initrd — это Initial RAM Disk - временный диск в оперативной памяти
initrd используется самим ядром в качестве временной корневой файловой системы, пока kernel не загрузится в реальную смонтированную файловую систему. Этот временный диск также содержит необходимые для загрузки драйверы, позволяющие получить доступ к разделам дисков и другому оборудованию

5. Init

Смотрит в файл /etc/inittab для того, чтобы определить уровень выполнения (run level).

Есть следующие уровни выполнения:

0 – прервать выполнение

1 – Однопользовательский режим, так называемый «Single user mode», или консоль восстановления

2 – Многопользовательский режим без поддержки NFS

3 – Полноценный многопользовательский режим

4 – не используется

5 – загрузка в многопользовательском режиме с поддержкой сети и графического входа систему

6 – перезагрузка

Init определяет уровень выполнения по умолчанию исходя из /etc/inittab и использует его для загрузки всех необходимых программ.

В большинстве случаев достаточно уровня 3 или 5.

6. Уровень выполнения программ (Runlevel)

Когда ОС выполняет свою загрузку, вы можете наблюдать загрузку различных служб. К примеру, это могут быть сообщения типа «starting Postfix … OK» (запускается Postfix).

Эти службы — и называются программами уровня выполнения, выполняемые из директории, которая соответствует нужному уровню выполнения.

Исходя из настроек по умолчанию, система будет выполнять файлы в соответствии с нижеприведенными директориями.

Выполнение уровня 0 – /etc/rc.d/rc0.d/
Выполнение уровня 1 – /etc/rc.d/rc1.d/
Выполнение уровня 2 – /etc/rc.d/rc2.d/
Выполнение уровня 3 – /etc/rc.d/rc3.d/
Выполнение уровня 4 – /etc/rc.d/rc4.d/
Выполнение уровня 5 – /etc/rc.d/rc5.d/
Выполнение уровня 6 – /etc/rc.d/rc6.d/

Но имейте ввиду, что еще в каталоге /etc могут быть символические ссылки. Например, /etc/rc0.d залинкован на /etc/rc.d/rc0.d.

- В каталогах /etc/rc.d/rc*.d/ вы можете увидеть список программ, имя которых начинается из букв S и K.
  Программы, начинающиеся на S используются для запуска (startup).
  Программы, которые начинаются с литеры K используются для завершения работы (kill).
  Еще есть номера рядом с буквами S и K в именах программ. Эти номера используются для определения порядка запуска этих программ.
  К примеру, S12syslog предназначен для запуска демона syslog, его порядковый номер 12. S80sendmail — для запуска демона sendmail, имеющего порядковый номер 80. Таким образом, программа syslog будет запущена перед sendmail.

Планирование заданий.

1. Очень часто в Linux администратор встречается с проблемой, когда выполнение какой-либо программы (или shell-сценария) может происходить и без его присутствия, но необходим инструмент, реализующий эту возможность.

Для этой цели принято использовать механизмы планирования заданий. Реализованы эти механизмы с помощью демонов планирования заданий – at и cron.

С помощью этих программ появляется возможность установить выполнение программы на заранее известное время. Команда at используется в тех случаях, когда выполнение задания — разовая процедура. Если же задание предполагается выполнять с какой-либо периодичностью, то лучше всего использовать демон cron и команду crontab.

Методика планирования представляет из себя понимание процессов, происходящих с сервером (или персональным компьютером) в каждый момент времени. Для планирования применяется форма, аналогичная приведенной ниже с дискретизацией в 5 минут:

01 число	0 м	5 м	10 м	15 м	20 м	25 м	30 м	35 м	40 м	45 м	50 м	55 м
0 ч	Резервная копия											
1 ч												
2 ч				Обновления Linux			Установка обновлений Linux					
3 ч												
4 ч	СМОТРИ НЕДЕЛЬНУЮ ФОРМУ											
5 ч												
6 ч												
7 ч												
8 ч	РЕЗЕРВ ДЛЯ НЕПРЕДВИДЕННЫХ СЛУЧАЕВ											
9 ч												

Составляется расписание на каждый день месяца. Совместно с ней составляется форма по дням недели, которая позволяет планировать выделенные 2 часа (или больше, если это потребуется). Пример такой формы для понедельника:

понедельник	0 м	5 м	10 м	15 м	20 м	25 м	30 м	35 м	40 м	45 м	50 м	55 м
4 ч	Ответ SARG											
5 ч												

После того, как будут выписаны все задания, стоящие в текущий момент, нужно будет найти подходящее место для вновь вставляемого задания. Опытные системные администраторы считают, что стоит выделять около часа в сутки в расписании заданий для того, чтобы всегда можно было вставить непредвиденное разовое задание, а также освобождать от выполнения заданий время наивысшей загрузки системы. Не советуется планировать несколько заданий на одной и то же время.

Семейство команд **at** (at, atq, atrm) представляет собой инструменты для выполнения задания в определенное время по таймеру. Для правильного функционирования данной команды в системе должен быть запущен демон **atd**. Он поддерживает очередь заданий, которые должны быть выполнены в то или иное время.

В отличие от команд at, демон **cron** и команда управления планированием **crontab** позволят точно планировать задания. Как в случае с at, задания запускает программа-демон crond. Команда crontab служит лишь для управления заданиями. Перед использованием команды необходимо создать файл, описывающий таблицу заданий. Формат файла таков:

минуты часы дни_месяца месяц дни_недели **команда**

```
0 10 * * *  /home/student/bin/script   #запуск в 10:00 ежедневно
15 * * * 1  /home/student/bin/script2  #в 15 минут каждого часа
```

Вся информация о пользователе обычно хранится в файлах /etc/passwd и /etc/grpoup.

/etc/passwd – этот файл содержит информацию о пользователях.

Запись для каждого пользователя занимает одну строку:

root:
1QydTRu2w$Cm5gk.6w6nmNdUjerh5pu:0:0:root:/root:/bin/bash

- **имя пользователя** – имя, используемое пользователем на все приглашения типа login при аутентификации в системе.
- **зашифрованный пароль** – обычно хешированный по необратимому алгоритму MD5 пароль пользователя.
- **UID** – числовой идентификатор пользователя. Система использует его для распределения прав файлам и процессам.
- **GID** – числовой идентификатор группы. Имена групп расположены в файле /etc/group. Система использует его для распределения прав файлам и процессам.
- **Настоящее имя пользователя** – используется в административных целях, а также командами типа finger (получение информации о

/etc/group – этот файл содержит информацию о группах, к которым принадлежат пользователи:

project:
1QydTRu2w$Cm5gk.6w6nmNdUjerh5pu:100:root,bin,daemon

Имя группы – имя, используемое для удобства использования таких программ, как newgrp.
Шифрованный пароль – используется при смене группы командой newgrp. **Пароль для групп** может отсутствовать.
GID – числовой идентификатор группы. Система использует его для распределения прав файлам и процессам.
Пользователи, включенные в несколько групп – В этом поле через запятую отображаются те пользователи, у которых по умолчанию (в файле /etc/passwd) назначена другая группа.
На сегодняшний день хранение паролей в файлах passwd и group считается ненадежным. В новых версиях Linux применяются так называемые теневые файлы паролей – shadow и gshadow. Права на них назначены таким образом, что даже чтение этих фалов без

Файл shadow хранит защищенную информацию о пользователях, а также обеспечивает механизмы устаревания паролей и учетных записей. Вот структура файла shadow:

cisco:1oAJZcVg0$EGORy8Mh3swT1RfJeX.UR0:13770:10:9999

а) имя пользователя

б) шифрованный пароль – применяются алгоритмы хеширования, как правило MD5

в) число дней последнего изменения пароля, начиная с 1 января 1970 года, последнего изменения пароля

г) число дней, перед тем как пароль может быть изменён

д) число дней, после которых пароль должен быть изменён

е) число дней, за сколько пользователя начнут предупреждать, что пароль устаревает

ж) число дней, после устаревания пароля для блокировки учётной записи

з) дней, отсчитывая с 1 января 1970 года, когда учётная запись будет заблокирована

и) зарезервированное поле

Файл shadow хранит защищенную информацию о пользователях, а также обеспечивает механизмы устаревания паролей и учетных записей. Вот структура файла shadow:

```
root:$1$QydTRu2w$Cm5gk.6w6nmNdUjerh5pu:root:cisco,oem
```

- ***Имя группы*** – имя, используемое для удобства использования таких программ, как newgrp.
- ***Шифрованный пароль*** – используется при смене группы командой newgrp. Пароль для групп может отсутствовать.
- ***Администратор группы*** – пользователь, имеющий право изменять пароль с помощью gpasswd.
- ***Список пользователей*** – В этом поле через запятую отображаются те пользователи, у которых по умолчанию (в файле /etc/passwd) назначена другая группа.

Тип операционной системы, установленной на компьютере. Для получения такой информации существует *утилита* `uname (Unix NAME)` .

`uname` , запущенная без параметров, покажет базовое имя системы:

```
uname
Linux
```

Также она может принимать следующие параметры:

`-s` – показывает название ядра системы

`-r` – имя релиза ядра системы

`-v` – имя версии, а также дату компиляции ядра

`-o` – операционную систему

`-p` – тип процессора

`-m` – *тип оборудования* (i386, i686, *Alpha*)

`-a` – всю информацию сразу

Команда free показывает объем памяти и объем ее использования, а также использование swap :

```
$ free
             total       used       free     shared    buffers     cached
Mem:        498916     483332      15584          0       4392     112924
-/+ buffers/cache:     366016     132900
Swap:      1453840     412532    1041308
```

Практически вся свободная память резервируется системой под дисковые буферы и дисковый кэш, что позволяет более эффективно работать с дисками.

Состояние системы в данный момент, степень ее загруженности и время без перезагрузок показывает *команда* `uptime`:

```
uptime
14:24:08 up 1 day, 6:01, 2 users, load average: 0.08, 0.19, 0.16
```

Первым идет *текущее время*, потом, после слова `up` – время, прошедшее с момента включения компьютера, потом показано сколько пользователей зарегистрировано сейчас в системе (это может быть и несколько регистраций одного и того же пользователя) и *загрузка* системы. *Загрузка* системы показывается в количестве процессов, одновременно работающих в системе, среднее *значение* за 1-ну, 5 и 15 минут. Система считается нагруженной, если это *значение* превышает 1 в расчете на 1 *процессор*.

Другим средством мониторинга производительности является *команда* `vmstat`:

```
procs -----------memory----- ---swap-- -----io---- --system-- -----cpu------
 r b      swpd free buff cache   si so     bi bo          in cs     us sy id wa st
0 0    268928 776168 15072 203316 1 2 10 14          207 225     13 3 84 0 0
```

Эта *команда* выдает за раз достаточно большой объем информации.

Раздел `procs`:

`r` — количество ожидающих процессов

`b` — количество спящих процессов

Раздел `memory`:

`swpd` — объем используемой виртуальной памяти

`free` — объем свободной виртуальной памяти

`buff` — объем памяти, занятой под дисковые буферы

`cache` - объем памяти, занятой под дисковый *кэш*

Раздел `swap`:

`si` — объем памяти, подкачанной с диска

`so` — объем памяти, выгруженной на *диск*

Раздел `io`:

`bi` — количество блоков, отправленных на *блочное устройство*

`bo` — количество блоков, прочитанных с *блочного устройства*

Раздел `system`:

`in` — количество прерываний в секунду

`cs` — количество переключений контекста в секунду

Раздел `cpu`:

`us` — *время выполнения* кода уровня пользователя (в процентах от общего времени)

`sy` — *время выполнения* кода уровня системы (в процентах от общего времени)

`id` — время простоя процессора (в процентах от общего времени)

`wa` — *время ожидания* ввода/вывода

`st` — время работы виртуальной машины уровня ядра

У `df` существует *ключ* `-h` (или `--human`), позволяющий увидеть объемы в привычных нам единицах измерения:

```
df --human
Файловая система Разм Исп Дост Исп% смонтирована на
 /dev/hdb2 36G 9,7G 24G 29% / /dev/hdb1 99M 16M 78M 17%
/boot tmpfs 633M 0 633M 0% /dev/shm
```

- **Tails** (“The Amnesic Incognito Live System”) является операционной системой такого же уровня как Windows или Macintosh OS, специально разработанной для обеспечения неприкосновенности частной жизни и анонимности своих пользователей. Она позволяет использовать Интернет в анонимном режиме, практически, в любом месте в сети Интернет и на любом компьютере, не оставляя каких-либо следов от выполненных операций.
- По умолчанию Tails нигде не фиксирует данные, только в ОП. Как только вы нажмете Reset на системнике, все данные исчезнут. Либо можете в стандартном меню Linux воспользоваться функциями незамедлительного выключения или перезагрузки.
- Даже если вы завиртуалили на флешке какие-то разделы, где храните информацию, рассекретить их невозможно без ключа. Внешние носители и флешки шифруются с помощью LUKS (это метод защиты, стандартный для Linux). Чтобы удалить данные с носителя, можно воспользоваться утилитой Nautilus Wipe – она стирает файлы без возможности восстановления.