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 (упрощение BCPL наследника CPL) и переработка Unix на этом языке – Томпсон.
5. 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.

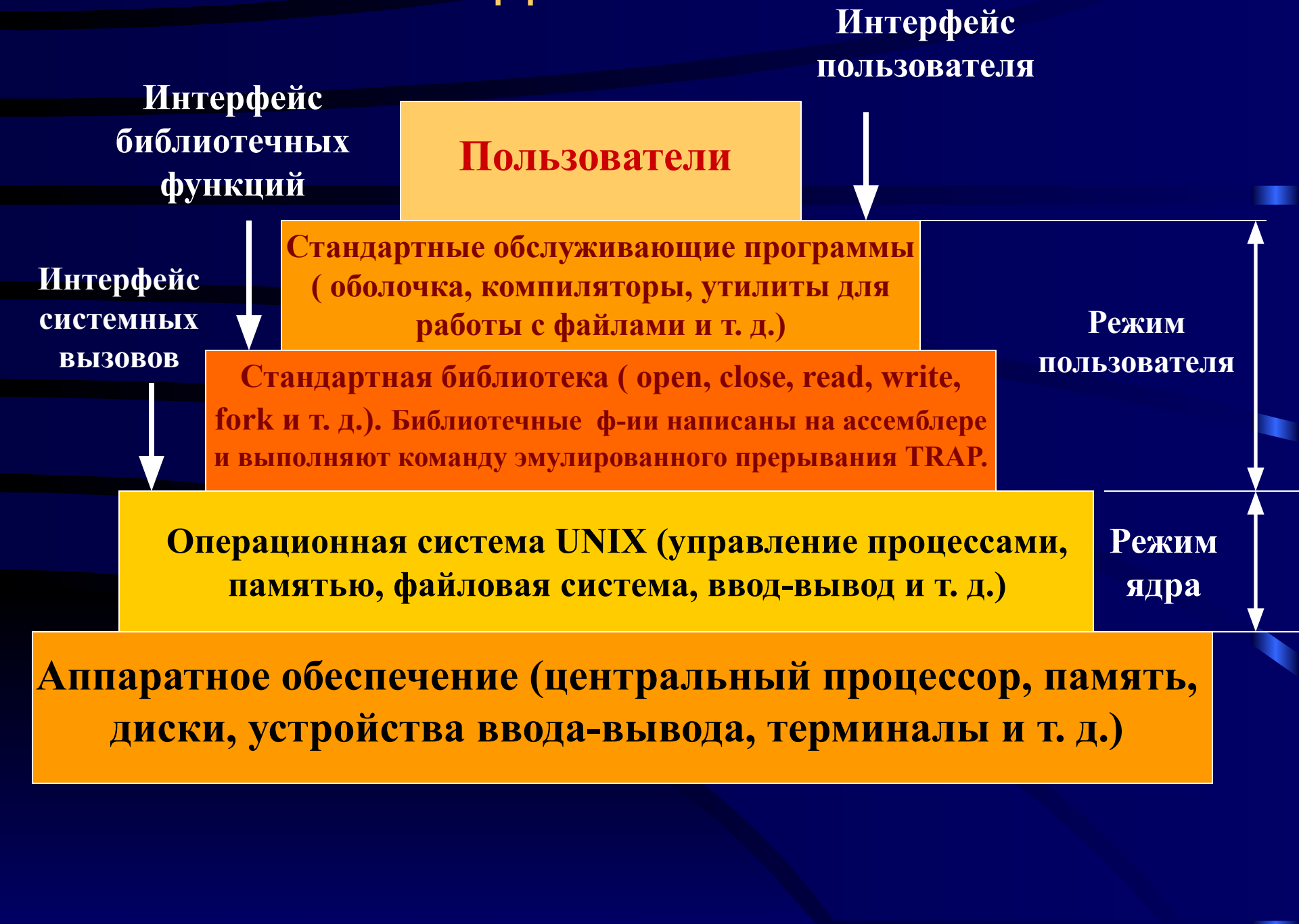
Общая характеристика системы UNIX

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 единое дерево 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

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 системе Unix

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 дочерним. У процессов собственные образы памяти. Открытые файлы 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 дочерняя программа */
}
```

Создание процесса

- **Процесс-родитель создает дочерние процессы, которые, в свою очередь, создают другие процессы, тем самым формируя *дерево процессов***
- **Разделение ресурсов**
  - **Процесс-родитель и дочерние процессы разделяют все ресурсы**
  - **Дочерние процессы разделяют подмножество ресурсов процесса-родителя**
  - **Процесс-родитель и дочерний процесс не имеют общих ресурсов**
- **Исполнение**
  - **Процесс-родитель и дочерние процессы исполняются совместно**
  - **Процесс-родитель ожидает завершения дочерних процессов**

Создание процесса

- Адресное пространство
  - Дочернего процесса копирует адресное пространство процесса-родителя
  - У дочернего процесса имеется программа, загруженная в него
- UNIX:
  - fork – системный вызов, создающий новый процесс
  - exec (execve) – системный вызов, используемый после fork, с целью замены пространства памяти процесса новой программой

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).

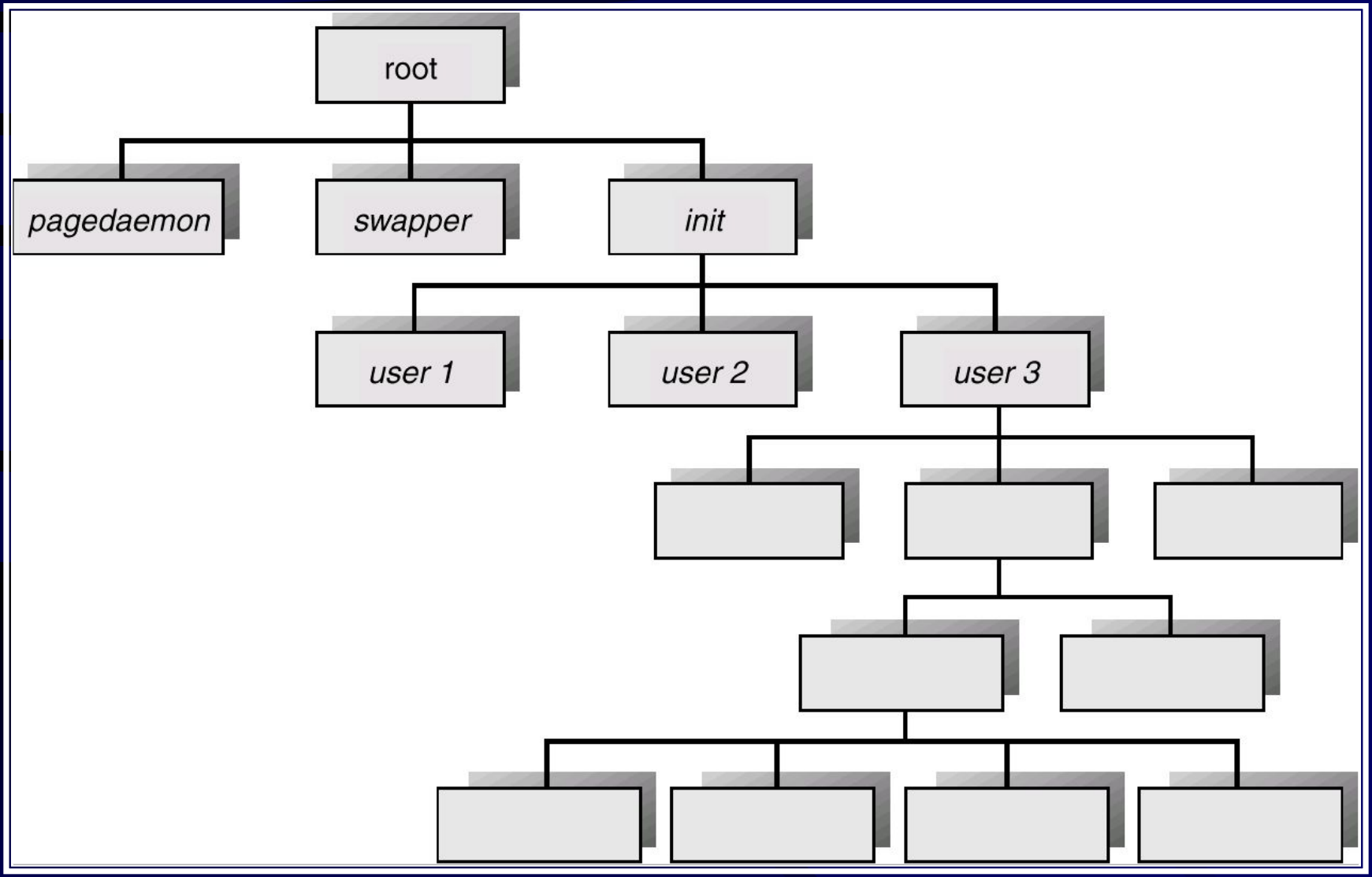
Таблица процессов содержит:

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.**

Дерево процессов в системе UNIX



Уничтожение процесса

- Процесс исполняет заключительный оператор и обращается к ОС для своей ликвидации (exit).
  - Передача данных от дочернего процесса процессу-родителю (wait).
  - Ресурсы процесса освобождаются операционной системой
- Процесс-родитель может уничтожить дочерние процессы (abort).
  - Дочерний процесс превысил выделенные ему ресурсы
  - Решения задачи, порученной дочернему процессу, больше не требуется
  - Происходит выход из процесса-родителя
    - ОС не допускает продолжения исполнения дочернего процесса, если его процесс-родитель уничтожается
    - “Каскадное” уничтожение процессов

- Процесс-зомби — дочерний процесс в Unix-системе, завершивший своё выполнение, но ещё присутствующий в списке процессов операционной системы, чтобы дать родительскому процессу считать код завершения.
- Процесс при завершении освобождает все свои ресурсы (за исключением PID — идентификатора процесса) и становится «зомби» — пустой записью в таблице процессов, хранящей код завершения для родительского процесса.

- Система уведомляет родительский процесс о завершении дочернего с помощью сигнала SIGCHLD. Предполагается, что после получения SIGCHLD он считает код возврата с помощью системного вызова wait(), после чего запись зомби будет удалена из списка процессов.

- Процесс-сирота — в семействе операционных систем UNIX вспомогательный процесс, чей основной процесс (или связь с ним) был завершен нештатно (не подав сигнала на завершение работы).
- Обычно, «сиротой» остается дочерний процесс после неожиданного завершения родительского, но возможно возникновение сервера-сироты (локального или сетевого) при неожиданном прерывании связи или завершении клиентского процесса.
- Процессы-сироты расходуют системные ресурсы сервера и могут быть источником проблем. Существует несколько их решений:
- Уничтожение— наиболее часто используется, заключается в немедленном завершении процесса (SIGKILL)
- Перевоплощение (англ. reincarnation) — система пытается «воскресить» родителей в состоянии на момент перед их удалением или найти других (например, более старших) родителей.
- Выдача лимита времени (англ. expiration) — процессу выдаётся временная квота для завершения до момента, когда он будет «убит» принудительно.
- В Unix-подобных системах все процессы-сироты немедленно усыновляются специальным системным процессом «init». Эта операция ещё называется (англ. re-parenting) и происходит автоматически. Хотя технически процесс «init» признаётся родителем этого процесса, его всё равно считают «осиротевшим», поскольку первоначально создавший его процесс более не существует.

- **Де́мон** (*daemon*) —служба, работающая в фоновом режиме без прямого общения с пользователем.
- Демоны обычно запускаются во время загрузки системы. Типичные задачи демонов: серверы сетевых протоколов (HTTP, *FTP*, электронная почта и др.), управление оборудованием, поддержка очередей печати, управление выполнением заданий по расписанию и т. д. В техническом смысле демоном считается процесс, который не имеет управляющего терминала.

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

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.

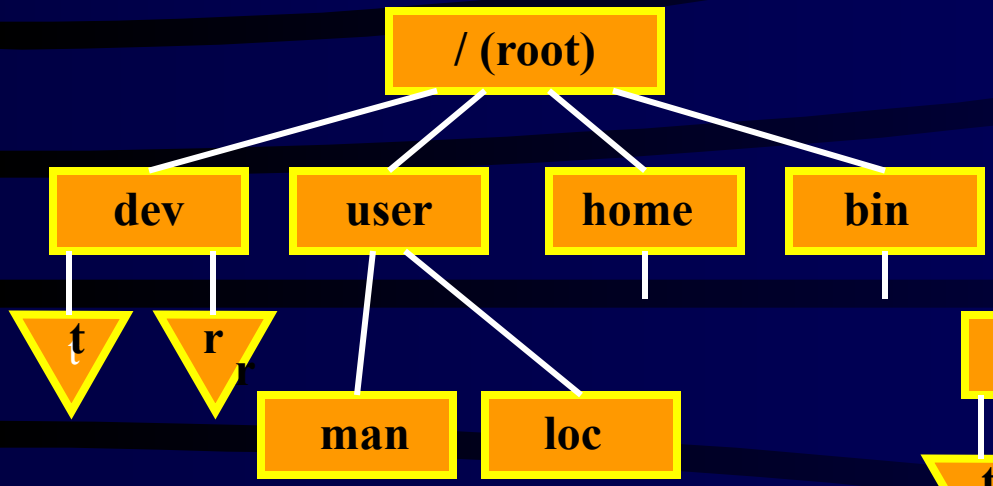
Priority = CPU_usage + nice + base

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.)

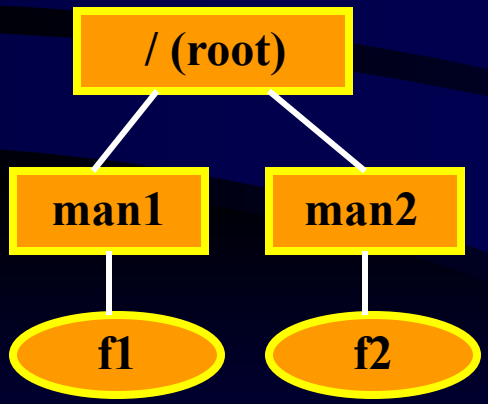
Priority пересчитывается каждую секунду.

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

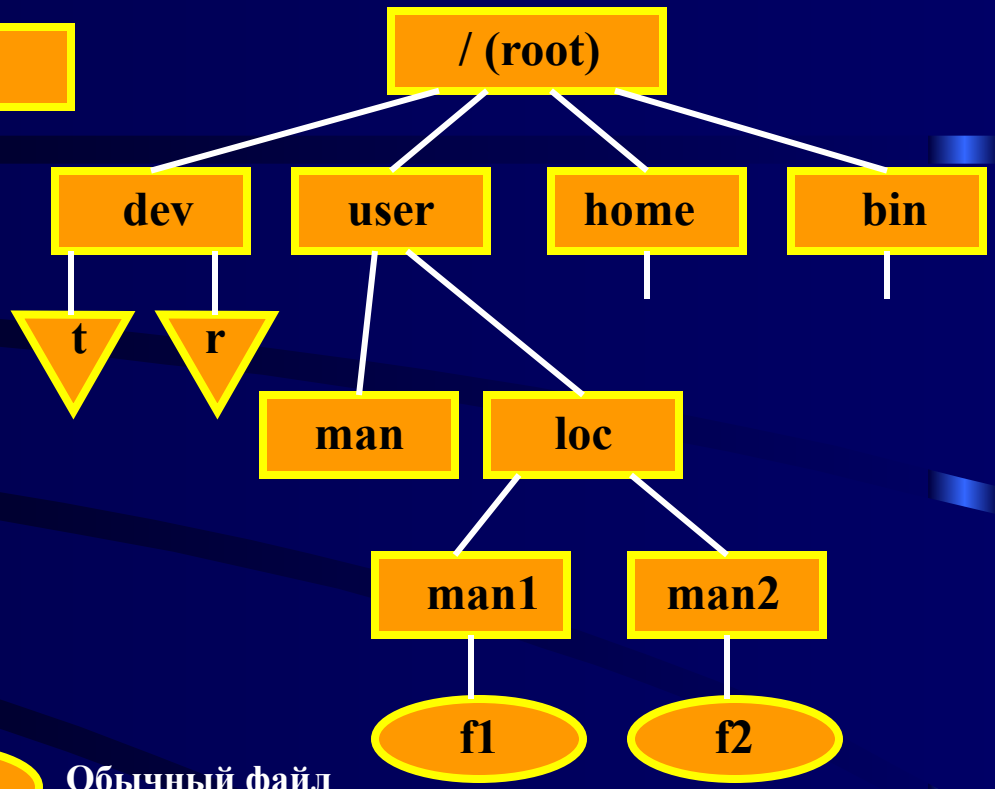
Монтирование



Файловая система 1



Файловая система 2



Общая файловая система
после монтирования

Типы файлов

- (дефис) — обычный файл;

d — каталог;

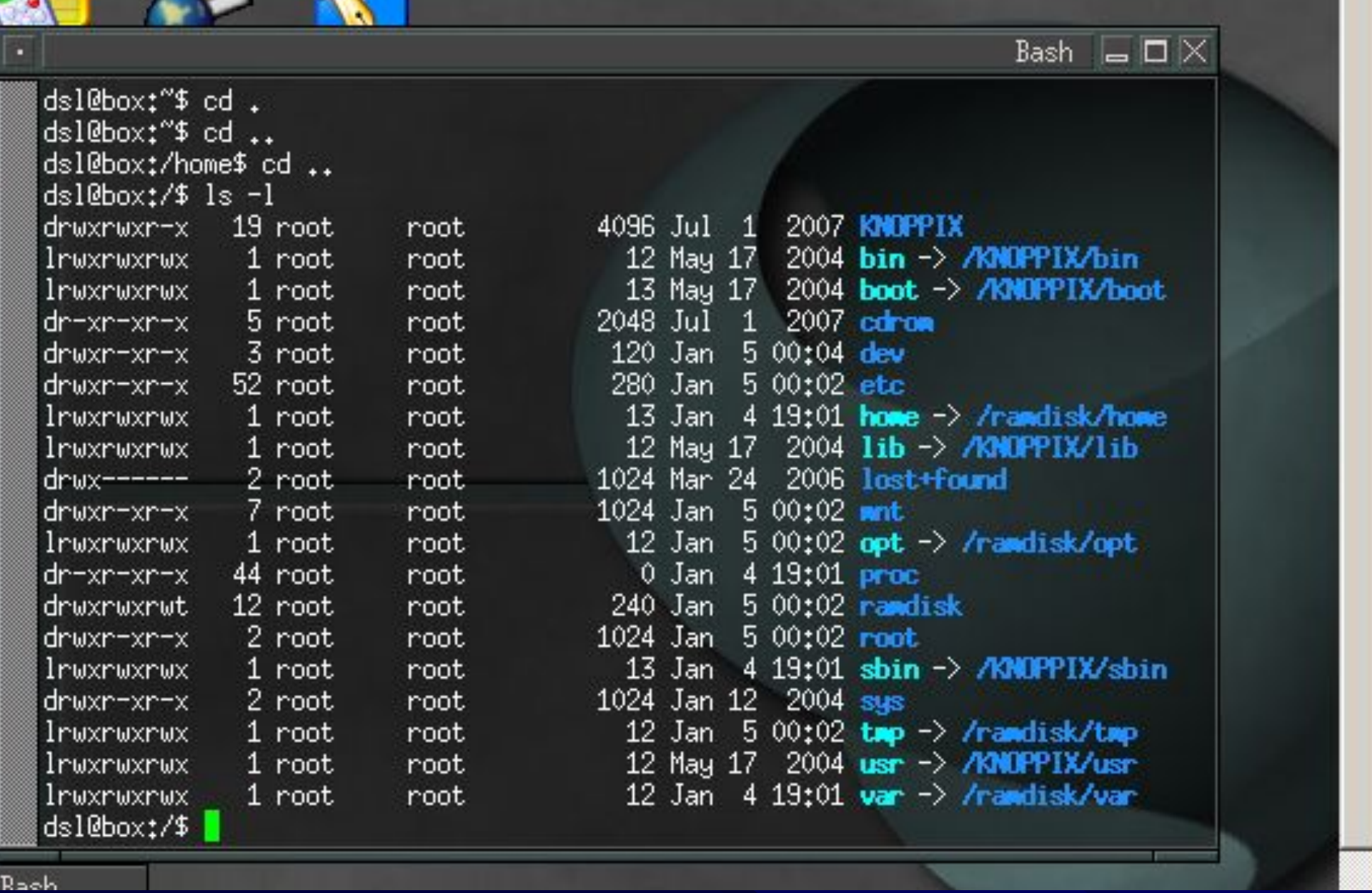
c — символьное устройство;

b — блочное устройство;

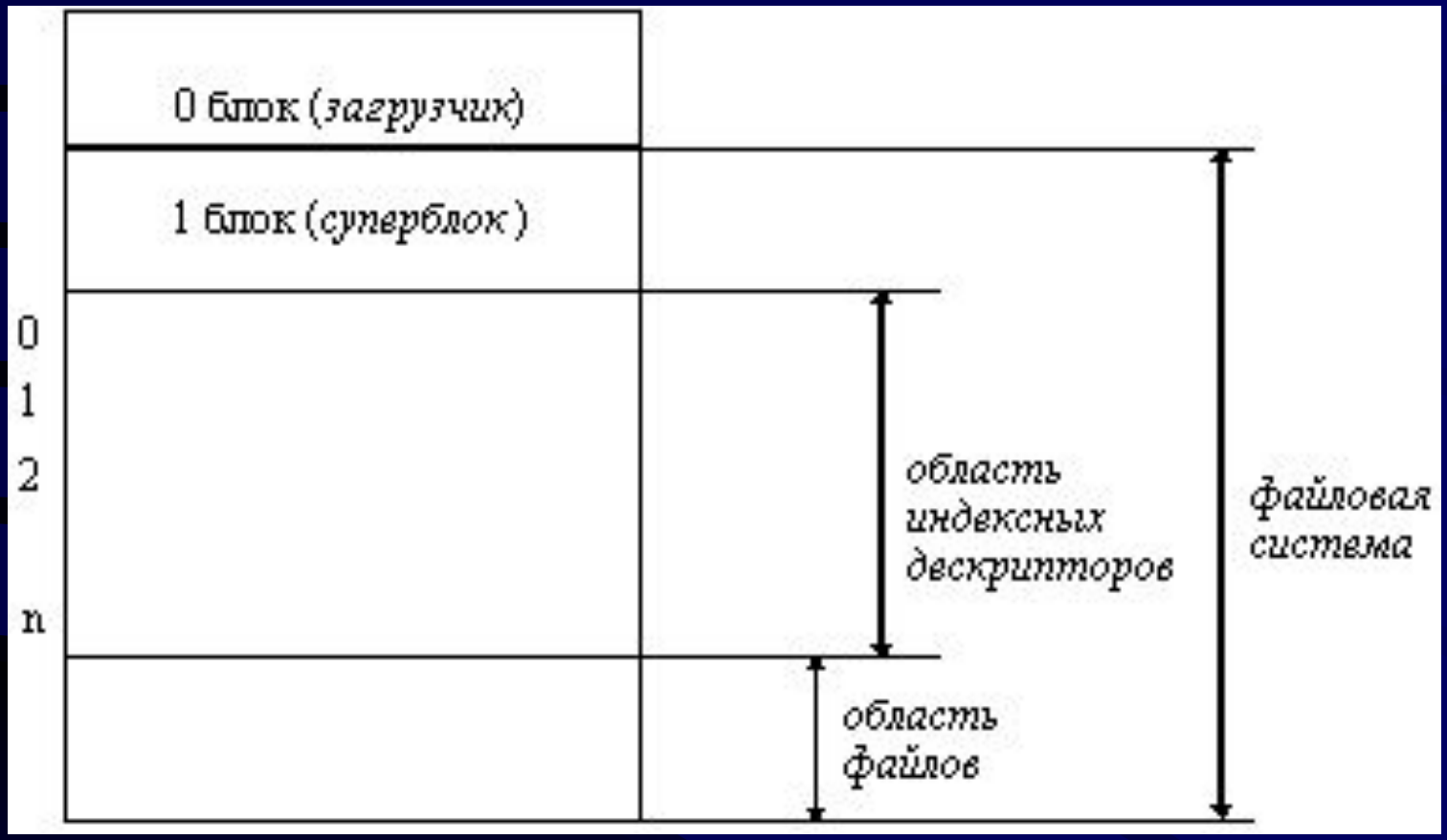
p — именованный канал (named pipe);

s — сокет (socket);

l — символическая ссылка.


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:/$

Физическая организация S5 и ufs



Расположение файловой системы s5 на диске

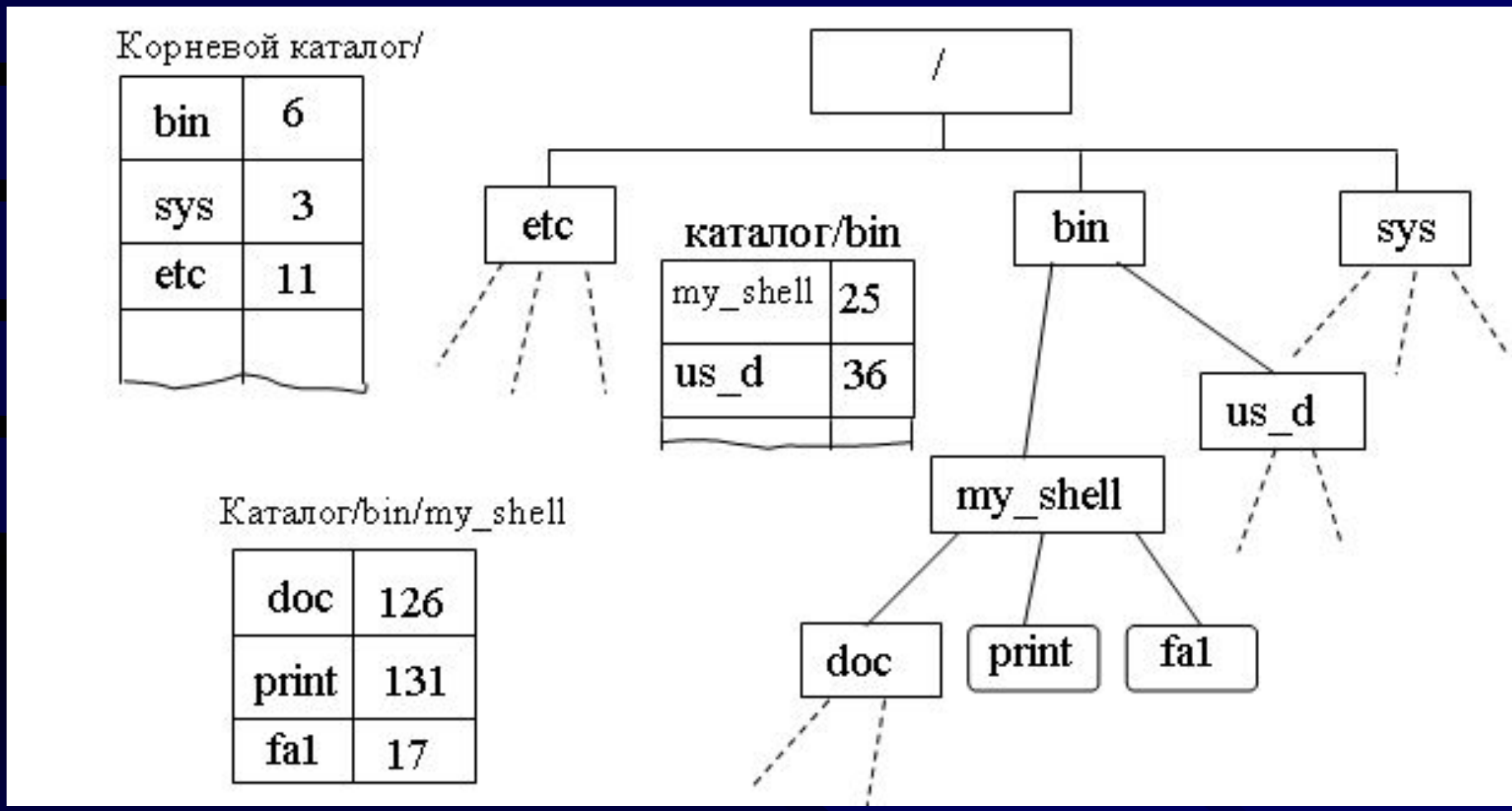
Структура *индексного дескриптора* (i-node)

- идентификатор владельцев файла;
- тип файла, файл может быть файлом обычного типа, каталогом, специальным файлом, конвейером и символьной связью;
- права доступа к файлу;
- временные характеристики: время последней модификации файла, время последнего обращения к файлу, время последней модификации индексного дескриптора;
- число ссылок на данный индексный дескриптор равно количеству псевдонимов файла;
- адресная информация ;
- размер файла в байтах.

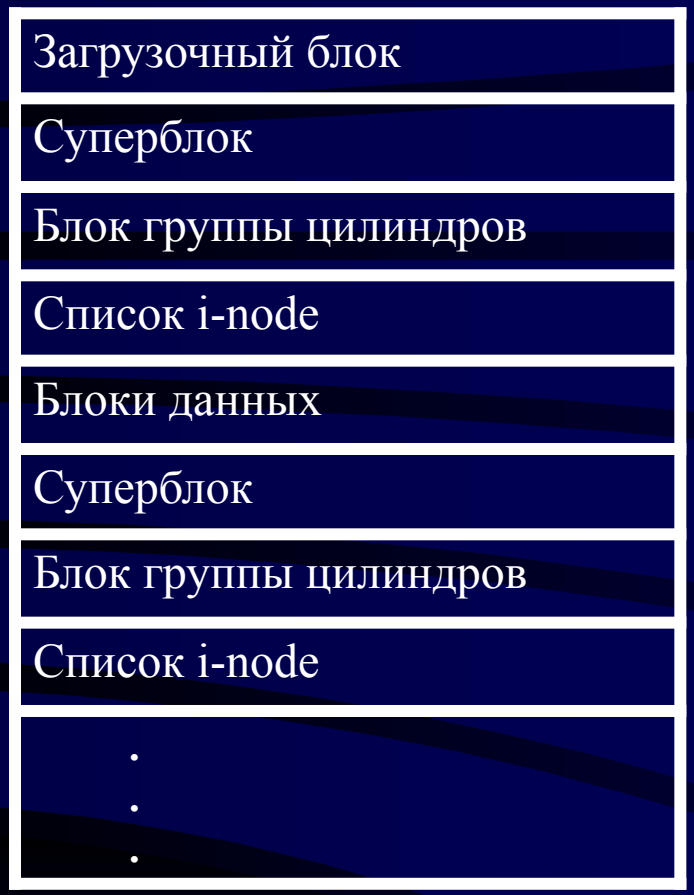
Копирование индексного дескриптора входит в процедуру открытия файла. При открытии файла ядро выполняет следующие действия:

- 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.

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.

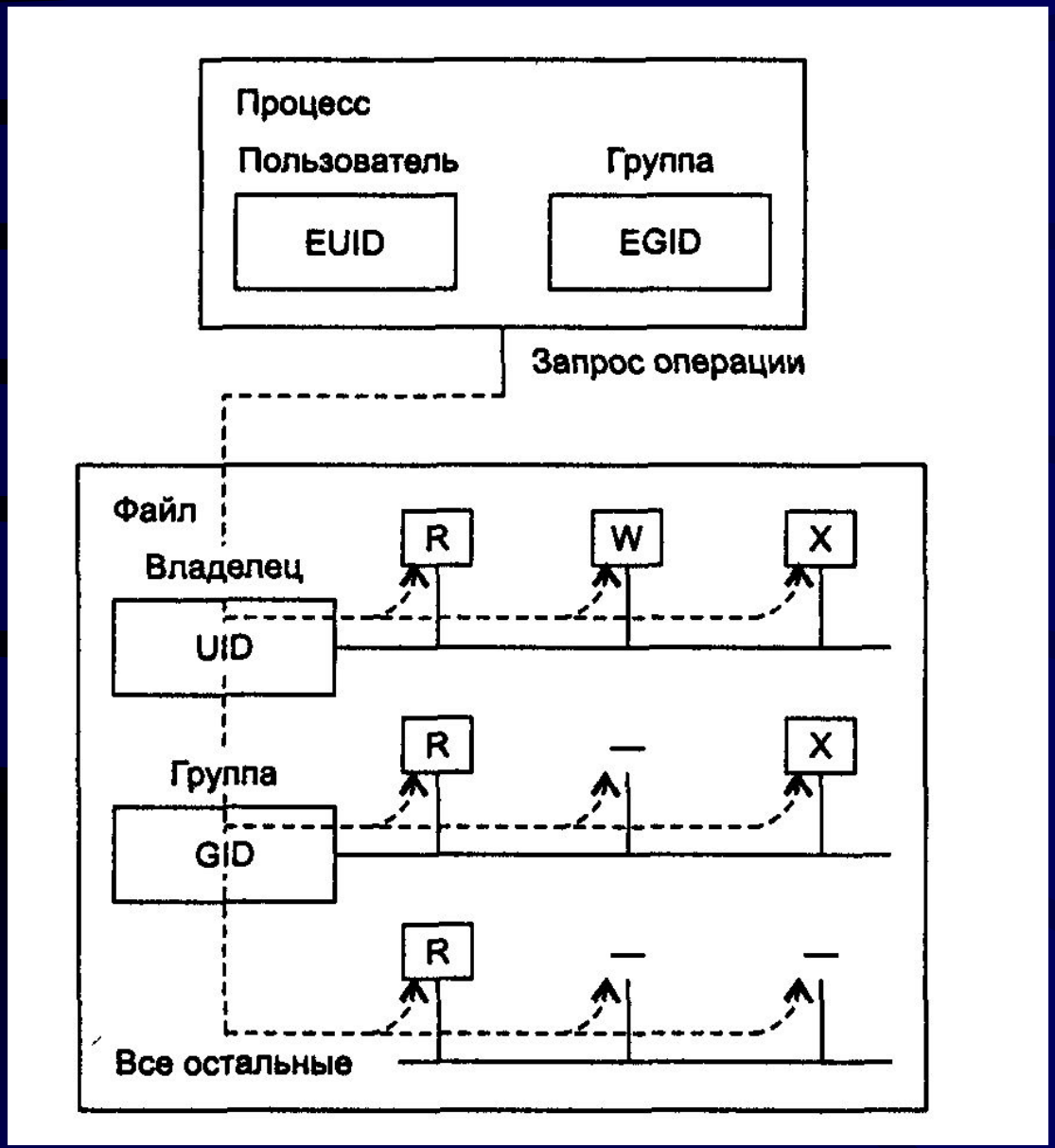


Физическая организация файловой системы *ufs*

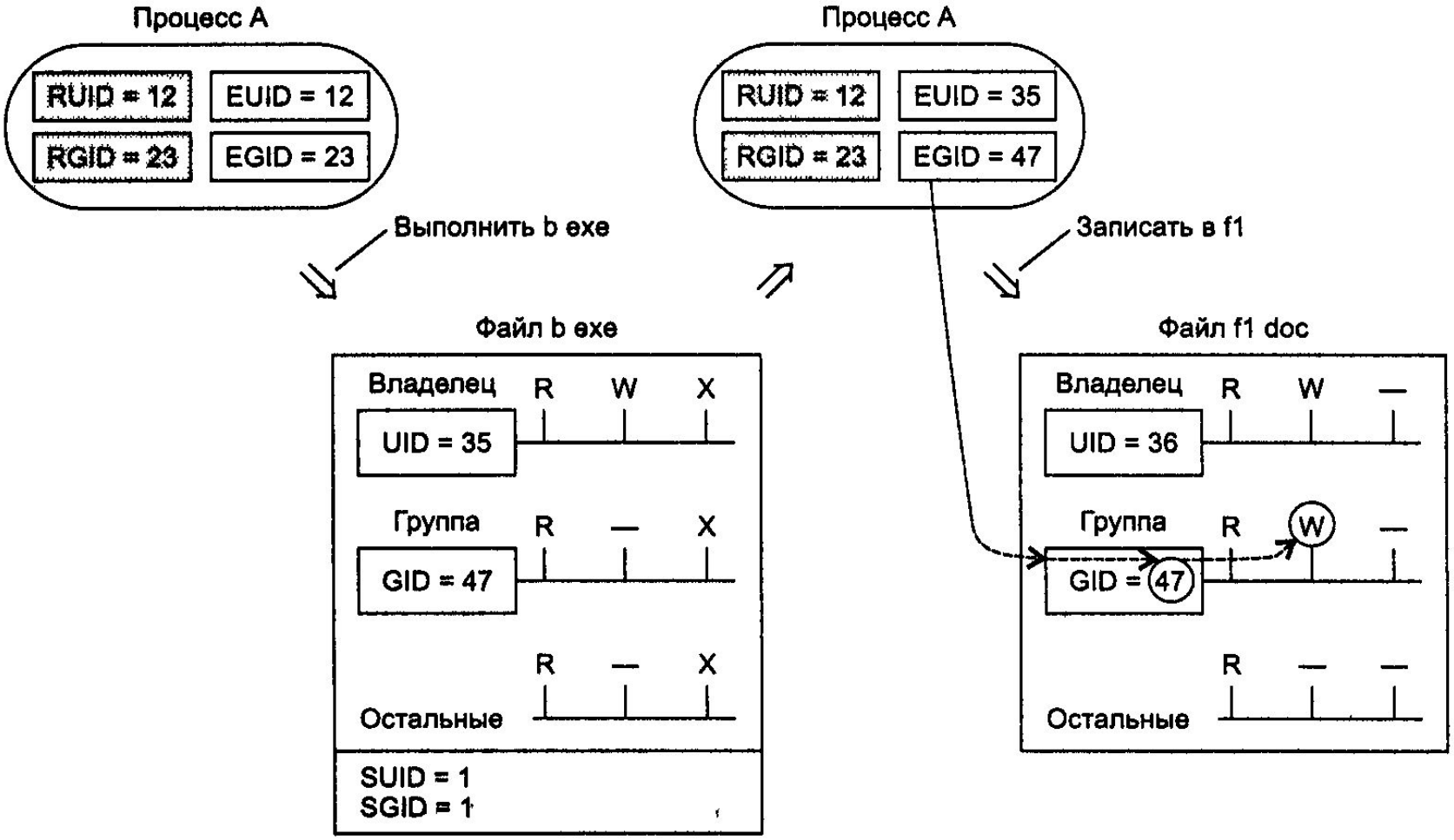
Реализация прав доступа в UNIX

- 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



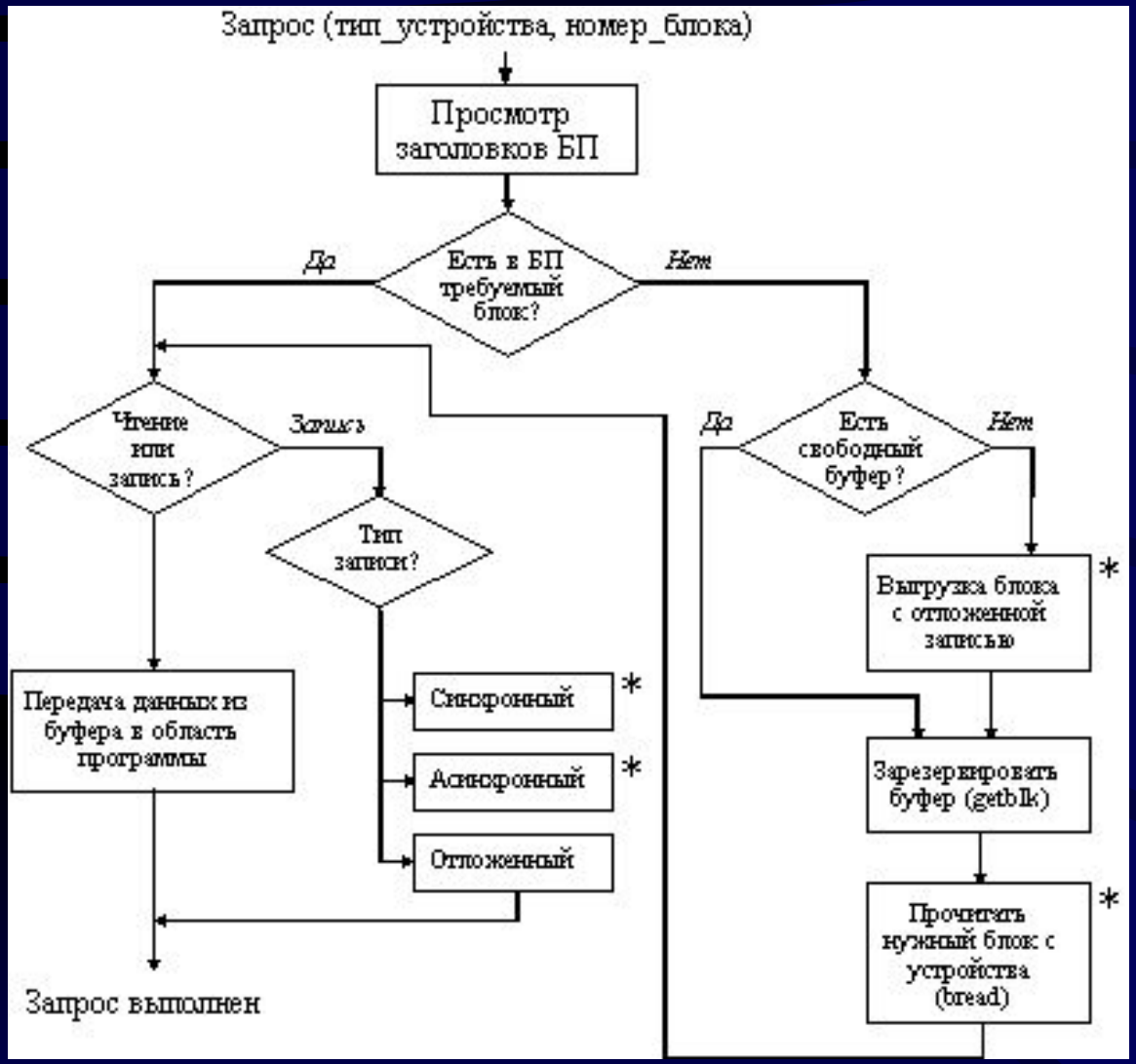
Система ввода-вывода

- 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.

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.

Упрощенная 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.

- **Последние новости о рынке операционных систем.**

Доли рынка серверов выглядят следующим образом (анализ данных за прошлый год):

Windows: 55,1%

Linux: 23,1%

UNIX: 11%

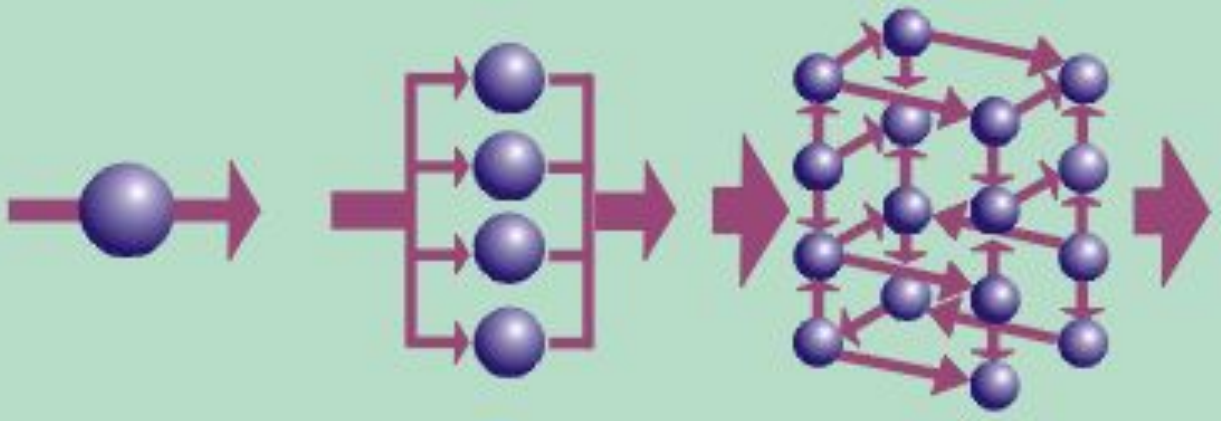
NetWare: 9,9%

Доли рынка настольных (клиентских) машин (опять же данные за прошлый год):

Windows: 93,8%

MacOS: 2,9%

Linux: 2,8%



Обычные компьютеры

содержат один центральный процессор, который выполняет всю работу, по одной операции за один такт.

Параллельная обработка

предполагает распределение задачи между несколькими процессорами.

Суперпараллельные компьютеры

содержат множество объединенных вместе процессоров.

Суперкомпьютер RoadRunner

создан компанией IBM для Министерства Энергетики США и установлен в Лос-Аламосской национальной лаборатории в Нью-Мексико, США.

- Суммарная информация по системе: 6120 двухъядерных процессоров Opteron с 49 Тбайт памяти (на 3060 LS21);
- 12240 процессоров Cell с 49 Тбайт памяти (на 6120 QS22);
- 204 узла ввода-вывода System x3755;
- 26 288-портовых маршрутизаторов ISR2012 Infiniband 4x DDR;
- 278 стоек;
- Энергопотребление системы 2.35 МВт.