Лекция 2

**Командная строка**

Основная среда взаимодействия с Linux - *командная строка*. Суть ее в том, что каждая строка, передаваемая пользователем системе, - это *команда*, которую та должна выполнить. Пока не нажата кнопка **Enter**, строку можно редактировать, затем она отсылается системе:[methody@localhost methody]$ cal

Сентября 2004

Вс Пн Вт Ср Чт Пт Сб

1 2 3 4

5 6 7 8 9 10 11

12 13 14 15 16 17 18

19 20 21 22 23 24 25

26 27 28 29 30

[methody@localhost methody]$ echo Hello, world!

Hello, world!

**Файлы**

Требования к хранению информации:

* возможность хранения больших объемов данных
* информация должна сохраняться после прекращения работы процесса
* несколько процессов должны иметь одновременный доступ к информации

**2.1.1 Именование файлов**

Длина имени файла зависит от ОС, может быть от 8 (MS-DOS) до 255 (Windows, LINUX) символов.

ОС могут различать прописные и строчные символы. Например, WINDOWS и windows для MS-DOS одно и тоже, но для UNIX это разные файлы.

Во многих ОС имя файла состоит из двух частей, разделенных точкой, например windows.exe. Часть после точки называют **расширением файла**. По нему система различает тип файла.

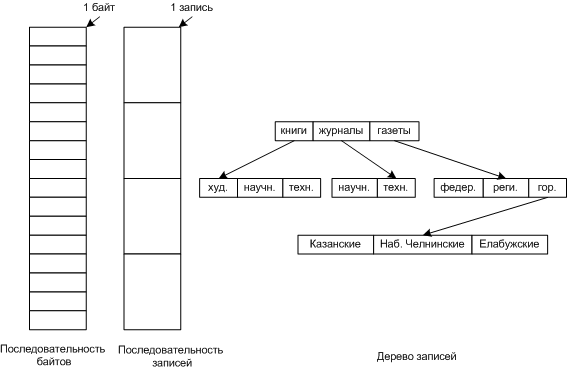
У MS-DOS расширение составляет 3 символа. По нему система различает тип файла, а также можно его исполнять или нет.

У UNIX расширение ограничено размером имени файла в 255 символов, также у UNIX может быть несколько расширений, но расширениями пользуются больше прикладные программы, а не ОС. По расширению UNIX не может определить исполняемый это файл или нет.

**2.1.2 Структура файла**

Три основные структуры файлов:

1. **Последовательность байтов** - ОС не интересуется содержимым файла, она видит только байты. Основное преимущество такой системы, это гибкость использования. Используются в Windows и UNIX.
2. **Последовательность записей** - записей фиксированной длины (например, перфокарта), считываются последовательно. Сейчас не используются.
3. **Дерево записей** - каждая запись имеет ключ, записи считываются по ключу. Основное преимущество такой системы, это скорость поиска. Пока еще используется на мэйнфреймах.



Три типа структур файла.

**2.1.3 Типы файлов**

Основные типы файлов:

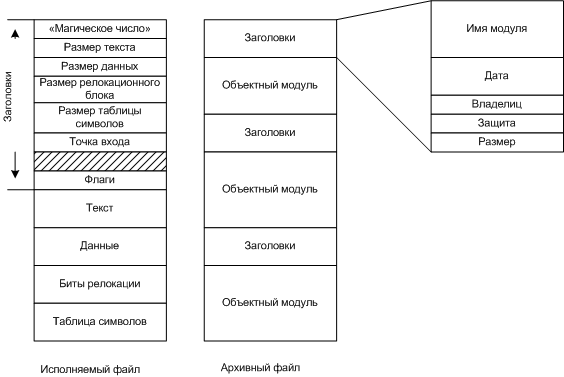
* **Регулярные** - содержат информацию пользователя. Используются в Windows и UNIX.
* **Каталоги** - системные файлы, обеспечивающие поддержку структуры файловой системы. Используются в Windows и UNIX.
* **Символьные** - для моделирования ввода-вывода. Используются только в UNIX.
* **Блочные** - для моделирования дисков. Используются только в UNIX.

Основные типы регулярных файлов:

* **ASCII файлы**- состоят из текстовых строк. Каждая строка завершается возвратом каретки (Windows), символом перевода строки (UNIX) и используются оба варианта (MS-DOS). Поэтому если открыть текстовый файл, написанный в UNIX, в Windows, то все строки сольются в одну большую строку, но под MS-DOS они не сольются (*это достаточно частая ситуация*). Основные преимущества ASCII файлов:  
  - могут отображаться на экране, и выводится на принтер без преобразований  
  - могут редактироваться почти любым редактором
* **Двоичные файлы**- остальные файлы (не ASCII). Как правило, имеют внутреннею структуру.

Основные типы двоичных файлов:

* **Исполняемые** - программы, их может обрабатывать сама операционная система, хотя они записаны в виде последовательности байт.
* **Неисполняемые** - все остальные.



Примеры исполняемого и не исполняемого файла

**«Магическое число»** - идентифицирующее файл как исполняющий.

**2.1.4 Доступ к файлам**

Основные виды доступа к файлам:

* **Последовательный** - байты читаются по порядку. Использовались, когда были магнитные ленты.
* **Произвольный** - файл можно читать с произвольной точки. Основное преимущество возникает, когда используются большие файлы (например, баз данных) и надо считать только часть данных из файла. Все современные ОС используют этот доступ.

**2.1.5 Атрибуты файла**

Основные атрибуты файла:

* Защита - кто, и каким образом может получить доступ к файлу (пользователи, группы, чтение/запись). Используются в Windows и UNIX.
* Пароль - пароль к файлу
* Создатель - кто создал файл
* Владелец - текущий владелец файла
* Флаг "только чтение" - 0 - для чтения/записи, 1 - только для чтения. Используются в Windows.
* Флаг "скрытый" - 0 - виден, 1 - невиден в перечне файлов каталога (по умолчанию). Используются в Windows.
* Флаг "системный" - 0 - нормальный, 1 - системный. Используются в Windows.
* Флаг "архивный" - готов или нет для архивации (не путать сжатием). Используются в Windows.
* Флаг "сжатый" - файл сжимается (подобие zip архивов). Используются в Windows.
* Флаг "шифрованный" - используется алгоритм шифрования. Если кто-то попытается прочесть файл, не имеющий на это прав, он не сможет его прочесть. Используются в Windows.
* Флаг ASCII/двоичный - 0 - ASCII, 1 - двоичный
* Флаг произвольного доступа - 0 - только последовательный, 1 - произвольный доступ
* Флаг "временный" - 0 - нормальный, 1 - для удаления файла по окончании работы процесса
* Флаг блокировки - блокировка доступа к файлу. Если он занят для редактирования.
* Время создания - дата и время создания. Используются UNIX.
* Время последнего доступа - дата и время последнего доступа
* Время последнего изменения - дата и время последнего изменения. Используются в Windows и UNIX.
* Текущий размер - размер файла. Используются в Windows и UNIX.

**2.1.6 Операции с файлами**

Основные системные вызовы для работы с файлами:

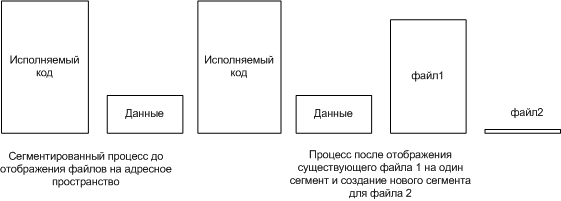
* Create - создание файла без данных.
* Delete - удаление файла.
* Open - открытие файла.
* Close - закрытие файла.
* Read - чтение из файла, с текущей позиции файла.
* Write - запись в файл, в текущею позицию файла.
* Append - добавление в конец файла.
* Seek - устанавливает файловый указатель в определенную позицию в файле.
* Get attributes - получение атрибутов файла.
* Set attributes - установить атрибутов файла.
* Rename - переименование файла.

**2.1.7 Файлы, отображаемые на адресное пространство памяти**

Иногда удобно файл отобразить в памяти (не надо использовать системные вызовы ввода-вывода для работы с файлом), и работать с памятью, а потом записать измененный файл на диск.

При использовании страничной организации памяти, файл целиком не загружается, а загружаются только необходимые страницы.

При использовании сегментной организации памяти, файл загружают в отдельный сегмент.



Пример копирования файла через отображение в памяти.

Алгоритм:

1. Создается сегмент для файла 1
2. Файл отображается в памяти
3. Создается сегмент для файла 2
4. Сегмент 1 копируется в сегмент 2
5. Сегмент 2 сохраняется на диске

Недостатки этого метода:

* Тяжело определить длину выходного файла
* Если один процесс отобразил файл в памяти и изменил его, но файл еще не сохранен, второй процесс откроет это же файл, и будет работать с устаревшим файлом.
* Файл может оказаться большим, больше сегмента или виртуального пространства.

**Понятие процесса**

**Процесс**(задача) - программа, находящаяся в режиме выполнения.

С каждым процессом связывается его **адресное пространство,** из которого он может читать и в которое он может писать данные.

Адресное пространство содержит:

* саму программу
* данные к программе
* стек программы

С каждым процессом связывается набор **регистров*,*** например:

* счетчика команд (в процессоре) - регистр в котором содержится адрес следующей, стоящей в очереди на выполнение команды. После того как команда выбрана из памяти, счетчик команд корректируется и указатель переходит к следующей команде.
* указатель стека
* и д.р.

Во многих операционных системах вся информация о каждом процессе, дополнительная к содержимому его собственного адресного пространства, хранится в **таблице процессов** операционной системы.

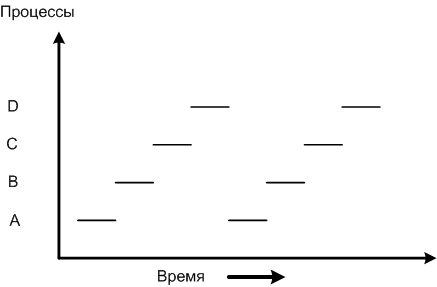
Некоторые поля таблицы:

|  |  |  |
| --- | --- | --- |
| **Управление процессом** | **Управление памятью** | **Управление файлами** |
| Регистры  Счетчик команд  Указатель стека  Состояние процесса  Приоритет  Параметры планирования  Идентификатор процесса  Родительский процесс  Группа процесса  Время начала процесса  Использованное процессорное время | Указатель на текстовый сегмент  Указатель на сегмент данных  Указатель на сегмент стека | Корневой каталог  Рабочий каталог  Дескрипторы файла  Идентификатор пользователя  Идентификатор группы |

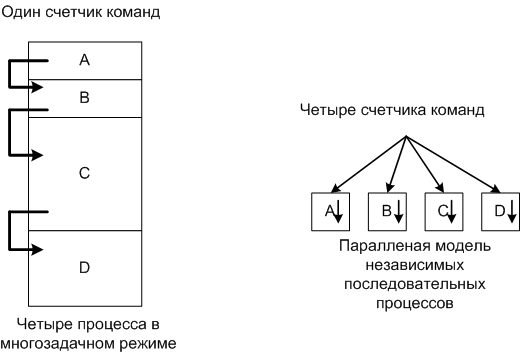
**4.1.2 Модель процесса**

В многозадачной системе реальный процессор переключается с процесса на процесс, но для упрощения модели рассматривается набор процессов, идущих параллельно (псевдопараллельно).

Рассмотрим схему с четырьмя работающими программами.



В каждый момент времени активен только один процесс



С права представлены параллельно работающие процессы, каждый со своим счетчиком команд. Разумеется, на самом деле существует только один физический счетчик команд, в который загружается логический счетчик команд текущего процесса. Когда время, отведенное текущему процессу, заканчивается, физический счетчик команд сохраняется в памяти, в логическом счетчике команд процесса.

**4.1.3 Создание процесса**

Три основных события, приводящие к созданию процессов (вызов **fork** или **CreateProcess**):

* Загрузка системы
* Работающий процесс подает системный вызов на создание процесса
* Запрос пользователя на создание процесса

Во всех случаях, активный текущий процесс посылает системный вызов на создание нового процесса.

В UNIX каждому процессу присваивается идентификатор процесса ( PID - Process IDentifier)

**4.1.4 Завершение процесса**

Четыре события, приводящие к остановке процесса (вызов **exit** или **ExitProcess**):

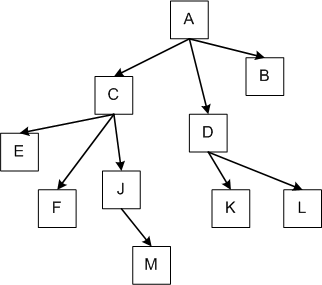
* Плановое завершение (окончание выполнения)
* Плановый выход по известной ошибке (например, отсутствие файла)
* Выход по неисправимой ошибке (ошибка в программе)
* Уничтожение другим процессом

Таким образом, приостановленный процесс состоит из собственного адресного пространства, обычно называемого **образом памяти**(**core image**), и компонентов таблицы процессов (в числе компонентов и его регистры).

**4.1.5 Иерархия процессов**

В UNIX системах заложена жесткая иерархия процессов. Каждый новый процесс созданный системным вызовом fork, является дочерним к предыдущему процессу. Дочернему процессу достаются от родительского переменные, регистры и т.п. После вызова fork, как только родительские данные скопированы, последующие изменения в одном из процессов не влияют на другой, но процессы помнят о том, кто является родительским.

В таком случае в UNIX существует и прародитель всех процессов - процесс **init**.

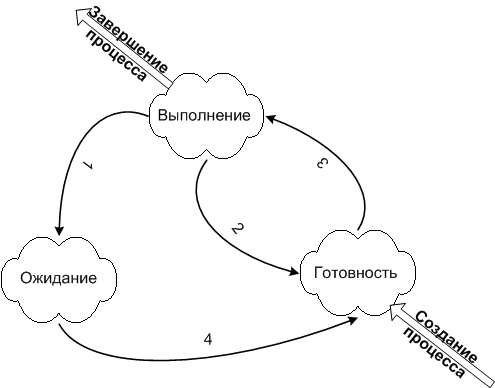


Дерево процессов для систем UNIX

**4.1.6 Состояние процессов**

Три состояния процесса:

* Выполнение (занимает процессор)
* Готовность (процесс временно приостановлен, чтобы позволить выполняться другому процессу)
* Ожидание (процесс не может быть запущен по своим внутренним причинам, например, ожидая операции ввода/вывода)



Возможные переходы между состояниями.

1. Процесс блокируется, ожидая входных данных

2. Планировщик выбирает другой процесс

3. Планировщик выбирает этот процесс

4. Поступили входные данные

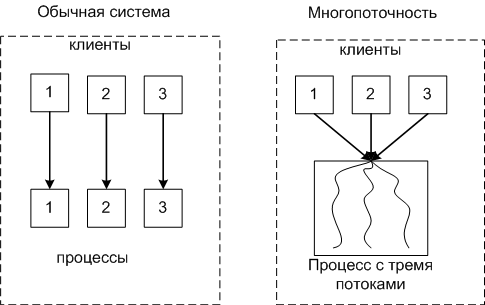
Переходы 2 и 3 вызываются планировщиком процессов операционной системы, так что сами процессы даже не знают о этих переходах. С точки зрения самих процессов есть два состояния выполнения и ожидания.

На серверах для ускорения ответа на запрос клиента, часто загружают несколько процессов в режим ожидания, и как только сервер получит запрос, процесс переходит из "ожидания" в "выполнение". Этот переход выполняется намного быстрее, чем запуск нового процесса.

**4.2 Потоки (нити, облегченный процесс)**

**4.2.1 Понятие потока**

Каждому процессу соответствует адресное пространство и одиночный **поток** исполняемых команд. В многопользовательских системах, при каждом обращении к одному и тому же сервису, приходится создавать новый процесс для обслуживания клиента. Это менее выгодно, чем создать квазипараллельный поток внутри этого процесса с одним адресным пространством.



Сравнение многопоточной системы с однопоточной

**4.2.2 Модель потока**

С каждым потоком связывается:

* Счетчик выполнения команд
* Регистры для текущих переменных
* Стек
* Состояние

Потоки делят между собой элементы своего процесса:

* Адресное пространство
* Глобальные переменные
* Открытые файлы
* Таймеры
* Семафоры
* Статистическую информацию.

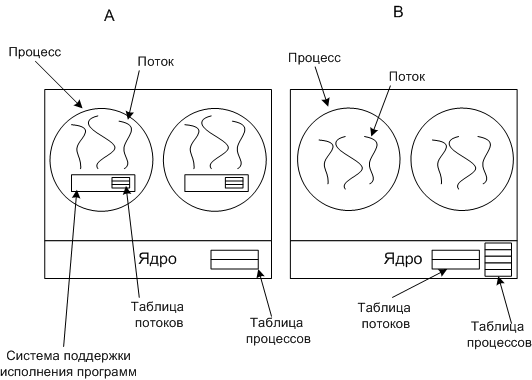
В остальном модель идентична модели процессов.

В POSIX и Windows есть поддержка потоков на уровне ядра.

**4.2.3 Преимущества использования потоков**

1. Упрощение программы в некоторых случаях, за счет использования общего адресного пространства.
2. Быстрота создания потока, по сравнению с процессом, примерно в 100 раз.
3. Повышение производительности самой программы, т.к. есть возможность одновременно выполнять вычисления на процессоре и операцию ввода/вывода. Пример: текстовый редактор с тремя потоками может одновременно взаимодействовать с пользователем, форматировать текст и записывать на диск резервную копию.

**4.2.4 Реализация потоков в пространстве пользователя, ядра и смешанное**



**А** - потоки в пространстве пользователя

**B** - потоки в пространстве ядра

В случае **А**ядро о потоках ничего не знает. Каждому процессу необходима **таблица потоков**, аналогичная таблице процессов.

Преимущества случая **А**:

* Такую многопоточность можно реализовать на ядре не поддерживающим многопоточность
* Более быстрое переключение, создание и завершение потоков
* Процесс может иметь собственный алгоритм планирования.

Недостатки случая **А**:

* Отсутствие прерывания по таймеру внутри одного процесса
* При использовании блокирующего (процесс переводится в режим ожидания, например: чтение с клавиатуры, а данные не поступают) системного запроса все остальные потоки блокируются.
* Сложность реализации

**Устройства ввода-вывода**

Устройства делят на две категории (некоторые не попадают ни в одну):

* блочные устройства - информация считывается и записывается по блокам, блоки имеют свой адрес (диски)
* символьные устройства - информация считывается и записывается посимвольно (принтер, сетевые карты, мыши)

**9.1.2 Контроллеры устройств**

Устройства ввода-вывода обычно состоят из двух частей:

* механическая (не надо понимать дословно) - диск, принтер, монитор
* электронная - **контроллер** или **адаптер**

Если интерфейс между контроллером и устройством стандартизован (ANSI, IEEE или ISO), то независимые производители могут выпускать совместимые как контроллеры, так и устройства. Например: диски IDE или SCSI.

Операционная система обычно имеет дело не с устройством, а с контроллером. Контроллер, как правило, выполняет простые функции, например, при считывании с диска, преобразует поток бит в блоки, состоящие из байт, и осуществляют контроль и исправление ошибок, проверяется контрольная сумма блока, если она совпадает с указанной в заголовке сектора, то блок считан без ошибок, если нет, то считывается заново.

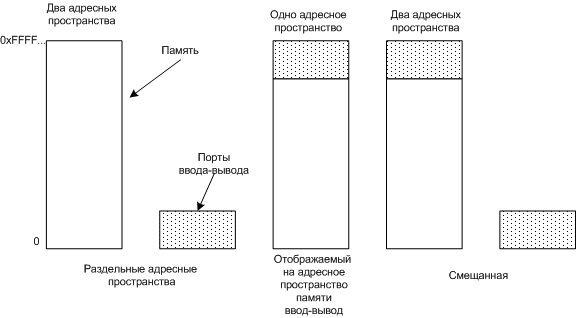
**9.1.3 Отображаемый на адресное пространство памяти ввод-вывод**

Каждый контроллер имеет несколько регистров, которые используются для взаимодействия с центральным процессором. При помощи этих регистров ОС управляет (считывает, пишет, включает и т.д.) и определяет состояние (готовность) устройства.

У многих устройств есть буфер данных (например: видеопамять).

Реализации доступа к управляющим регистрам и буферам:

* **номер порта ввода-вывода** - назначается каждому управляющему регистру 8- или 16-рзрядное целое число. Адресные пространства ОЗУ и устройства ввода-вывода в этой схеме не пересекаются.  
  Недостатки  
  - для чтения и записи применяются специальные команды, например: IN и OUT  
  - необходим специальный механизм защиты от процессов  
  - необходимо сначала считать регистр устройства в регистр процессора
* **отображаемый на адресное пространство памяти ввод-вывод**- регистры отображаются на адресное пространство памяти.  
  Недостатки  
  - при кэшировании памяти, могут кэшироваться и регистры устройств  
  - все устройства должны проверять все обращения к памяти, чтобы определить, на какие им реагировать. На одной общей шине это реализуется легко, но на нескольких будут проблемы.
* **смешанная реализация**- используется в х86 и Pentium,  
  от 0 до 64К отводится портам,  
  от 640 до 1М зарезервировано под буферы данных.



Способы реализации доступа к управляющим регистрам и буферам

**9.1.4 Прямой доступ к памяти (DMA - Direct Memory Access)**

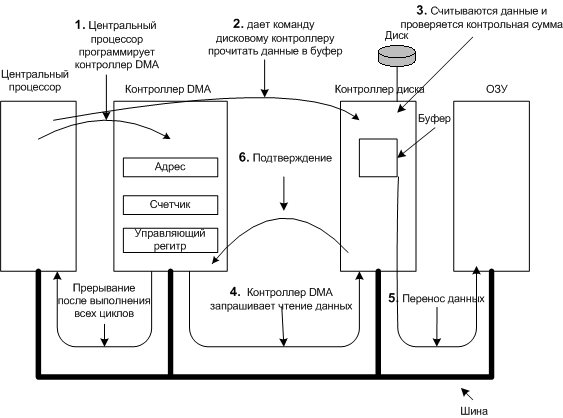
Прямой доступ к памяти реализуется с помощью **DMA - контроллера**.

Контроллер содержит несколько регистров:

* регистр адреса памяти
* счетчик байтов
* управляющие регистры, могут содержать:  
  - порт ввода-вывода  
  - чтение или запись  
  - единицы переноса (побайтно или пословно)

Без контроллера происходит следующее:

1. Процессор дает команду дисковому контроллеру прочитать данные в буфер,
2. Считываются данные в буфер, контроллер проверяет контрольную сумму считанных данных (проверка на ошибки). Процессор, до прерывания, переключается на другие задания.
3. Контроллер диска инициирует прерывание
4. Операционная система начинает работать и может считывать из буфера данные в память



Работа DMA - контроллера

С контроллером происходит следующее:

1. Процессор программирует контроллер (какие данные и куда переместить)
2. Процессор дает команду дисковому контроллеру прочитать данные в буфер
3. Считываются данные в буфер, контроллер диска проверяет контрольную сумму считанных данных, (процессор, до прерывания, переключается на другие задания).
4. Контроллер DMA посылает запрос на чтение дисковому контроллеру
5. Контроллер диска поставляет данные на шину, адрес памяти уже находится на шине, происходит запись данных в память
6. Когда запись закончена, контроллер диска посылает подтверждение DMA контроллеру
7. DMA контроллер увеличивает используемый адрес и уменьшает значение счетчика байтов
8. Все повторяется с пункта 4, пока значение счетчика не станет равной нулю.
9. Контроллер DMA инициирует прерывание

Операционной системе не нужно копировать данные в память, они уже там.

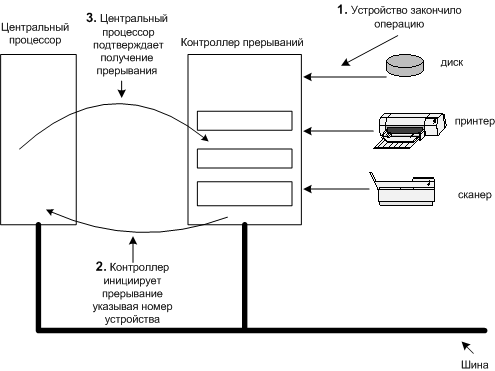
**9.1.5 Прерывания**

После того как устройство ввода-вывода начало работу, процессор переключается на другие задачи.

Чтобы сигнализировать процессору об окончании работы, устройство инициализирует прерывание, выставляя сигнал на выделенную устройству линию шины (а не выделенный провод).

**Контроллер прерываний** - обслуживает поступающие прерывания от устройств.

1. Если необработанных прерываний нет, прерывание выполняется немедленно.
2. Если необработанных прерываний есть, контроллер игнорирует прерывание. Но устройство продолжает удерживать сигнал прерывания на шине до тех пор, пока оно не будет обработано.



Работа прерываний

Алгоритм работы:

* Устройство выставляет сигнал прерывания
* Контроллер прерываний инициирует прерывание, указывая номер устройства
* Процессор начинает выполнять обработку прерывания, вызывая процедуру
* Эта процедура подтверждает получение прерывания контроллеру прерываний

**9.2 Принципы программного обеспечения ввода-вывода**

**9.2.1 Задачи программного обеспечения ввода-вывода**

Основные задачи, которые должно решать программное обеспечение ввода-вывода:

* Независимость от устройств - например, программа, читающая данные из файла не должна задумываться с чего она читает (CD, HDD и др.). Все проблемы должна решать ОС.
* Единообразное именование - имя файла или устройства не должны отличаться. (В системах UNIX выполняется дословно).
* Обработка ошибок - ошибки могут быть отловлены на уровне контроллера, драйвера и т.д.
* Перенос данных - **синхронный и асинхронный** (в последнем случае процессор запускает перенос данных, и переключается на другие задачи до прерывания).
* Буферизация
* Проблема выделенных (принтер) и невыделенных (диск) устройств - принтер должен предоставляться только одному пользователю, а диск многим. ОС должна решать все возникающие проблемы.

Три основных способа осуществления операций ввода-вывода:

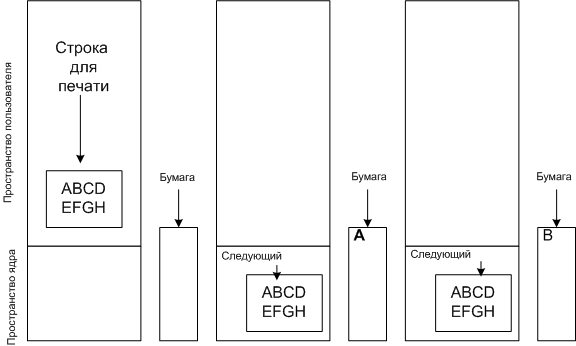
* Программный ввод-вывод
* Управляемый прерываниями ввод-вывод
* Ввод-вывод с использованием DMA

Рассмотрим их подробнее.

**9.2.2 Программный ввод-вывод**

В этом случае всю работу выполняет центральный процессор.

Рассмотрим процесс печати строки ABCDEFGH этим способом.



Этапы печати строки ABCDEFGH

Алгоритм печати:

1. Строка для печати собирается в пространстве пользователя.
2. Обращаясь к системному вызову, процесс получает принтер.
3. Обращаясь к системному вызову, процесс просит распечатать строку на принтере.
4. Операционная система копирует строку в массив, расположенный в режиме ядра.
5. ОС копирует первый символ в регистр данных принтера, который отображен на памяти.
6. Символ печатается на бумаге.
7. Указатель устанавливается на следующий символ.
8. Процессор ждет, когда бит готовности принтера выставится в готовность.
9. Все повторяется.

При использовании буфера принтера, сначала вся строка копируется в буфер, после этого начинается печать.

**9.2.3 Управляемый прерываниями ввод-вывод**

Если в предыдущем примере буфер не используется, а принтер печатает 100 символов в секунду, то на каждый символ будет уходить 10мс, в это время процессор будет простаивать, ожидая готовности принтера.

Рассмотрим тот же пример, но с небольшим усовершенствованием.

Алгоритм печати:

1. До пункта 8 тоже самое.
2. Процессор не ждет готовности принтера, а вызывает планировщик и переключается на другую задачу. Печатающий процесс блокируется.
3. Когда принтер будет готов, он посылает прерывание процессору.
4. Процессор переключается на печатающий процесс.

**9.2.4 Ввод-вывод с использованием DMA**

Недостаток предыдущего метода в том, что прерывание происходит при печати каждого символа.

Алгоритм не отличается, но всю работу на себя берет контроллер DMA.

**9.3 Программные уровни и функции ввода-вывода**

Четыре уровня ввода-вывода:



Уровни ввода-вывода

**9.3.1 Обработчики прерываний**

Прерывания должны быть скрыты как можно глубже в недрах операционной системы, чтобы как можно меньшая часть ОС имела с ними дело. Лучше всего блокировать драйвер, начавший ввод-вывод.

Алгоритм:

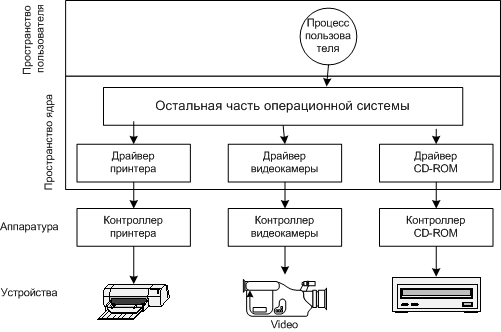
1. Драйвер начинает операцию ввод-вывод.
2. Драйвер блокирует сам себя,  
   - выполнив на семафоре процедуру down  
   - выполнив на переменной состояния процедуру wait  
   - выполнив на сообщении процедуру receive
3. Происходит прерывание
4. Обработчик прерываний начинает работу
5. Обработчик прерываний может разблокировать драйвер (например, выполнив на семафоре процедуру up)

**9.3.2 Драйвера устройств**

Драйвер устройства - необходим для каждого устройства. Для разных ОС нужны разные драйверы.

Драйверы должны быть частью ядра (в монолитной системе), что бы получить доступ к регистрам контроллера.

Это одна из основных причин приводящих к краху операционных систем. Потому что драйверы, как правило, пишутся производителями устройств, и вставляются в ОС.



Логическое расположение драйверов устройств. На самом деле обмен данными между контроллерами и драйверами идет по шине.

Драйвера должны взаимодействовать с ОС через стандартные интерфейсы.

Стандартные интерфейсы, которые должны поддерживать драйвера:

* Для блочных устройств
* Для символьных устройств

Раньше для установки ядра приходилось перекомпилировать ядра системы.

Сейчас в основном ОС загружают драйверы. Некоторые драйверы могут быть загружены в горячем режиме.

Функции, которые выполняют драйвера:

* обработка запросов чтения или записи
* инициализация устройства
* управление энергопотреблением устройства
* прогрев устройства (сканера)
* включение устройства или запуска двигателя

**9.3.3 Независимое от устройств программное обеспечение ввода-вывода**

Функции независимого от устройств программного обеспечения ввода-вывода:

* Единообразный интерфейс для драйверов устройств,
* Буферизация
* Сообщения об ошибках
* Захват и освобождение выделенных устройств (блокирование)
* Размер блока, не зависящий от устройств

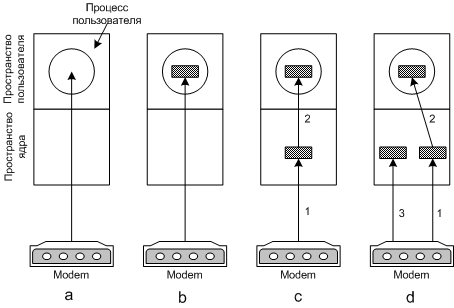
**Единообразный интерфейс для драйверов устройств**

Кроме интерфейса, в него также входят проблемы,

* именование устройств
* защита устройств

**Буферизация**

Рассмотрим несколько примеров буферизации.



a) Не буферизованный ввод - после ввода каждого символа происходит прерывание

b) Буферизация в пространстве пользователя - приходится держать загруженными необходимые страницы памяти в физической памяти.

c) Буферизация в ядре с копированием в пространство пользователя - страница загружается только когда буфер ядра полный, данные из буфера ядра в буфер пользователя копируется за одну операцию. Проблема может возникнуть, когда буфер ядра полный, а страница буфера пользователя еще не загружена.

d) Двойная буферизация в ядре - если один буфер заполнен, и пока он выгружается, символы пишутся во второй буфер.

**Сообщения об ошибках**

Наибольшее число ошибок возникает именно от операции ввода-вывода, поэтому их нужно определять как можно раньше. Ошибки могут быть очень разные в зависимости от устройств.

**Захват и освобождение выделенных устройств**

Для устройств (принтер) с которыми должен работать в одно время только один процесс, необходима возможность захвата и освобождения устройств. Когда один процесс занял устройство, остальные встают в очередь.

**Независимый от устройств размер блока**

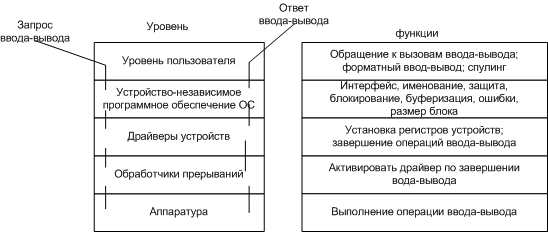
Размер блока должен быть одинаковый для верхних уровней, и не зависеть от устройств (размеров секторов на диске).

**9.3.4 Программное обеспечение ввода-вывода пространства пользователя**

Функции этого обеспечения:

* Обращение к системным вызовам ввода-вывода (через библиотечные процедуры).
* Форматный ввод-вывод (меняют формат, например, в ASCII)
* Спулинг (для выделенных устройств) - создается процесс (например, демон печати) и каталог спулера.

**9.3.5 Обобщение уровней и функций ввода-вывода**



Уровни и основные функции системы ввода-вывода