Операционные системы, среды и оболочки. Часть 1

Лекции – 18 час.

Лабораторные работы – 18 час.

Самостоятельная работа – 68 час.

Расчетно-графическое задание

Подготовка к зачету – 2 час.

Аттестация – диф. зачет

Лектор – доц. Кобылянский Валерий Григорьевич

Каф. ТПИ

Литература

1. Танненбаум Э., Бос Х. Современные операционные системы. – СПб.: «Питер», 2015. - 1120 стр.

2. Гордеев А.В. Операционные системы.- СПб.: «Питер» 2007 – 415 стр.

3. Олифер В. Г., Олифер Н. А. Сетевые операционные системы: Учебное пособие. – СПб.: Питер, 2009. – 669стр.

4. Назаров В.С. Современные операционные системы: учеб. пособие/ С.В. Назаров, А.И. Широков. – М.: Интернет - университет Информационных Технологий, 2011. – 279с.

5. Кобылянский В. Г. Операционные системы, среды и оболочки: учебное пособие. – Санкт-Петербург: Изд-во Лань, 2020. – 120 с.

6. Операционная система Linux: Курс лекций. Учебное пособие/ Г.В.Курячий, К.А.Маслинский – М. : ALT Linux; Издательство ДМК Пресс, 2010. -348с.

7. Курячий Г.В. Операционная система UNIX [Электронный ресурс]— М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 258 с.— Режим доступа: http://www.iprbookshop.ru/52199.html.— ЭБС «IPRbooks»

Тема 1. Классификации и объекты управления операционных систем

Операционная система (ОС) – комплекс программ, предназначенный для эффективного управления ресурсами ЭВМ и обеспечения интерфейса с пользователем.

ОС всегда содержит машинно-независимую и машинно-зависимую части кода. Вторая часть – это код, работающий непосредственно с аппаратурой.

Часть модулей ОС являются резидентными (т.е. находятся в ОЗУ постоянно), остальные модули подгружаются по мере необходимости. Резидентная часть обычно называется **ядром** ОС.

Ресурсы ЭВМ – аппаратные, программные, информационные.

Возможные классификации ОС – по способу обработки задач, по количеству одновременно обслуживаемых пользователей, по критерию эффективности, по аппаратной платформе, по истории развития, по архитектуре.

Классификации ОС

1. По способу обработки задач:

- однозадачные (MS-DOS, MSX);
- многозадачные (UNIX, Windows, MacOS, OS/2).

2. По количеству пользователей:

- однопользовательские (MacOS, Android, Windows для ПК);
- многопользовательские (UNIX, Windows Server, OS/2) ;

3. По критерию эффективности:

- общего назначения (UNIX, Windows, MacOS, OS/2);
- реального времени (VxWorks, QNX, OS-9).

4. По аппаратной платформе:

- для мобильных устройств;
- для персональных компьютеров;
- для мини-компьютеров;
- для мейнфреймов;
- для кластеров и сетей ЭВМ.

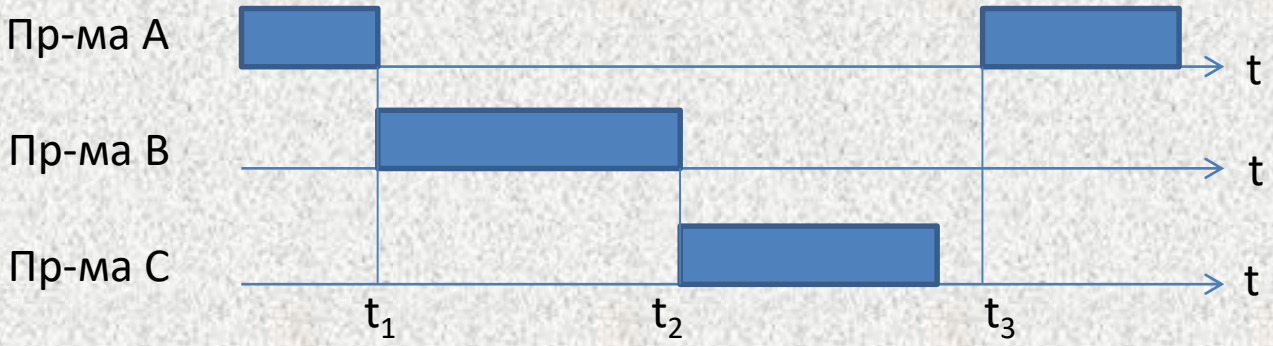
5. По истории развития:

- мониторные (ЭВМ I поколения);
- пакетные (ЭВМ II поколения);
- разделения времени (ЭВМ III - V поколений),
- реального времени (ЭВМ III - V поколений);
- многопроцессорные (ЭВМ IV - V поколений);
- сетевые (ЭВМ IV - V поколений).

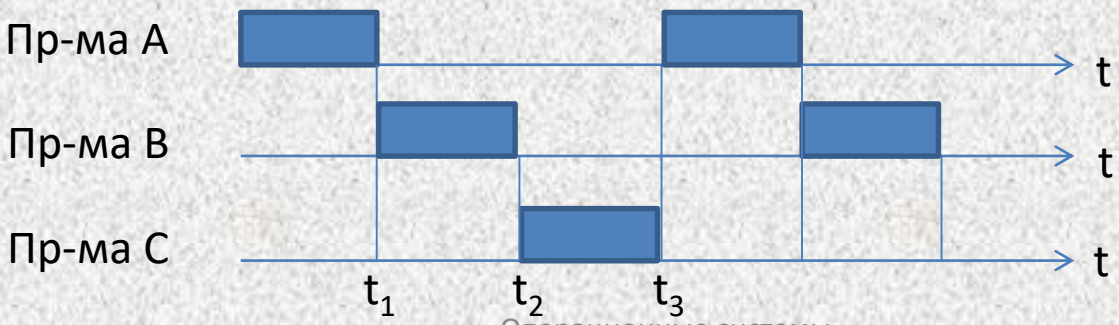
Мониторная ОС – однозадачная система, работающая в командном режиме. Каждая команда анализируется интерпретатором и исполняется этим интерпретатором или соответствующей внешней программой.

Многозадачная (мультипрограммная) ОС - система, в которой процессор ЭВМ переключается между несколькими одновременно готовыми к исполнению программами. К ним относятся пакетные ОС, ОС разделения времени, ОС реального времени, многопроцессорные и сетевые ОС. **Пакетная ОС** – система, в которой пользователь не может принимать участия в выполнении своих задач.

Классический мультипрограммный режим (кооперативная многозадачность):



Режим разделения времени - вытесняющая многозадачность:



Квант времени ЦП:

$$T_{кв} = t_2 - t_1$$

ОС реального времени – предназначены для управления задачами реального времени, в которых критичным является не только получение корректного результата, но и время получения этого результата. Используются в системах управления технологическими процессами и сложными техническими комплексами, а также в системах массового обслуживания (продажа билетов, бронирование мест в гостиницах).

Многопроцессорные ОС – используются в многопроцессорных ЭВМ, у которых одной из основных проблем является распределение задач между процессорами.

Сетевые ОС – предназначены для управления сетью ЭВМ, которая представляет собой совокупность однородных или разнородных ЭВМ, объединенных каналами связи. Задачи, решаемые сетевой ОС:

- администрирование ресурсов сети;
- формирование и обработка пакетов сообщений;
- сжатие и восстановление данных при передаче пакетов сообщений;

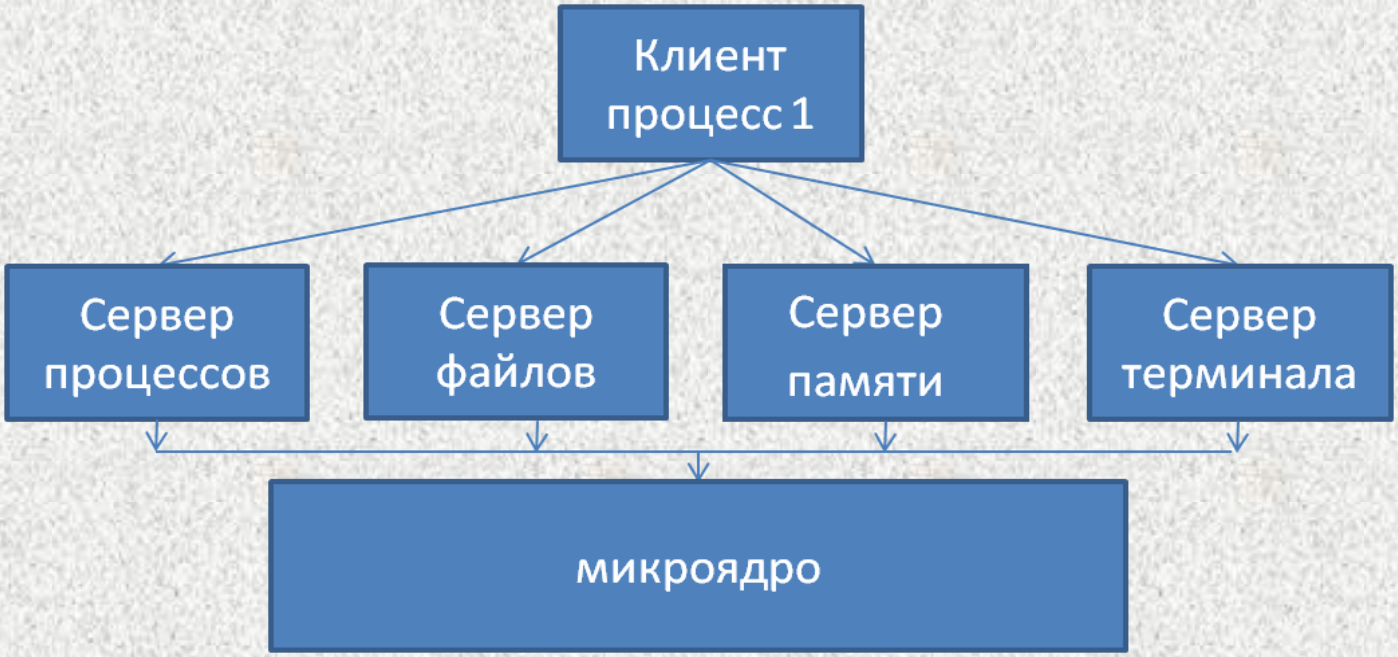
Классификация ОС по архитектуре

1. **Монолитные** - ОС написана в виде большого числа процедур, каждая из которых может вызывать другие, когда ей это нужно, т.е. каждая процедура видит все другие процедуры; скрытие полностью отсутствует; в настоящее время используются редко.

2. **Многоуровневые** – процедуры ОС организованы в виде иерархии уровней, каждый из которых может взаимодействовать только с непосредственно примыкающим к нему уровнем (семейство UNIX).

3. **Микроядерные** (клиент-серверные) – с аппаратной частью работает только микроядро, реализующее минимальный набор функций. Все остальные функции ОС реализованы как набор отдельных модулей, реализующих процессы-серверы (например, сервер процессов, сервер памяти, сервер файлов и т.д.). Например, в ОС реального времени CTOS размер микроядра составляет 4 Кб, в QNX – 8 Кб.

Архитектура микроядерных ОС



Микроядерные ОС используют технологию «клиент - сервер». Сервер – программа, которая после загрузки находится в ждущем режиме, ожидая запросов от других программ (клиентов). После получения запроса сервер выполняет определенную работу и передает результаты клиенту.

Серверные процессы в микроядерной ОС выполняются процессором в пользовательском режиме, а не в режиме ядра. Пользовательские процессы выступают в роли клиентских процессов, которые обращаются с запросами к серверным процессам.

Объекты управления ОС

Основным предметом изучения в данном курсе будут многозадачные многопользовательские ОС общего назначения, имеющие иерархическую архитектуру и работающие в режиме разделения времени.

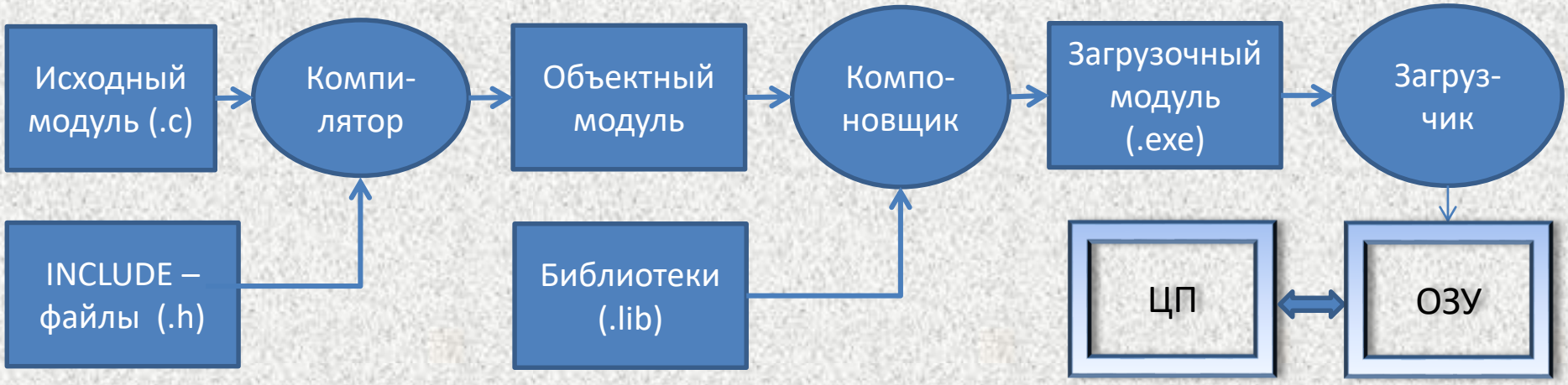
1. **Задание** – минимальная единица работы, выполняемая для одного пользователя.

2. **Процесс** – основная единица работы, требующая предоставления системных ресурсов, и которая представляет собой *объединение программы и данных*. Существует 2 типа процессов: *задачи* и *примитивы*. Задачи в любой момент времени могут прерваны и возобновлены, примитивы прерываться не могут. Программы пользователей выполняются как задачи, системные программы – как примитивы.

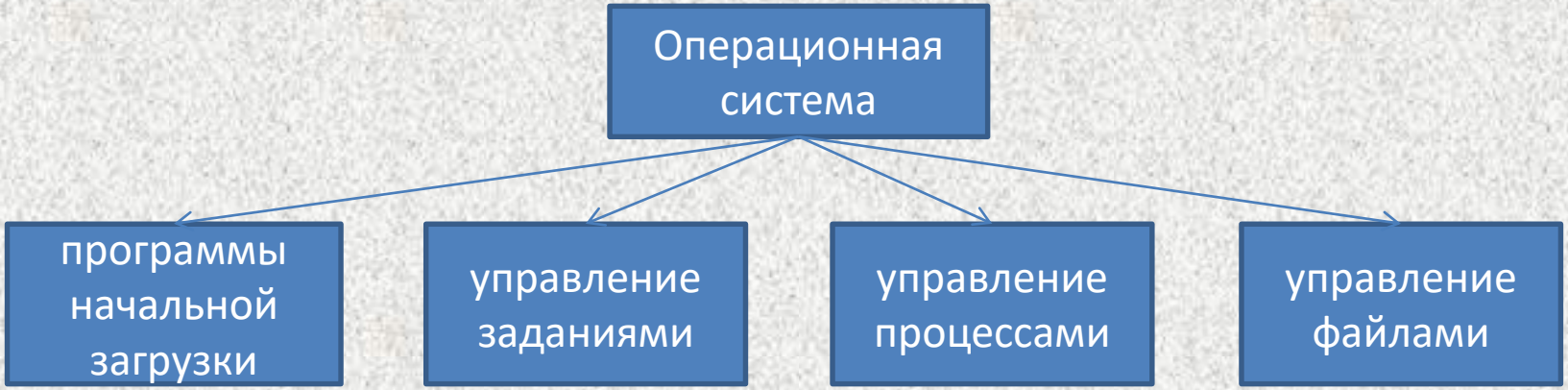
3. **Поток** - наименьшая единица работы, выполняемая в рамках процесса; является частью программы и представляет собой отдельно управляемый поименованный набор команд. В виде потоков обычно выступают функции или процедуры.

Иерархия объектов управления

Схема прохождения программы в среде ОС



Структура и основные функции типовой ОС



1.Начальное тестирование оборудования

2.Загрузка ОС

Планирование работы

1.Диспетчеризация

2. Управление памятью

3.Синхронизация

4.Обработка преры-ваний

5.Служба времени

Организация хранения и доступа к данным на ВЗУ

Основные характеристики ОС

1. **Пропускная способность** – количество заданий, выполняемых в единицу времени.

2. **Время обращения задания** – интервал времени от момента запуска задания до получения результатов.

3. **Время ответа (реакции)** - интервал времени от момента выдачи запроса на системный ресурс до момента фактического выделения этого ресурса.

4. **Доступность** – мера возможности использования системы.

5. **Безопасность** –предоставление определенных гарантий по сохранности данных пользователя от разрушения и несанкционированного доступа.

6. **Надежность**.

7. **Стоимость**.

Тема 2. Планирование работ

Иерархия планирования работ

Наименование	Объект управления	Компонент ОС
Планирование высокого уровня	Задания	Планировщик
Планирование промежуточного уровня	Процессы	Промежуточный планировщик
Планирование низкого уровня	Процессы	Диспетчер

Планирование высокого уровня для каждого задания выполняется однократно (допуск задания в систему), а диспетчеризация процессов - многократно.

Планирование высокого уровня

1. В порядке очередности поступления заданий (FIFO – First Input First Output).
2. Согласно приоритета заданий.
3. По минимуму использования ресурсов (время использования процессора, время проведения операций ввода-вывода, размер требуемой памяти).

Планирование низкого уровня (диспетчеризация)

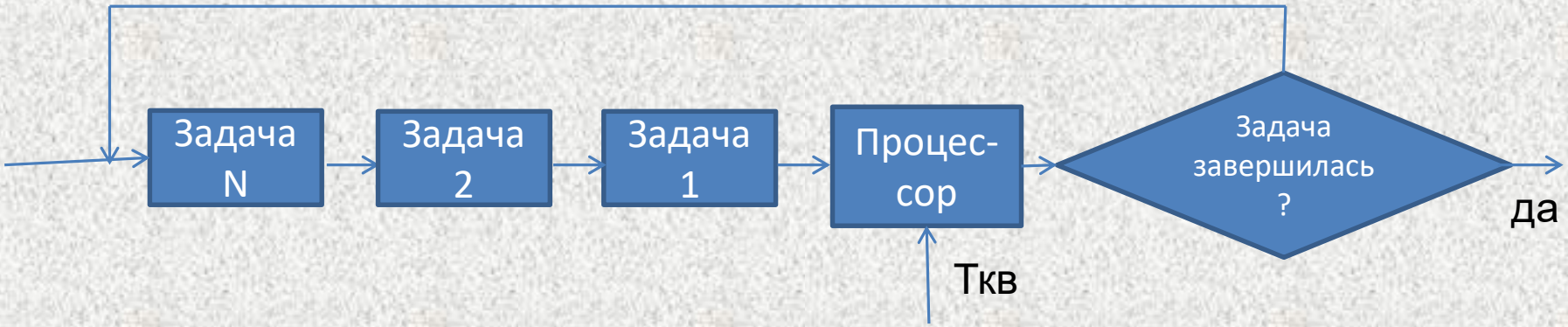
Диспетчер управляет доступом процессов к ЦП, является частью ядра ОС и запускается в следующих случаях:

- активная задача добровольно отказалась от продолжения работы;
- какая-либо задача переходит в состояние готовности, если ЦП свободен;
- время, отпущенное для исполнения активной задаче, истекло;
- требуется выполнить одну из функций ОС .

Простейшие алгоритмы диспетчирования:

- FIFO (First Input First Output) – для задач;
- LIFO (Last Input First Output) – для примитивов.

Диспетчирование: круговой циклический алгоритм



Период обслуживания задач – интервал времени, в течение которого прерванная задача повторно получает процессорное время:

Тобсл = Ткв *N, где Ткв – значение кванта времени, N – число задач в очереди.

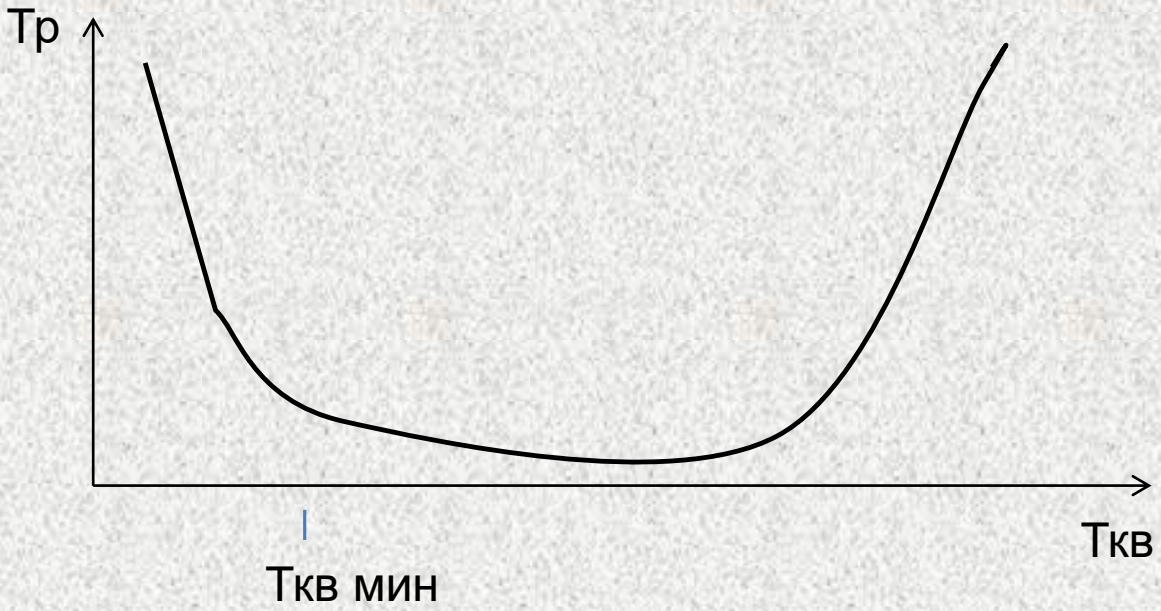
Возможные модификации алгоритма основаны на переменном значении кванта:

а) Тобсл = fix; при N ↑ значение Ткв ↓;

б) Тобсл = fix; Ткв мин = fix; при N↑ значение Ткв↓ ; при Ткв = Ткв мин прием задач в очередь прекращается.

Значение кванта вычисляется в начале цикла обслуживания очереди или после завершения очередного кванта времени.

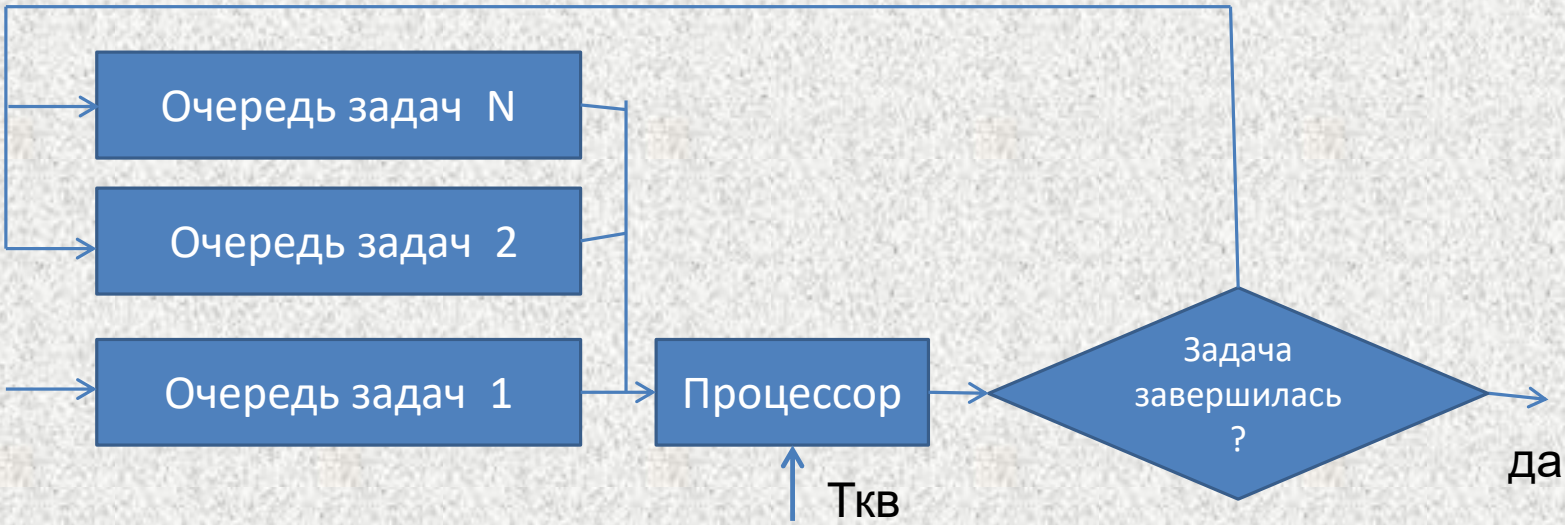
Зависимость времени реакции от величины кванта



Для исключения влияния времени переключения контекста ($T_{перекл}$) значение кванта выбирается следующим образом: $T_{кв} \approx 10 * T_{перекл}$

В современных ОС время переключения контекста не превышает 150 мкС

Диспетчирование: алгоритм обратной связи



1. Алгоритм относится к группе адаптивных алгоритмов.
2. Организуется несколько очередей задач.
3. Очередь № 1 имеет высший приоритет, N-я очередь – низший. Очереди обслуживаются в порядке приоритетов, т.е. задачи из очереди № 2 выбираются только при отсутствии задач в очереди № 1.
4. Новые задачи всегда поступают в очередь № 1. Если они завершаются в течение кванта, то выходят из системы, иначе переходят в очередь № 2 с понижением приоритета и т.д.
5. Возможна модификация, при которой новые задачи поступают в любую очередь в соответствии со своим приоритетом.

Промежуточное планирование

Управляет доступом задач в очередь диспетчера. Возможны два варианта:

- включение через фиксированные интервалы времени;
- включение при увеличении нагрузки на диспетчер (например, при уменьшении кванта до величины Ткв мин).

В последнем случае создается очередь отложенных задач, которые попадают в очередь диспетчера при снижении нагрузки.

Определение эффективности многозадачного режима

Задача 1. Имеется 3 задачи, каждой необходимо 10 сек. процессорного времени. Сколько времени потребуется для выполнения всех задач в режиме последовательного решения и в режиме мультипрограммирования с квантом 1 сек.

Решение. В режиме последовательного решения требуется

$$10 * 3 = 30 \text{ секунд}$$

В режиме мультипрограммирования требуется

$$10 * 3 = 30 \text{ секунд}$$

Определение эффективности многозадачного режима (продолжение)

Задача 2. Одновременно запускаются две задачи, каждой из которых требуется 10 сек времени ЦП и 10 сек на проведение операций ввода – вывода (т.е. 50% общего времени выполнения задачи). Сколько времени будут выполняться эти задачи в режиме последовательного исполнения и в многозадачном режиме ?

Решение. При *последовательном* исполнении для каждой задачи надо 20 сек . Всего – *40 секунд*.

В *многозадачном* режиме вероятность одновременного ввода – вывода двух задач равна: $Робщ = P_1 * P_2 = 0,5 * 0,5 = 0,25$, т.е. 25%.

Составим пропорцию: 20 сек ЦП на две задачи – 75%

X сек - 100 %

Таким образом общее время выполнения двух задач будет равно:

$20 * 100 / 75 =$ *26,6 секунд.*

Т. е. в многозадачном режиме выигрыш по времени будет равен

40 / 26,6 = 1,5 раза

Тема 3. Управление процессами

Основные функции системы управления процессами:

1. Диспетчеризация
2. Обработка прерываний
3. Синхронизация
4. Межпроцессное взаимодействие
5. Служба времени
6. Управление памятью

Классификации процессов

1. По типу:

- непрерывные;
- дискретные (старт – стопные);

2. *По месту исполнения:*

- исполняемые центральным процессором;
- процессы ввода – вывода;

3. *По степени зависимости:*

- независимые;
- зависимые.

Например, задачи – это дискретные процессы, исполняемые центральным процессором.

Характеристики процесса

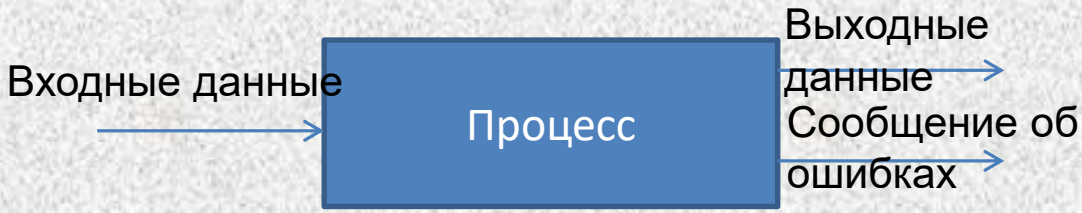
Для каждого процесса можно выделить образ и метаданные. ***Образ*** процесса называется потоком и представляет собой совокупность кода (команд программы) и данных. ***Метаданные*** – информация о процессе, которая хранится в структурах данных ОС и ресурсах, выделенных процессу.

Характеристики процесса (продолжение)

Процесс может содержать несколько потоков команд, достаточно независимых друг от друга. При этом все потоки могут пользоваться любыми ресурсами процесса и работают в пределах одного адресного пространства. Любой поток имеет доступ к любому адресу в адресном пространстве процесса и может даже стирать информацию из стека другого потока, никакой защиты здесь нет. Таким образом процесс может рассматриваться как **контейнер** потоков.

Понятие процесса удобно для группирования ресурсов, а понятие потока – для указания команд, поочередно исполняемых на ЦП.

По системным соглашениям любой процесс имеет три стандартных потока данных: 0 – ввод, 1 – вывод, 2 - диагностические сообщения. Эти имена можно использовать в командах ОС.



Многопоточные процессы

Многопоточные процессы используют:

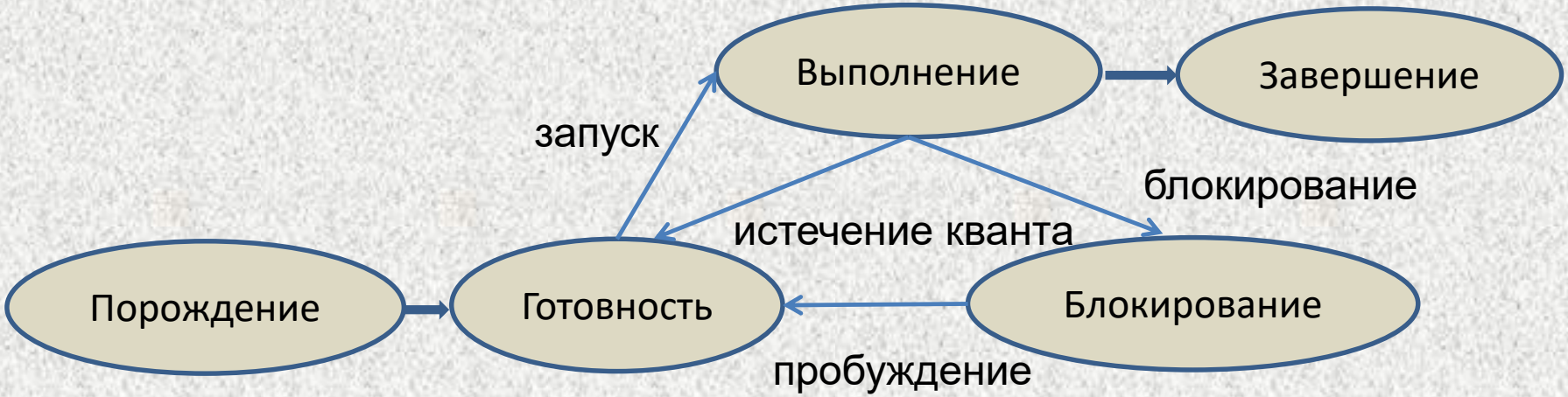
- для распараллеливания задачи на многоядерных процессорах и многопроцессорных ЭВМ;
- для более эффективного использования процессора (например, пока один поток ожидает пользовательский ввод, другой может выполнять расчеты);
- для облегчения совместного использования данных, т.к. любой поток имеет свободный доступ к данным всех потоков этого процесса

Для работы с потоком в программах используется специальный класс, например **threading.Thread**. Основные методы класса для управления потоком:

- создание;
- приостановка;
- возобновление исполнения;
- ожидание другого потока;
- уничтожение.

Диаграмма состояний процесса

Задача – это дискретный процесс, который в любой момент времени может быть прерван или возобновлён.

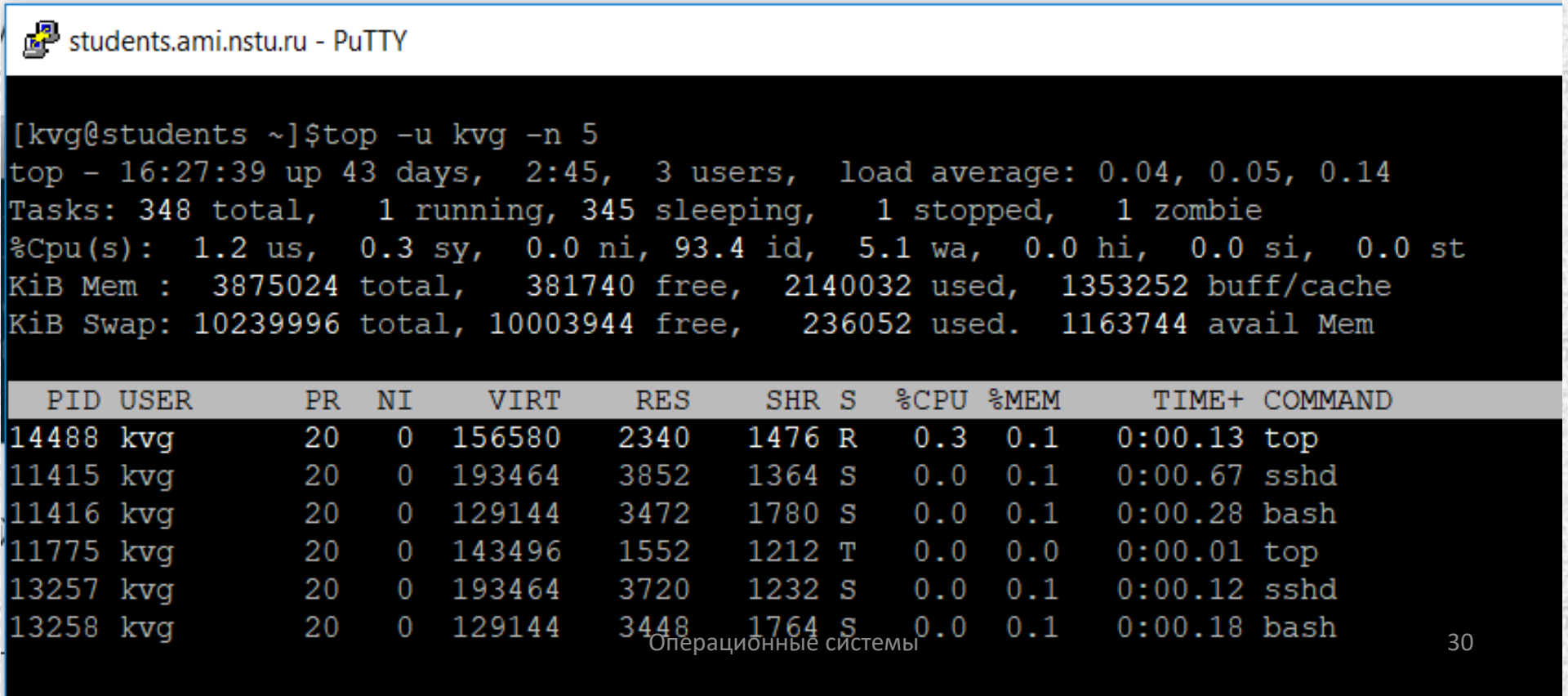


Возможные причины блокирования:

- ожидание завершения операции ввода – вывода;
- ожидание освобождения ресурса;
- ожидание завершения другой задачи для получения от нее данных для обработки;
- ожидание завершения операции страничного обмена.

Пример вывода команды top

В первой строке заголовка выводится текущее системное время, общее время работы системы, число подключенных пользователей и средняя загрузка в течение последних 1 минуты, 5 минут и 15 минут. Вторая строка содержит данные об общем числе процессов в системе, а также о числе процессов, находящихся в состоянии выполнения (running), блокирования или готовности (sleeping), завершения (stopped) и зомби (zombie). Остальные строки заголовка содержат информацию о загрузке процессора, оперативной памяти и виртуальной памяти.



Обработка прерываний

Прерывание – сигнал, вырабатываемый аппаратным или программным способом при возникновении каких – либо событий. При этом ЦП прекращает выполнение активной задачи и переключается на выполнение программы – обработчика прерывания. По окончании обработки управление может быть передано прерванной программе (если прерывание не аварийное) или в ОС.

Прерывания независимы и могут возникать одновременно. ОС должна знать очередность обработки прерываний, для чего устанавливается система приоритетов в зависимости от причины прерывания.

Классификация прерываний

В общем случае может быть 5 групп прерываний:

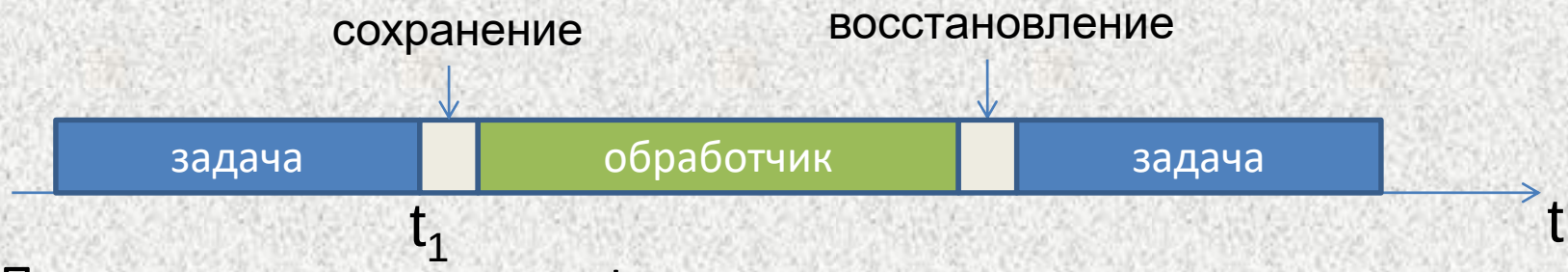
- *аппаратные* (неисправности в аппаратной части компьютера);
- *программные* (связаны с ошибками в исполняемой программе, например, деление на ноль, неправильная адресация и т.д.);
- *обращение к функциям ОС* (исполняемая программа сама обращается с запросом к ОС);
- *внешние* (сигналы таймера, с линий связи и т.д.);
- *ввода – вывода* (связаны с проведением операции ввода – вывода, например освобождение файла после записи) .

В ОС малых ЭВМ используется следующая классификация:

- *аппаратные*;
- *программные*;
- *обращение к функциям ОС*;
- *внешние*;

Максимальный приоритет устанавливается аппаратным прерываниям, минимальный – прерываниям ввода – вывода.

Алгоритм обработки прерываний



Пусть в момент времени t_1 возникает прерывание.

1. Завершается исполнение текущей команды программы.
2. Запоминается адрес следующей команды программы.
3. Сохраняется состояние прерванной задачи в специально выделенном участке ОЗУ.
4. Процессор переключается на выполнение обработчика прерывания.
5. Если прерывание не аварийное, то восстанавливается состояние задачи на момент прерывания, управление передается на адрес команды, сохраненный на шаге 2 и продолжается исполнение команд. Иначе управление возвращается в ОС.

Обработка прерываний: векторная обработка

Используется в малых ЭВМ.

В ОЗУ создается таблица для хранения адресов программ – обработчиков прерываний. Эти адреса называются векторами.

Номер прерывания	0	1	2	3	4	5 …
Адрес обработчика	A_0	A_1	A_2	A_3	A_4	A_5 …

Для уменьшения размера таблицы часто используется свойство постоянства длины адреса (в современных ОС адрес обычно занимает 4 байта). В MS Windows таблица прерываний располагается в ОЗУ с адреса 00 00 00 00, занимает 1 Кбайт и выглядит так:

A_0	A_1	A_2	A_3	A_4	A_5 …

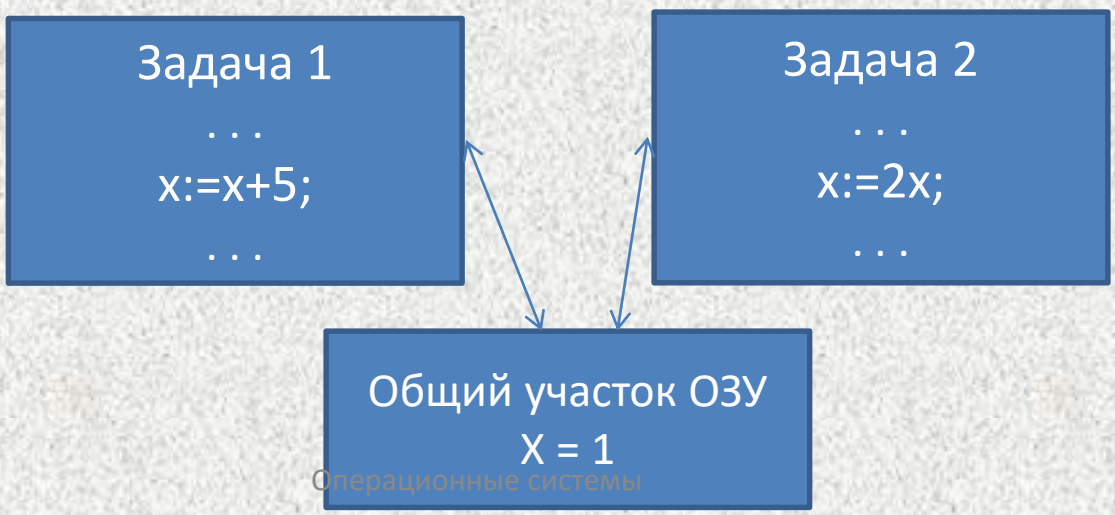
Вопрос: сколько прерываний знает Windows ?

Синхронизация процессов

Синхронизация – предотвращение конфликтных ситуаций между процессами при получении системных ресурсов.

Проблема 1. Обеспечение детерминированности процессов. При неоднократных запусках одной задачи необходимо получить одинаковый результат. Детерминированность может быть нарушена для зависимых процессов.

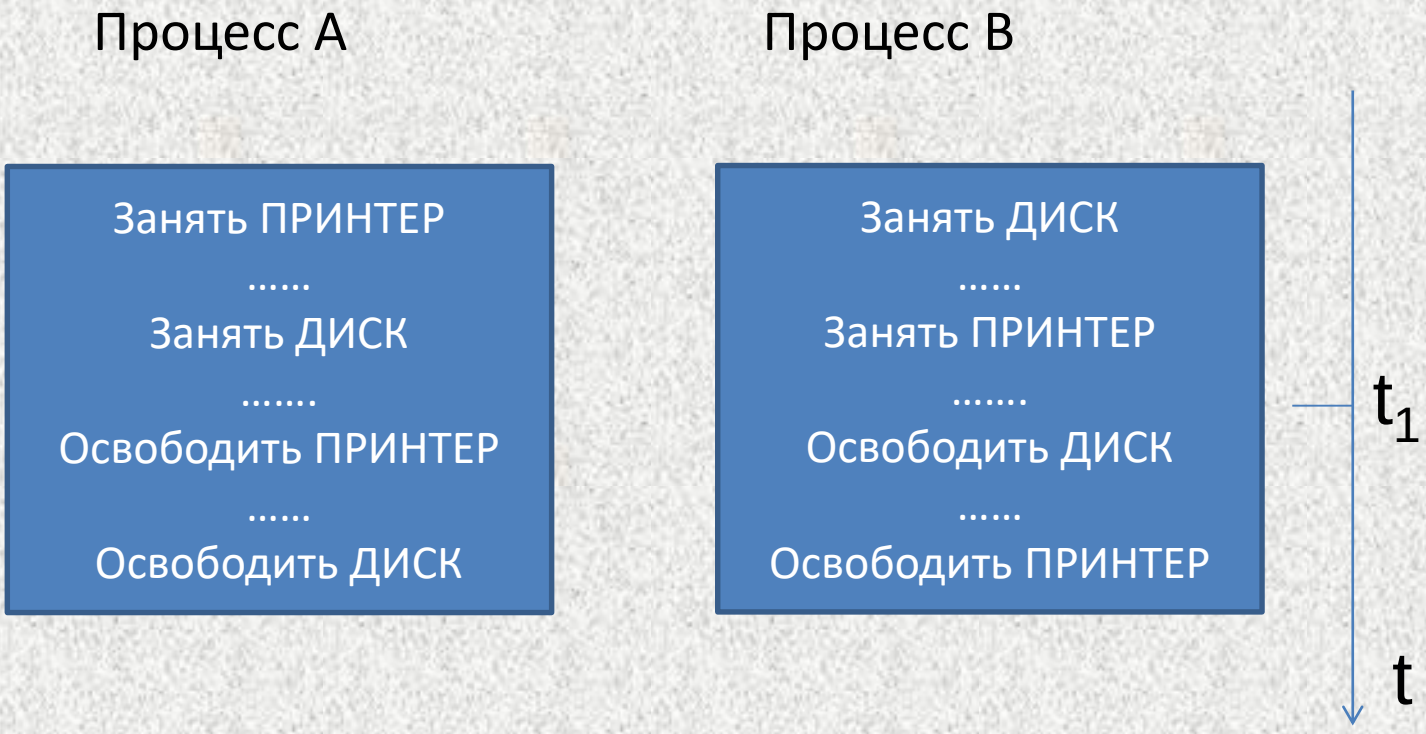
Пример: доступ к общему участку памяти двух задач. Какое значение будет иметь переменная X после завершения этих задач, если до начала их исполнения ее значение равно 1 ?



Синхронизация процессов

Проблема 2. Исключение взаимных блокировок (тупиков, клинчей, дедлоков).

В каких состояниях будут находиться процессы А и В в момент времени t_1 ?



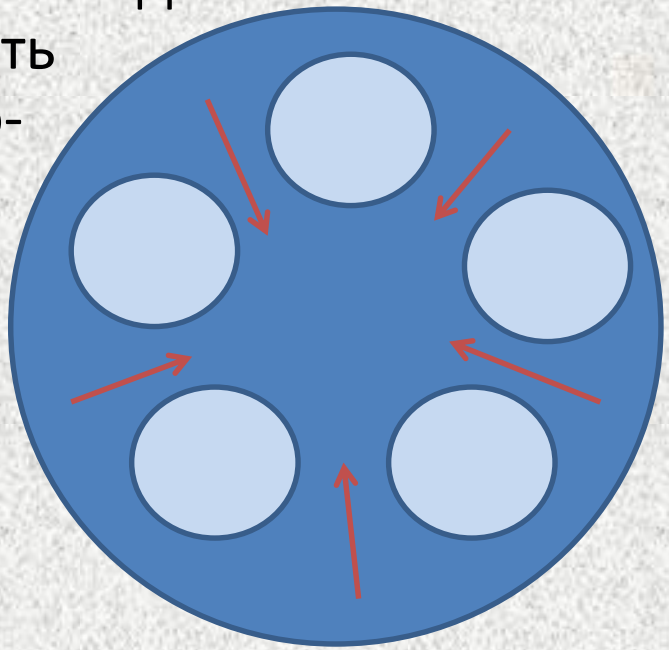
Ситуация, при которой низкоприоритетный процесс блокирует высокоприоритетный процесс, называется инверсией приоритетов

Синхронизация процессов

Проблема 3. Доступ к ресурсам (проблема обедающих философов).

Пять философов сидят за круглым столом, у каждого есть тарелка спагетти, которые можно есть только двумя вилками.

Между каждыми двумя тарелками лежит одна вилка. Когда философ голоден, он старается взять две вилки. Если это удается, то он некоторое время обедает, затем кладет обе вилки и продолжает думать. Если не удается, то он остается голодным и продолжает думать.



Задача: как не оставить философов голодными ? Эта задача моделирует борьбу процессов за исключительный доступ к ограниченному числу ресурсов .

Синхронизация процессов

Проблема 4. Доступ к данным (проблема читателей и писателей)

Пример: доступ к базе данных для бронирования ж/д билетов.

1. Несколько процессов могут одновременно читать данные из базы.
2. Если какой-то процесс записывает данные, то ни один процесс не может читать или записывать данные.

Если отдать приоритет «читателям», то процессы-»писатели» могут вообще не получить доступ к БД.

Решение: а) новые «читатели» ставятся в очередь после писателей;

б) процессам- «писателям» присваивать более высокий приоритет.

Синхронизация процессов

Проблемы, требующие синхронизации:

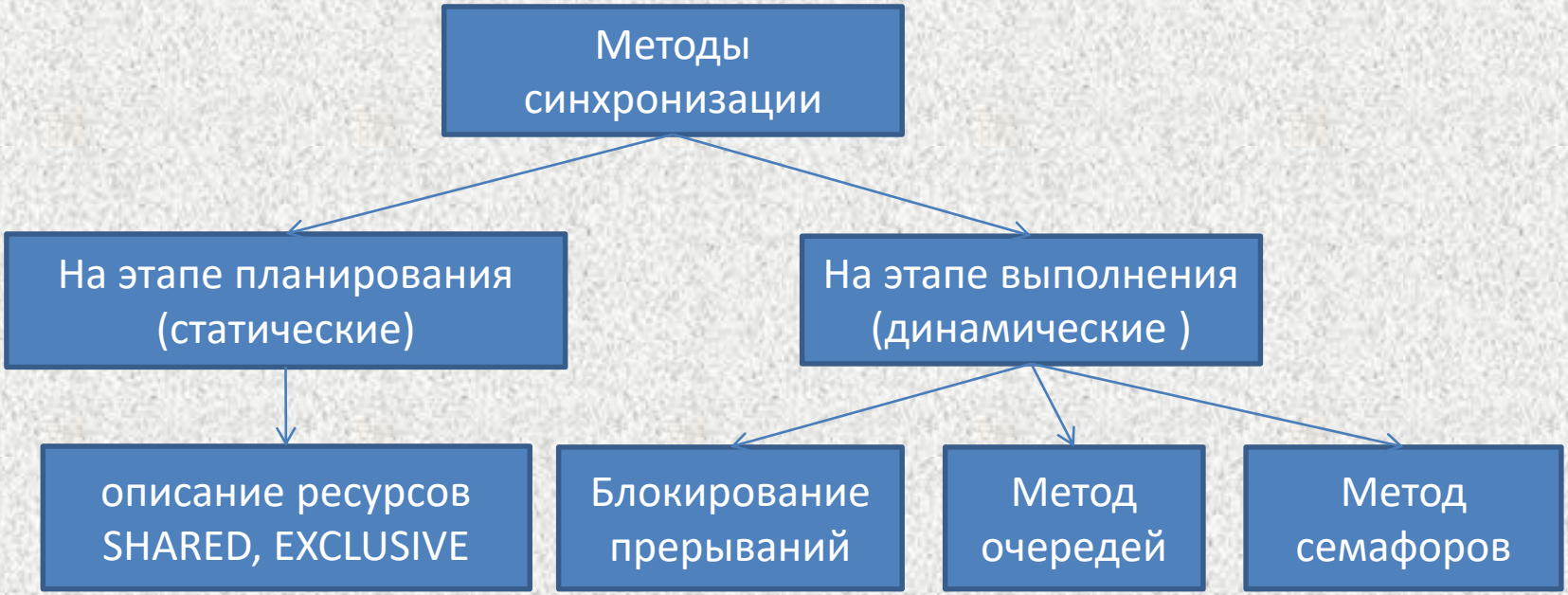
- обеспечение детерминированности процессов;
- исключение взаимных блокировок (тупиков, клинчей, дедлоков);
- доступ к ресурсам;
- доступ к данным;

Определение. Части процессов (точнее, потоков), работающие с общими ресурсами, называются критической секцией (критическим интервалом). Синхронизации необходимо подвергать именно эти секции.

Зависимые процессы могут находиться в соотношении «производитель – потребитель ресурса» или использовать ранее созданный ресурс. Для первой категории процессов единственный способ синхронизации – обеспечить определенный порядок запуска процессов (сначала производитель ресурса, затем потребитель).

Для второй категории возможности для синхронизации представлены на следующем слайде.

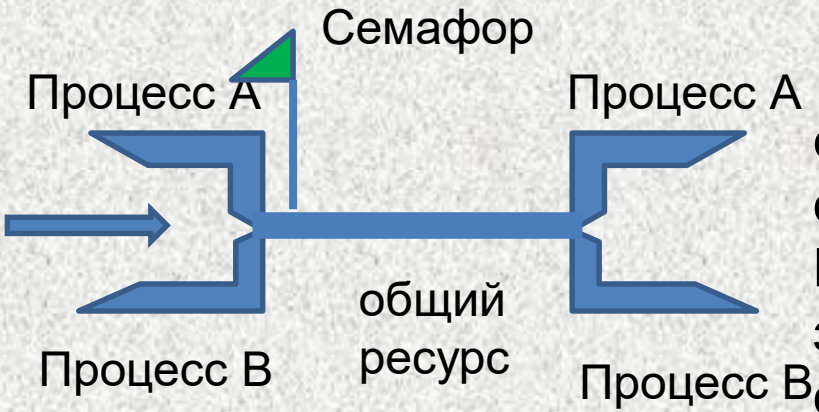
Методы синхронизации процессов



Блокирование прерываний может использоваться только для коротких критических секций, т.к. при этом механизм прерываний отключается и все процессы в системе приостанавливаются. Критическая секция выполняется как примитив.

Метод очередей используется, когда в качестве общего ресурса выступает другая задача (например, диспетчер печати). Здесь создается очередь заявок на обслуживание.

Метод семафоров



Каждый общий ресурс снабжается **сема-фором**. При входе в критическую секцию процесс проверяет семафор. Если он от-крыт, то процесс его закрывает и входит в критическую секцию, иначе процесс блоки-руется до открытия семафора. После про-хождения секции процесс должен открыть семафор и освободить ресурс.

Семафоры могут иметь счетчик, задающий количество процессов, имеющих одновременный доступ к ресурсу. Семафор со счетчиком, равным единице, называется **мутексом** (обеспечивает исключительный доступ к ресурсу).

Достоинство метода – блокируются только процессы, которые связаны с текущим процессом общим ресурсом, запросы на который возникают одновременно. Недостаток – возможность семафорного конфликта, когда процесс, выполняющийся в критической секции, прерывается и заверша-ется аварийно, а семафор остался закрытым.

Барьеры

Предназначены для синхронизации **группы потоков или процессов.** В некоторых случаях необходимо организовать такую обработку данных, когда ни один из процессов не может перейти к следующей фазе исполнения до тех пор, пока все процессы не будут готовы перейти к следующей фазе. Для этого в конце каждой фазы ставиться **барьер**, при достижении которого процесс должен остановиться и ждать, пока все процессы не достигнут этого барьера.

Обычно используются при организации сложных многопоточных вычислений. Например, матрицы большой размерности можно разбить на части и вычислять каждую в отдельном потоке с использованием многопроцессорных ЭВМ. При этом переход к следующей итерации возможен только тогда, когда завершится вычисление всех частей матрицы.

Средства межпроцессного взаимодействия

Обмен данными между процессами возможен следующими способами:

1. Системный буфер (clipboard);
2. Каналы;
3. Сокеты;
4. Разделяемая память;
5. Очередь сообщений.
6. Сигналы.

Системный буфер - часть ОЗУ, которая используется для обмена данными под управлением ОС (команды Ctrl+C и Ctrl+V).

Канал (конвейер, pipe) использует принцип обмена на основе функций управления файлами. Имеет ограничения по размеру (64Кбайт). Каналы могут быть поименованными.

Сокет – программный интерфейс для обмена данными между процессами, исполняемыми как на одной, так и на разных ЭВМ, объединенных в сеть. Для сокета обычно задается IP-адрес ЭВМ и номер порта, к которому он подключен. Этот номер используется протоколом TCP/IP.

Средства межпроцессного взаимодействия (продолжение)

Разделяемая память (общая) – используется для хранения глобальных переменных, доступных нескольким процессам.

Очередь сообщений – использует модель «много отправителей – один получатель». Процесс-получатель является владельцем очереди, дисциплины получения сообщений могут быть различными (FIFO, LIFO, по приоритету, по типу сообщений и т.д.).

Сигналы - не блокирующий отправителя способ взаимодействия между процессами, имеют размер 5 байтов (1 байт кода и 4 байта данных). Процесс - адресат может определить три варианта поведения при получении сигнала:

- использовать обработчик по умолчанию (обычно уничтожение процесса);
- игнорировать сигнал — для этого атрибуту процесса, называемому сигнальной маской, задается определенное значение. Существуют сигналы, которые не могут быть проигнорированы, например - SIGKILL (например, в ОС Linux команда kill -9 PID);
- зарегистрировать собственный обработчик - собственную функцию, которая будет вызвана при получении сигнала.

Тема 4.Файловая система

Часть 1. Хранение и доступ к данным

Общие сведения

Термин «файловая система» имеет два значения:

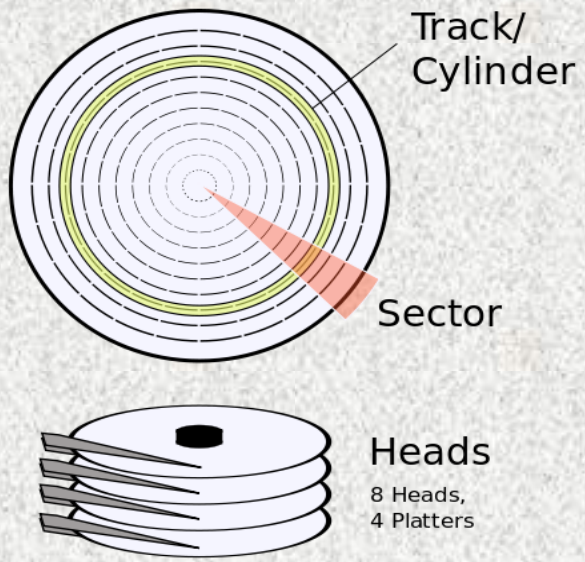
- способ хранения и организации доступа к информации, хранящейся на внешнем запоминающем устройстве (ВЗУ);
- компонент операционной системы, реализующий весь комплекс действий по работе с информацией, хранящейся на ВЗУ (распределение внешней памяти, контроль прав доступа, запись, удаление и т.д.)

В этой части курса будем рассматривать файловую систему как способ хранения и организации доступа.

Файл – поименованная совокупность данных на ВЗУ. При записи для каждого файла совместно с основным потоком данных создается и сохраняется набор метаданных (имя, атрибуты, размер, дата и время создания или последнего изменения, адрес и т.д.).

Файлы могут быть исполняемыми и неисполняемыми. Исполняемый файл содержит команды для центрального процессора (программа) или для ОС (командный файл, сценарий). Неисполняемые файлы содержат данные, обрабатываемые различными программами.

Физическая модель внешней памяти



Дорожка- концентрическая окружность на магнитной *поверхности*.

Сектор - участок дорожки МД, хранящий минимальную порцию информации, которая может быть считана или записана за одно обращение к диску.

Цилиндр – совокупность *д*орожек с одинаковыми номерами на различных *поверхностях* диска

Дорожки нумеруются в пределах поверхности, а сектора – в пределах дорожки. Стандартный размер сектора - 512 байт (0,5 Кбайт).

Для получения доступа к информации, хранящейся на диске, необходимо указать *физический адрес*, который включает номер цилиндра, номер головки и номер сектора *(CHS).*

Физическая модель используется контроллером диска.

Конструкция жесткого диска

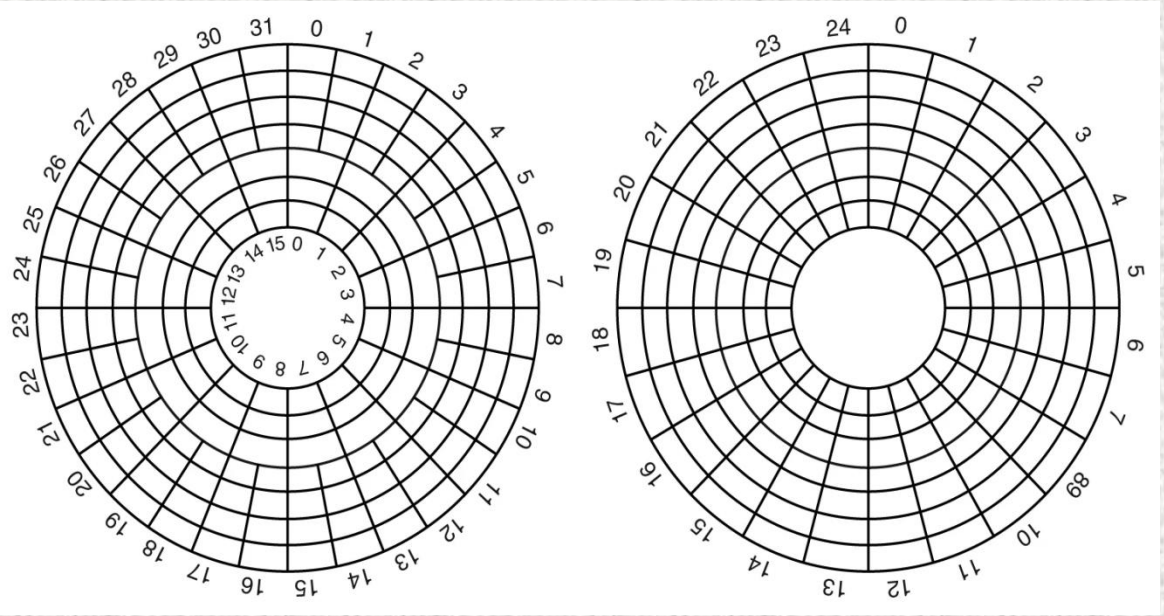
Схема жесткого диска

Фото жесткого диска Samsung

Разбиение дорожек на сектора

Возможны два способа разбивки дорожек на сектора – с постоянным количеством секторов на дорожке и с переменным количеством секторов на дорожке. В первом способе каждая дорожка диска имеет одинаковое число секторов, что обеспечивается **изменением плотности записи** при переходе с одной дорожки на другую. Максимальная плотность записи используется на дорожках с минимальным радиусом. Для получения доступа к произвольному сектору диска необходимо указать его *физический адрес (CHS).* Такой способ адресации использовался только для дисков небольшого объема (до 500 Мбайт).

Современные диски большого объема используют второй способ, при котором дорожки на каждой поверхности диска группируются в зоны. Все дорожки в одной зоне имеют одинаковое количество секторов, дорожки в зонах с меньшим радиусом имеют меньше секторов, чем зоны с большим радиусом, а плотность записи остается **постоянной**.



Т.е. без изменения технологии производства увеличивается общее число секторов на диске и его объем. При этом используется линейная адресация всех секторов диска (LBA). В настоящее время для задания LBA-номера сектора используется 6 байтов, что даёт возможность адресовать на диске 2^{48} секторов

Разбиение диска на разделы

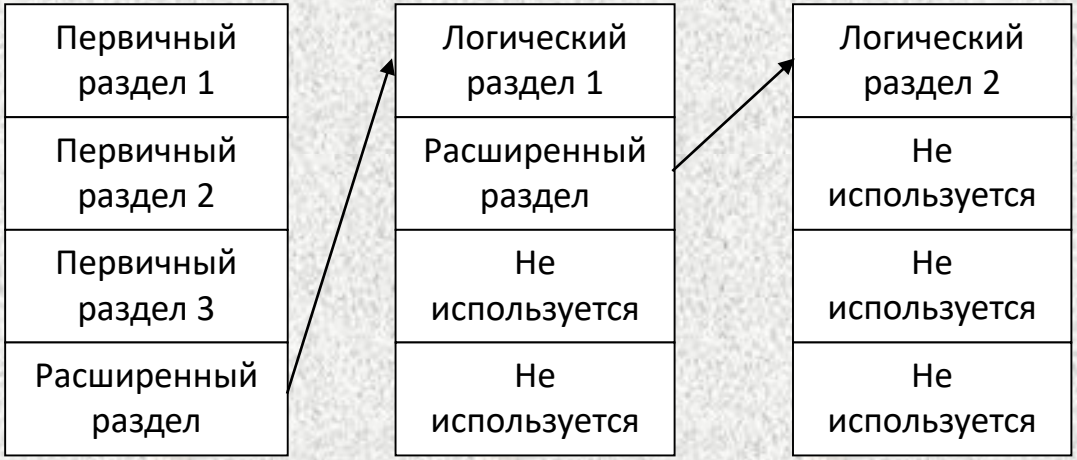
Все линейное дисковое пространство обычно делится на несколько разделов. Раздел – это часть диска, имеющая собственную файловую систему, в один раздел объединяется группа смежных блоков. Для каждого раздела на диске необходимо хранить информацию о его начале и конце (номера начального и конечного секторов).

Достоинства:

- увеличивается скорость выполнения операций чтения и записи;
- появляется возможность структурирования данных (например, можно отделить файлы пользователя от файлов ОС);
- на одном диске можно установить несколько ОС

Информация о разбиении диска хранится в главной загрузочной записи диска (MBR) в виде **таблицы разделов**, содержащей четыре записи по 16 байтов. Каждая запись может хранить информацию по одному разделу, который называется первичным.

Разбиение диска на разделы (продолжение)



Для увеличения числа разделов диска один из первичных разделов объявляется расширенным и в нем создаются логические разделы. Расширенный раздел в таблице может быть только один.

Расширенные разделы не используются для хранения данных, они могут лишь хранить информацию о логических разделах. Каждый расширенный раздел имеет свою таблицу разделов, в которой используются только две записи, задающие один логический и один расширенный, то есть получается цепочка из таблиц разделов.

Каждый раздел имеет собственное имя, например, в Windows – c:, d:, e:, в Linux – hda1, hda2 (диски с интерфейсом IDE или Parallel ATA, PATA) или sda1, sda2 (диски с интерфейсом Serial ATA,SATA).

В последнее время начинает все чаще использоваться таблица GPT (GUID Partition Table, таблица с глобальными идентификаторами). Если диск имеет таблицу разбиения GPT, в на нем по умолчанию зарезервировано место под 128 разделов, каждый из которых является первичным.

Пример разбиения на разделы

Раздел	Начало	Размер	Тип
C	0	200	первичный
D	200	200	первичный
E	400	200	первичный
	600	400	расширенный

Здесь для упрощения размер физического диска принят равным 1000 секторов.

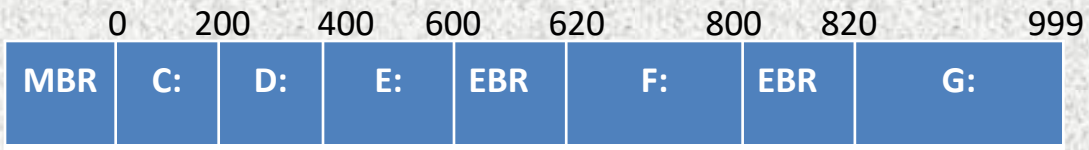
Расширенная загрузочная запись раздела F:

Раздел	Начало	Размер	Тип
F	620	180	логический
	800	200	расширенный

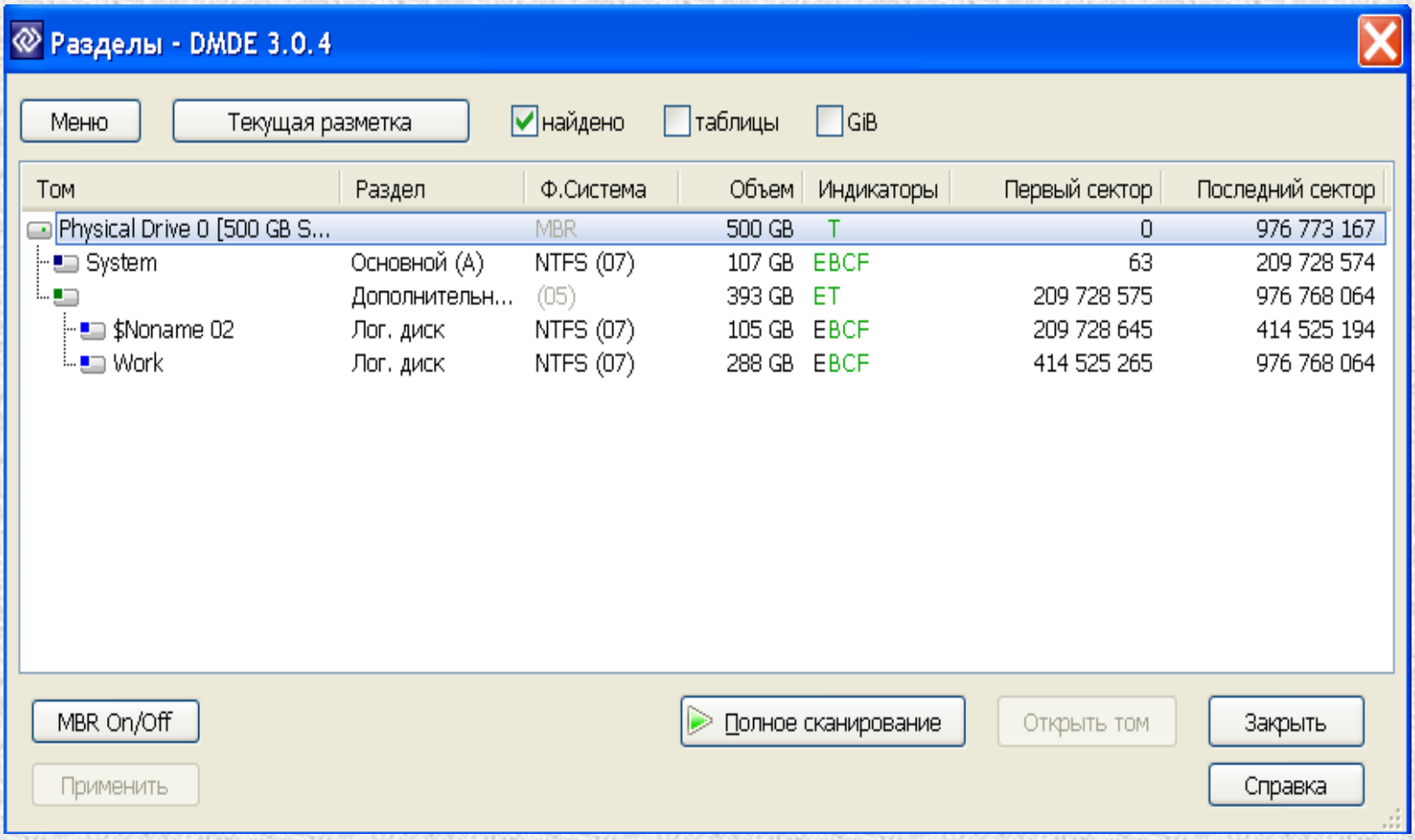
Расширенная загрузочная запись раздела G:

Раздел	Начало	Размер	Тип
G	820	180	логический

Общая структура физического диска



Пример разбиения диска на разделы



Здесь один первичный, один расширенный и два логических раздела

Объединение дисков

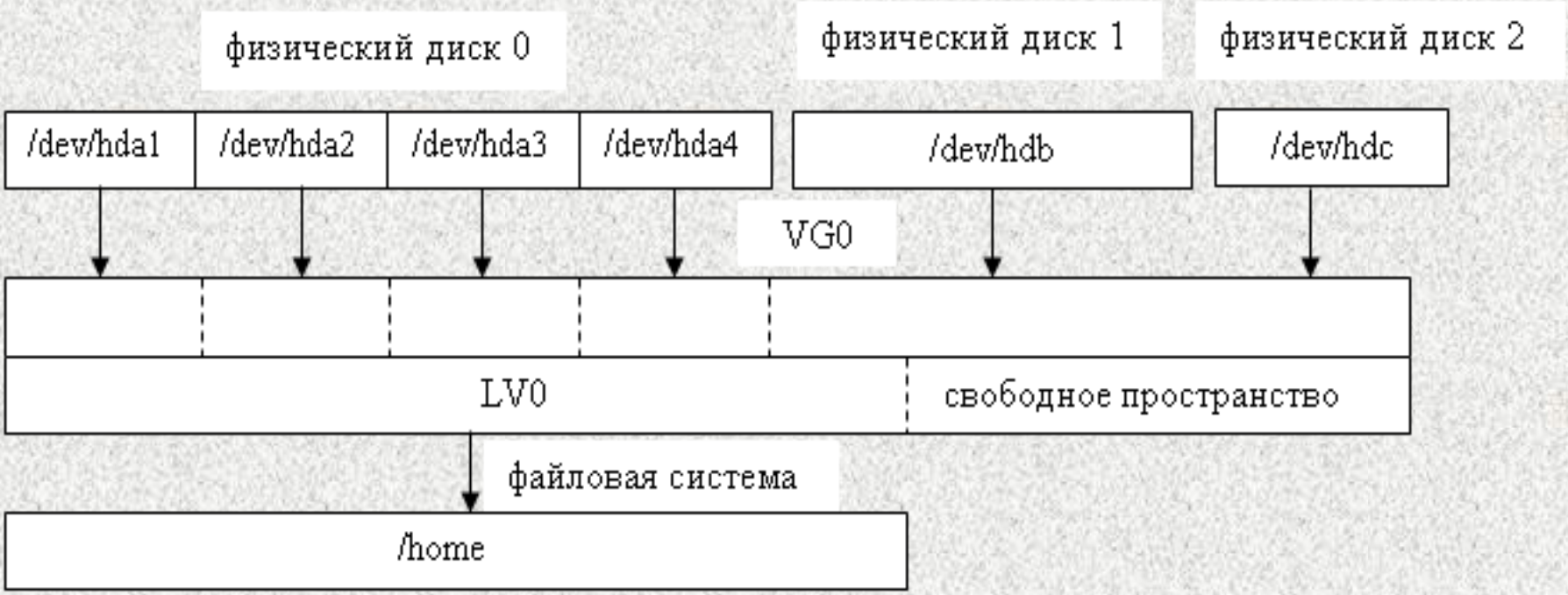
Дисковые разделы могут создаваться не только для разбиения, но и для объединения физических дисков. При этом на общем непрерывном дисковом пространстве может быть создана единая файловая система. Способы объединения : а) организация RAID массивов (redundant array of inexpensive disks, избыточный (резервный) массив недорогих дисков), б) применение менеджера логических томов.

Технология RAID используется для избыточности и повышения производительности систем хранения данных. Например, RAID уровня 1 предназначен для увеличения надежности хранения данных и реализуется путем зеркалирования (дублирования) дисков.

Менеджер логических томов (Logic Volume Manager, LVM) – компонент ОС, позволяющий:

- использовать разные области одного жёсткого диска и области различных жёстких дисков как один логический том;
- иметь файловую систему, которая превышает размер наибольшего диска;
- добавлять диски или разделы в дисковую группу и расширять существующие файловые системы «на лету»;

Пример использования LVM



Здесь 4 раздела физического диска hda и два физических диска (hdb и hdc) отображаются в группу томов VG0, в которой создан логический том LV0 и оставлено свободное место для других логических томов или для последующего роста LV0.

LVM может использоваться для разбиения на разделы одного физического диска как альтернатива классического метода разбиения с помощью расширенных разделов.

Структура дисковой памяти сервера students.ami.nstu.ru

Диск	Раздел	Тип	Размер (Гб)	Тип ФС	Точка монтир.	Логическое имя раздела
sda 127 Гб	sda1	первичный	0.5	xfs	/boot	sda1
	sda2	первичный под LVM	97,7	LVM2 member		sda2
	dm-0	логический том	9,8	swap	[SWAP]	centos-swap
	dm-1	логический том	39,1	ext4	/	centos-root
	dm-2	логический том	9,8	ext4	/tmp	centos-tmp
	dm-3	логический том	39,1	ext4	/home	centos-home
	свобо-ден		30			

Логическая модель внешней памяти

С логической точки зрения адресное пространство раздела диска представляет собой набор последовательно пронумерованных **секторов**. Небольшая часть секторов выделяется для хранения слу-жебной информации (**системная область**), а остальные сектора предназначены для хранения файлов и каталогов и образуют **область данных**.

Минимальной единицей дисковой памяти является **блок** (кластер), содержащий несколько секторов. Размер блока, а также размещение системной области (в смежных или несмежных блоках), определяется файловой системой. Например, в Linux размер блока обычно равен 1 Кбайт, а в Windows – 4 Кбайт.

С логической моделью диска работает операционная система.

Методы размещения файлов

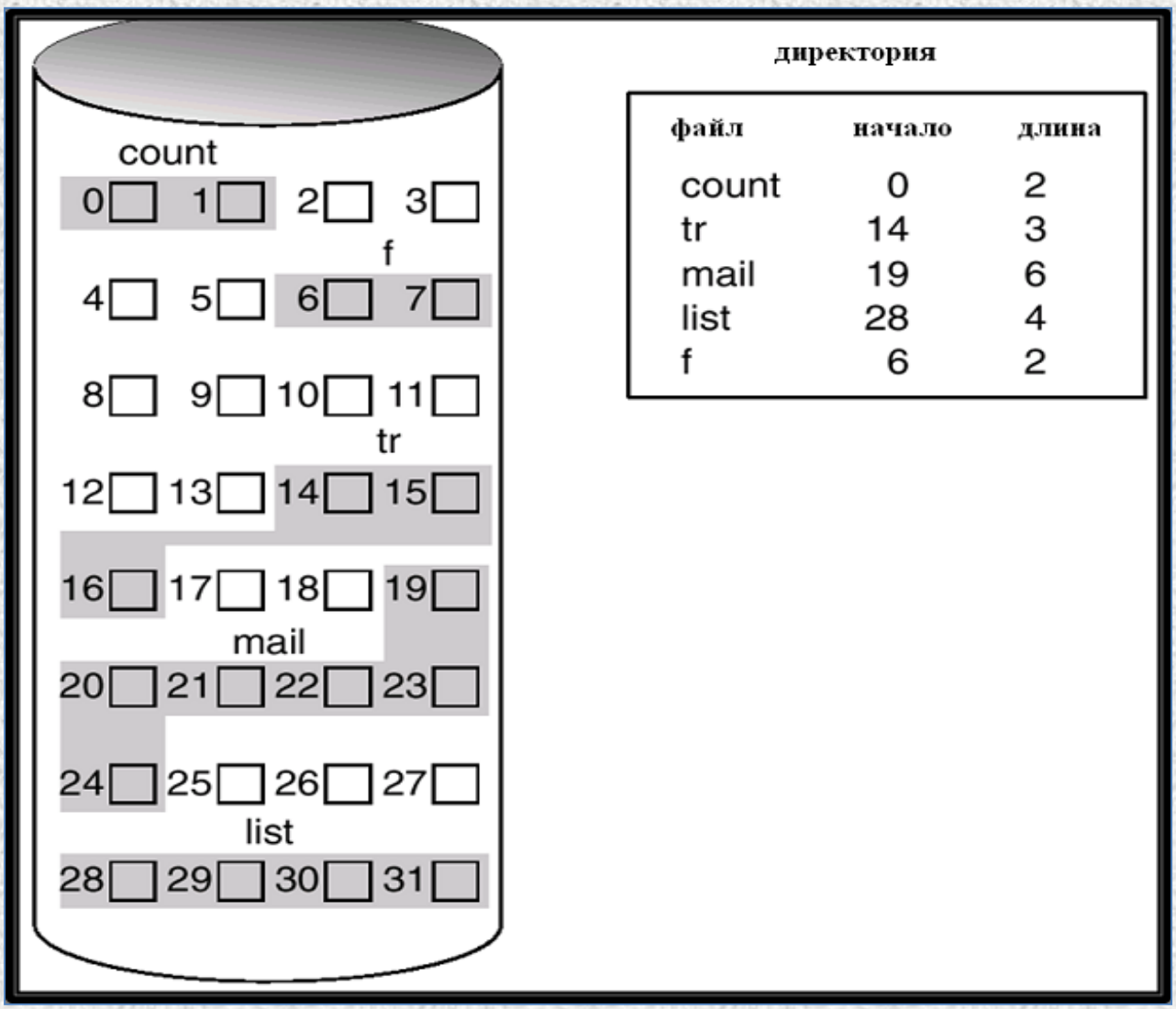
Возможны три метода размещения блоков файла на диске:

- смежное размещение;
- ссылочное размещение;
- индексированное размещение.

Смежное размещение.

1. Каждый файл занимает набор смежных блоков на диске.
2. Достоинство – простота, т.к. в каталоге требуется хранить только одну ссылку (номер блока) и длину (число блоков).
3. Недостатки - невозможность увеличить размер файла и проблема фрагментации.
4. Возможна модификация – файлы с применением расширений: дисковые блоки размещаются в расширениях (extents). Расширение – это набор смежных блоков на диске, файл состоит из одного или нескольких расширений.
5. Способ использовался в старых ОС.

Смежное размещение файлов

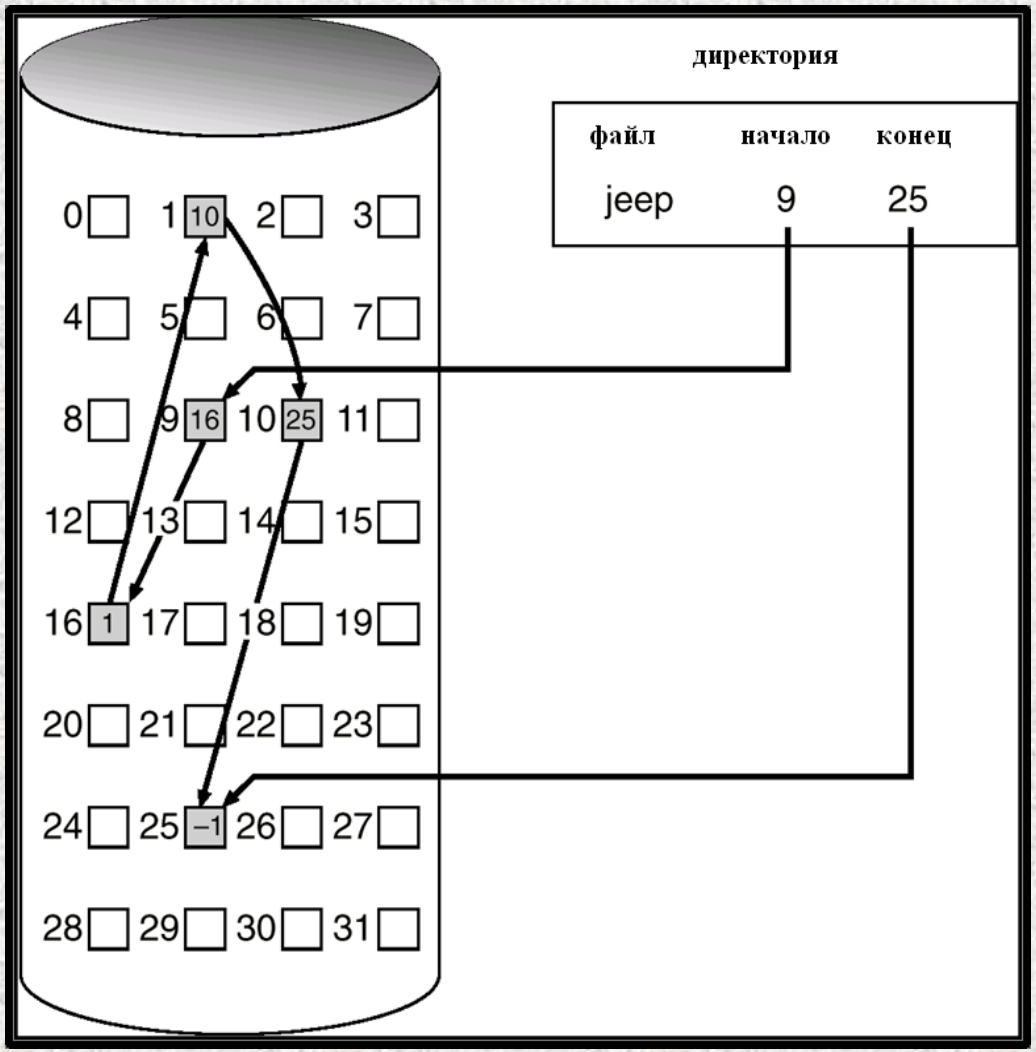


директория

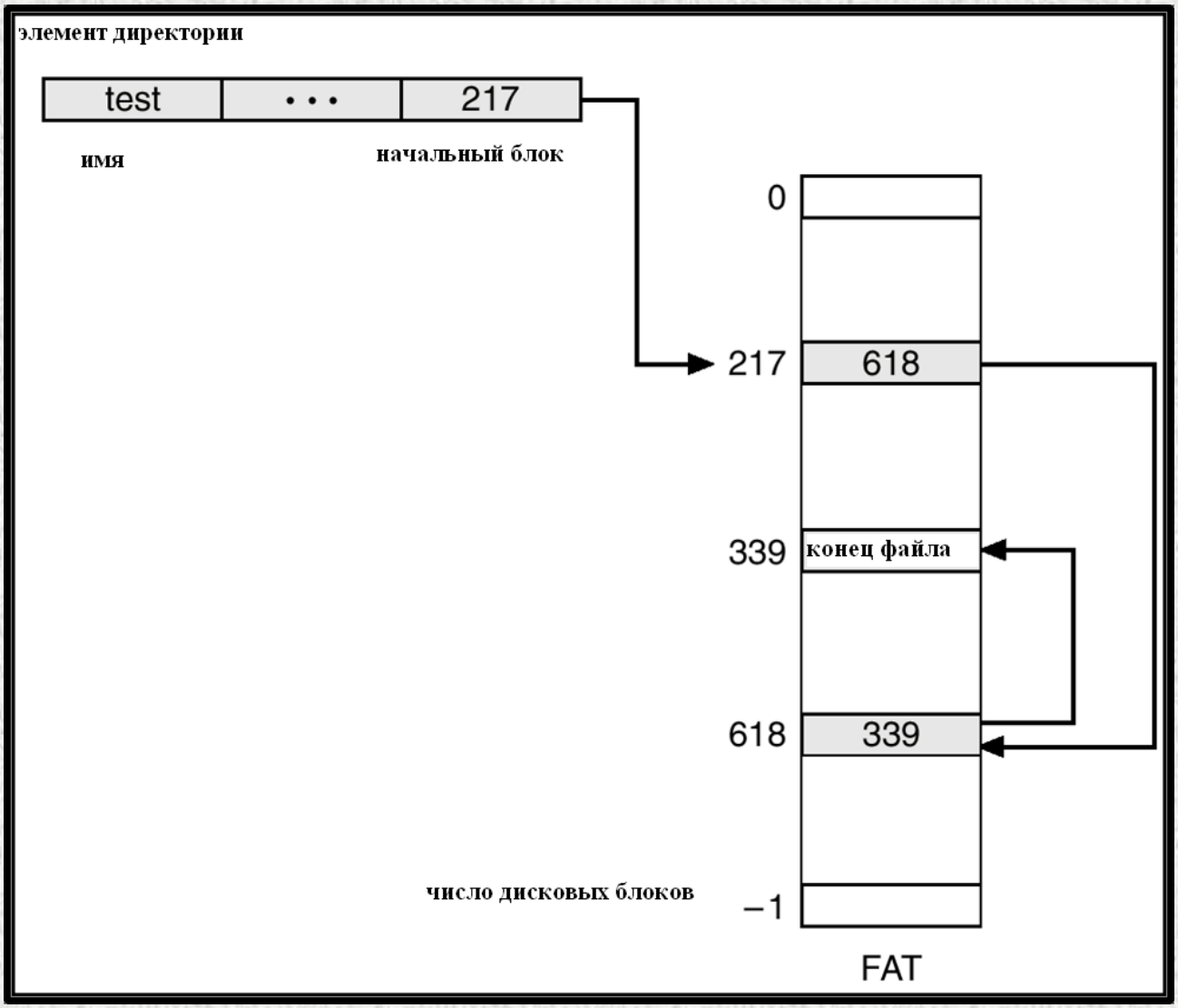
файл	начало	длина
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Ссылочное размещение файла

1. Каждый файл представлен в виде связанного списка дисковых блоков, которые могут быть разбросаны по диску.
2. Элементы списка могут размещаться в самих дисковых блоках или в отдельной специальной таблице размещения файлов.
3. Используется в MS DOS, MS Windows, OS/2 (файловая система FAT).
4. Достоинство – простота, т.к. необходимо хранить только начальный адрес и длину файла.



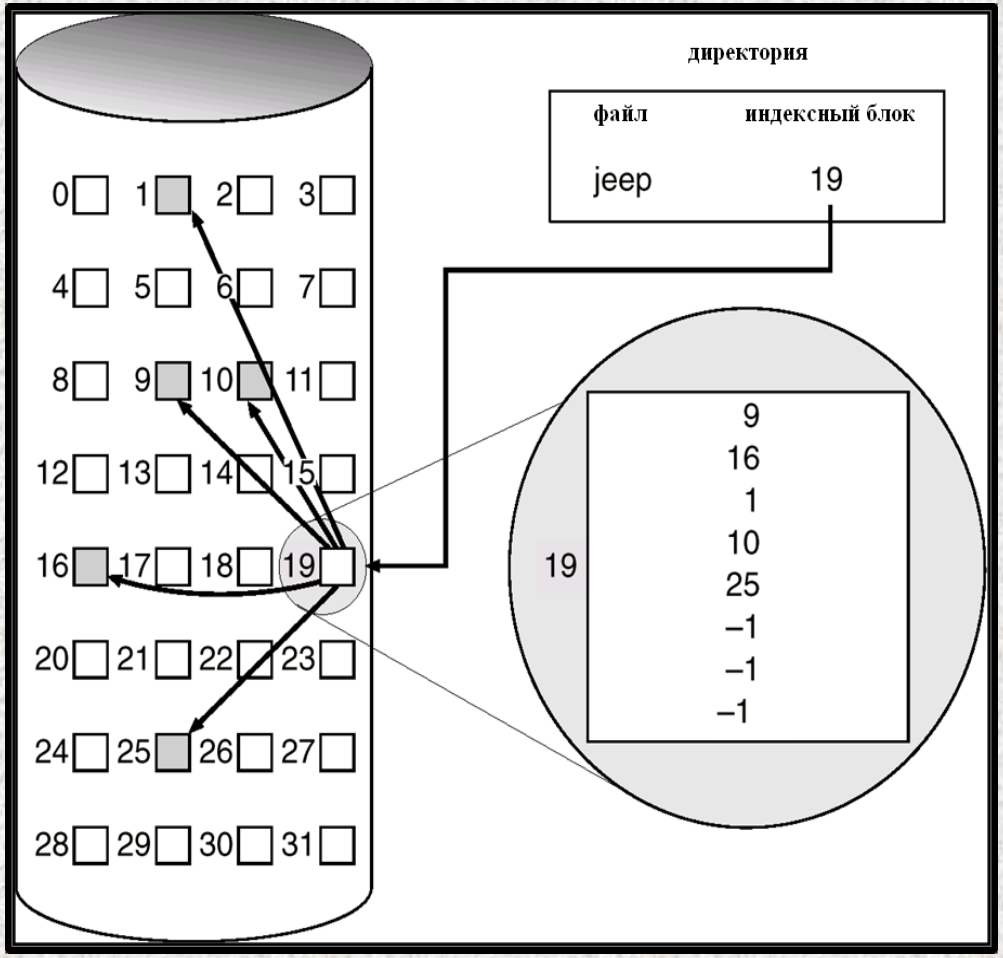
Использование таблицы размещения файлов



Здесь каждому блоку диска соответствует один элемент в табли-це FAT.

Индексированное размещение

1. Все указатели собраны вместе в индексный блок.
2. Используется индексная таблица, ссылающаяся на блоки данных файла.
3. Динамический доступ без внешней фрагментации, но ухудшается использование дисковой памяти, т.к. для каждого файла необходимо выделять индексный блок.
5. Используется в семействе ОС UNIX и в ОС Windows (NTFS).
6. Достоинство - уменьшение времени доступа.



Реализация файловых систем

1. Файловые системы ОС Linux

Особенности файловых систем Linux:

- отсутствует понятие логического диска;
- все файлы и каталоги физического диска образуют единое иерархическое дерево;
- под файлом понимается не только поименованная совокупность информации на ВЗУ, но и любое **устройство**, которое может хранить, поставлять или потреблять информацию. В этом случае устройство подключается (монтируется) к существующему дереву файловой системы в указанной пользователем точке.
- типы файлов: *обычные, каталоги, специальные (блочные и символьные), ссылки, каналы.*

Специальные блочные файлы соответствуют устройствам, на которые запись и считывание информации проводится блоками (например, ВЗУ). Символьные файлы соответствуют устройствам, взаимодействие с которыми производится посимвольно в режиме потока байтов (например, терминал).

Права доступа в ОС Linux

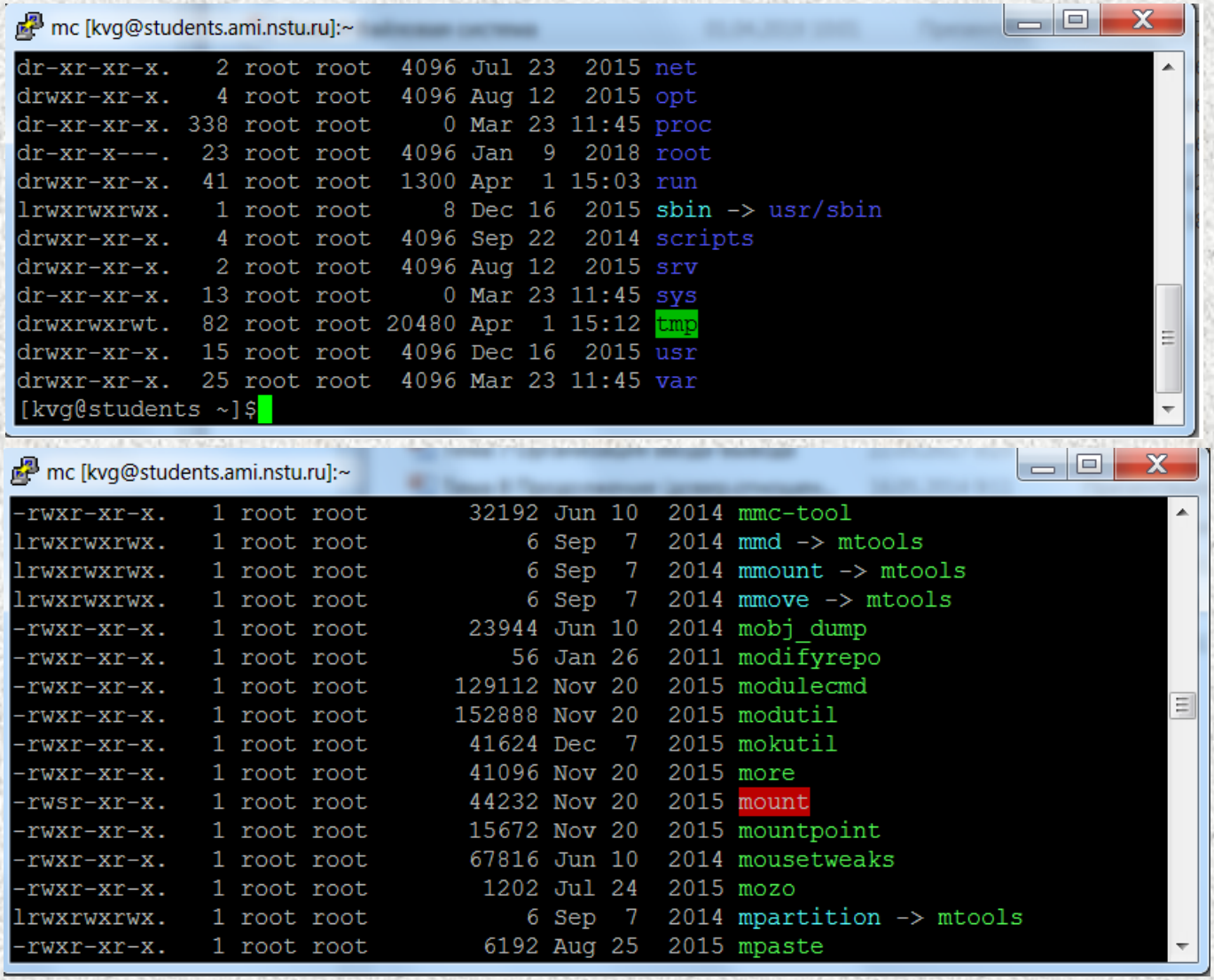
Каждый файл или каталог имеет права доступа. Права доступа определяют, **КТО** и **ЧТО** может делать с содержимым файла. Права доступа отображаются командой **ls -l** в виде строки, состоящей из 10 символов, например: -rwxr-xr- -

Права доступа задаются командой **chmod режим имя_файла**

Например, chmod g+x, o+x myfile1

Право	Обозначение	Файл	Каталог
Чтение	r	Файл можно посмотреть и скопировать	Можно посмотреть список входящих файлов
Запись	w	Файл можно изменить и переименовать	Можно создавать и удалять файлы
Выполнение	x	Файл можно запустить на выполнение (скрипты и программы)	Можно входить, делать текущим
Запрет удаления	t		Файлы в каталоге может удалить только владелец или администратор
Запуск от имени владельца	s	Программу разрешается запустить от имени владельца	

Примеры прав доступа



Обратите внимание: здесь в каталоге /tmp запрещено удалять чужие файлы, а программу mount разрешено запускать от имени администратора.

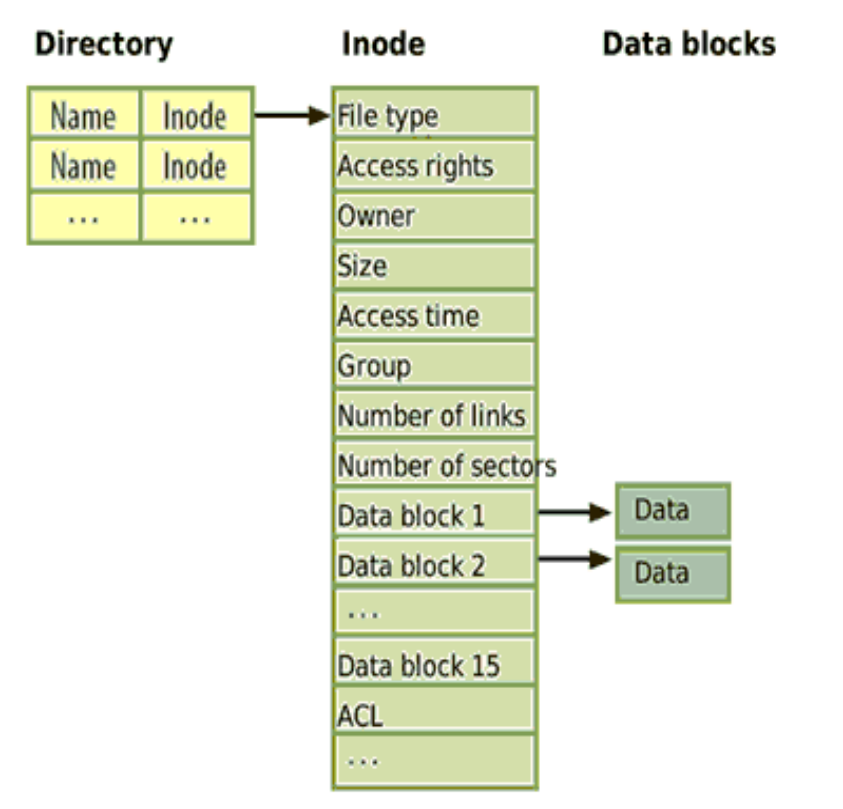
Типы файловых систем Linux

Файловые системы ОС Linux делятся на два типа – локальные и распределённые (сетевые). Локальные файловые системы могут располагаться во внешней памяти (ext2, ext3, ext4 и др.) или в оперативной памяти (псевдо-файловые системы). К последней группе относятся:

- /proc – используется в качестве интерфейса к структурам данных в ядре; большинство расположенных в ней файлов доступны только для чтения;
- /tmpfs – позволяет не записывать на физические диски временные файлы, которые формируются в оперативной памяти, а затем удаляются; поддерживает работу с виртуальной памятью;
- /devfs – предназначена для управления устройствами;
- /sysfs – используется для получения информации о всех устройствах и драйверах.

Распределенные файловые системы предназначены для объединения на логическом уровне файловых систем отдельных компьютеров в единое целое. В Linux такой системой является nfs (Network File System).

Организация хранения данных в Linux



Файловые системы Linux используют три главных структуры данных: каталоги, индексные дескрипторы (**inode, i-узлы**) и блоки данных (**data blocks**).

Каталоги содержат только записи имен файлов и соответствующих им номеров **inode**. При этом на один и тот же **inode** могут указывать несколько записей каталогов. Такие записи называются жесткими ссылками.

Мягкой или символической ссылкой (симлинк) называется файл, *содержимое* которого указывает на *имя* другого файла, а не на индексный дескриптор.

Каталоги хранятся на жестком диске как обычные файлы, отличающиеся от последних только типом файла и тем, что их содержимое представляет собой обязательную структуру.

Файловая система ext2

В первом блоке ext2 располагается **загрузчик**, все остальное пространство делится на блоки равного размера (обычно 1 Кбайт). Блоки объединяются в группы.

В каждую группу входит суперблок, описатель группы, два битовых массива (для блоков и для i-узлов), i-узлы и блоки для хранения данных. **Суперблок** хранит информацию о размере группы, о количестве i-узлов и блоков данных в группе, и т.д.

Описатель группы содержит информацию о расположении битовых массивов, количестве свободных блоков и i-узлов в группе, а также о количестве каталогов в группе. **Битовые массивы** предназначены для учета свободных блоков и i-узлов, для хранения каждого массива выделяется один блок. При размере блока 1Кбайт размер группы равен 8192 блока и количество i-узлов равно 8192.

Файловая система ext2 (продолжение)

Ниже показана структура i-узлов, размер каждого узла составляет 128 байт

Ре-жим	Счетчик связей	UID	GID	Размер	Метки времени	Адреса блоков	Однократный косвенный блок	Двукратный косвенный блок	Трехкратный косвенный блок

Здесь обозначено:

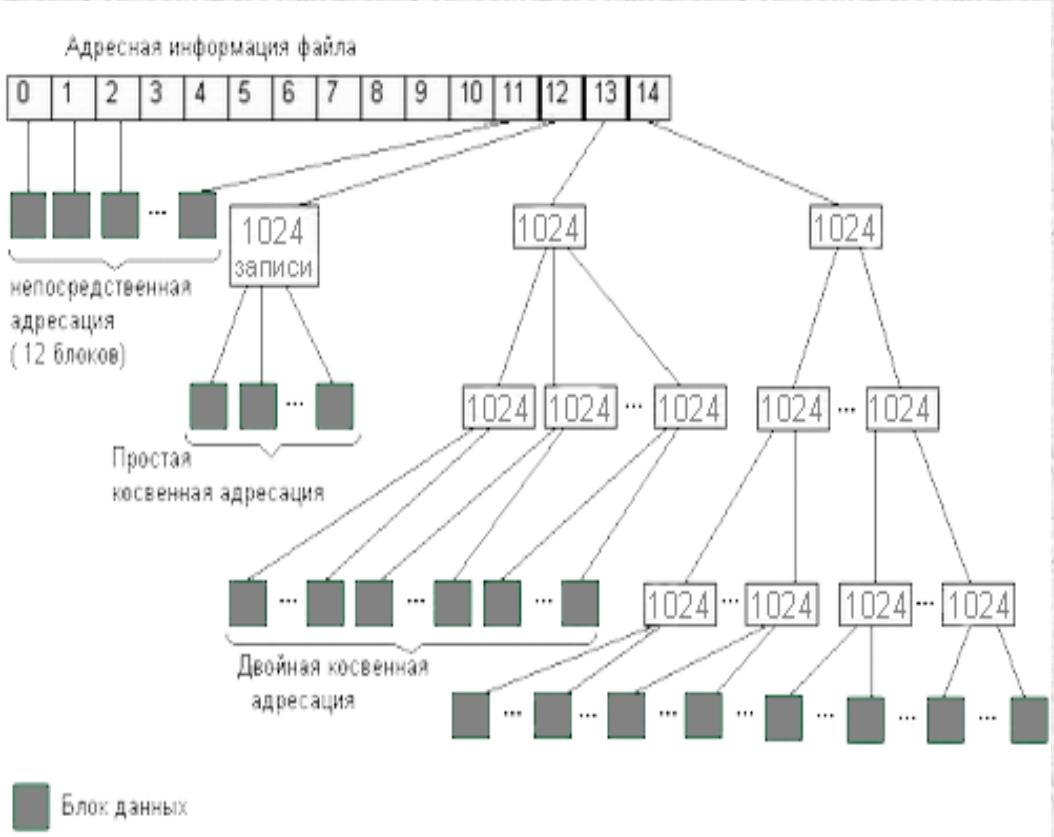
- режим – тип файла, биты защиты;
- счетчик связей – число записей каталогов, указывающих на этот i-узел;
- UID – идентификатор владельца файла;
- GID – идентификатор группы владельца;
- размер – размер файла в байтах;
- метки времени – времена последнего доступа и последнего изменения файла, а также время последнего изменения i-узла;
- адреса блоков – адреса первых 12 блоков файла, размер адреса – 4 байта;
- однократный косвенный блок – адрес одинарного косвенного блока, который содержит адреса дополнительных блоков файла;
- двукратный косвенный блок – содержит адреса 256 одинарных косвенных блоков, каждый из которых содержит адреса 256 блоков данных;
- трехкратный косвенный блок – содержит адреса 256 двукратных косвенных блоков.

Часть информации из i-узла можно посмотреть командой **stat *имя_файла***

Система адресации в ext2

Для хранения адреса файла выделено 15 полей размером 4 байта. Если файл умещается в 12 блоков, то номера этих блоков перечисляются в первых 12-ти полях адреса.

Если размер файла превышает 12 блоков, то в 13-е поле записывается адрес блока, в котором расположены номера следующих блоков файла. При размере блока 4096 байт он может содержать адреса 1024 блоков. Т.е. 13-е поле используется для косвенной адресации.



Если размер файла превышает (12 + 1024) блоков, то используется 14-е поле, в котором находится адрес блока, содержащего 1024 номеров блоков, каждый из которых ссылается на 1024 блока файла. Т.е. это поле двойной косвенной адресации.

15-е поле используется для тройной косвенной адресации (если файл включает более (12+1024+104857) блоков).

Данная система адресации позволяет для блока 4 Кбайт иметь файлы размером до 2 Тбайт.

Особенности файловой системы ext4

1. Максимальный размер файла – до 16 Тбайт (2^{44} байт).

2. Используются *48-битные* номера блоков. При размере блока 4 Кб это позволяет адресовать до одного экзобайта (2^{60} байтов), т.е. размер одного раздела диска может достигать 2^{60} байтов.

3. Используется смежно-индексированное выделение памяти с применением экстентов. Экстенты позволяют адресовать большое количество последовательно идущих блоков (до 128 Мб) одним дескриптором. До четырёх указателей на экстенты может размещаться непосредственно в i-узле.

4. Размер индексного дескриптора увеличен до 256 байтов.

5. Является журналируемой файловой системой.

2. Файловая система FAT

Системная область магнитного диска содержит:

- *загрузочная запись* (начальный загрузчик), содержится в нулевом секторе диска;
- *таблица размещения файлов* (File Allocation Table, FAT) – карта диска, содержит информацию о размещении файлов в области данных. Размер таблицы зависит от объема диска и размера блока (кластера). На любом диске всегда хранится два экземпляра FAT для обеспечения надежного доступа к данным. Элементы FAT могут быть 12-ти, 16-ти и 32-х разрядными, в зависимости от объема диска, соответственно файловые системы называются FAT12, FAT16 или FAT32 .
- *корневой каталог* – главный каталог диска, который занимает сектора, следующие за FAT.

FAT.Организация каталогов

Корневой каталог – совокупность записей размером 32 байта, структура которых показана в таблице. Номер начального кластера определяет точку входа в FAT для данного файла и одновременно дисковый адрес собственно файла. Все атрибуты хранятся в одном байте. Корневой каталог обычно имеет размер 512 записей.

Каталоги нижнего уровня (подкаталоги) имеют структуру, аналогичную корневому каталогу, только в отличие от него они не имеют фиксированного размера и фиксированного дискового адреса, т.е. хранятся на диске в области данных как обычные файлы.

При создании каталога в него сразу добавляются две служебные записи, в которых поле имени содержит «.» и «..». Первая в поле адреса содержит адрес созданного каталога, а вторая – адрес родительского каталога.

Номер поля	**Длина (байт)**	**Назначение поля**
1	8	имя файла
2	3	расширение имени
3	1	атрибуты
4	10	резерв
5	2	время создания/модификации
6	2	дата создания/модификации
7	2	номер начального кластера
8	4	размер файла

FAT. Хранение длинных имен в Windows

Файловая система VFATдля каждого файла и подкаталога хранит два имени – длинное и короткое. Хранение длинных имен организуется в специальных записях каталога, у которых байт атрибутов равен 0Fh. Такие записи невидимы для 16-разрядных программ и могут хранить до 13 символов в кодировке UNICODE.

Для регистрации файла с длинным именем в каталоге выделяется необходимое количество *специальных* записей, а также одна *стандартная* запись для хранения короткого имени. Блок специальных записей всегда располагается в каталоге перед стандартной записью, поэтому если к каталогу обращается 16-разрядная программа, то она будет видеть только короткое имя файла, а 32-разрядные Windows-приложения могут работать с длинными именами.

Короткое имя образуется из длинного следующим образом: оставляется 6 символов длинного имени и дописываются знак “~” (тильда) и порядковый номер в пределах каталога для файлов, у которых первые 6 символов имени совпадают.

Например для файла «Отчет по лабораторной работе» (28 символов) в каталоге будет выделено 3 специальных записи и одна стандартная , хранящая короткое имя «Отчет ~1» и другие метаданные файла.

FAT. Таблица размещения файлов

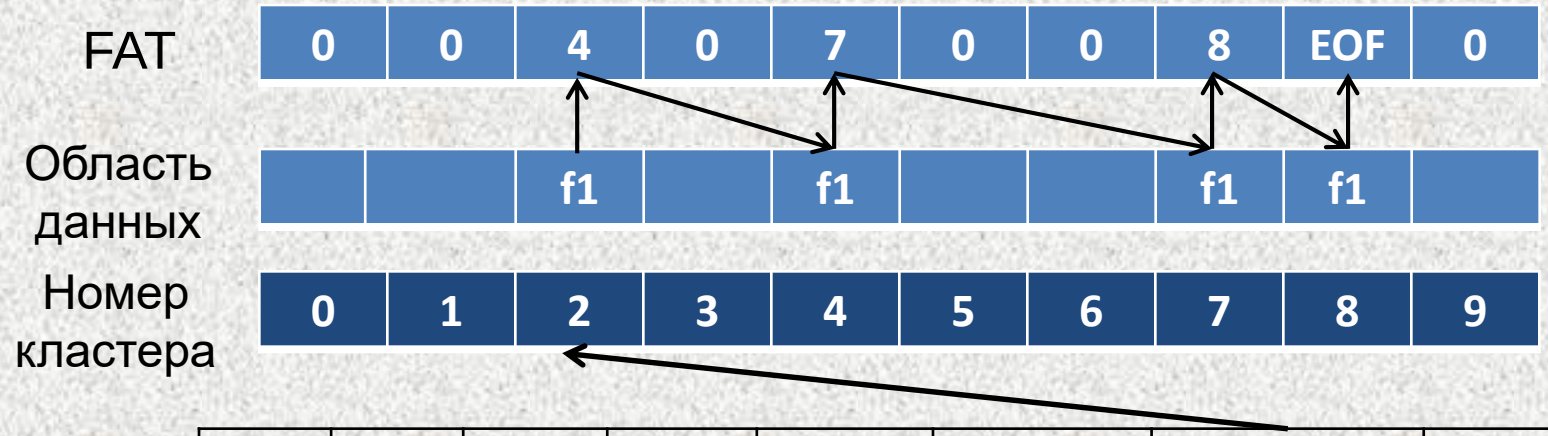
Таблица FAT содержит информацию о номерах кластеров, выделенных для хранения каждого файла. Она представляет собой карту области данных. Каждый элемент таблицы соответствует одному кластеру в области данных. Возможные значения элементов:

- «0» - если кластер свободен;
- целое число (номер следующего кластера) – если кластер занят, но не является последним для файла;
- «EOF» (признак конца файла) - если кластер занят и является последним для файла.

Минимальный размер кластера в разделе диска с файловой системой FAT зависит об объема раздела ($V_{разд}$) и разрядности элемента FAT (r):

$$V_{кл\_min} = V_{разд} / 2^r$$

Пример доступа к файлу с именем F1, занимающему кластеры 2, 4, 7, 8.



Запись каталога

Имя	Тип	Атри-бут	Резерв	Время созд.	Дата созд.	Номер нач. кластера	Размер файла
F1		r		12-50	01.05.14	2	14652

FAT. Удаление файлов

При удалении файла выполняются следующие действия:

- в FAT обнуляются все элементы, выделенные для этого файла;

- в соответствующем элементе каталога изменяется имя файла – вместо первого символа в поле имени записывается символ «x».

Остальные характеристики файла в элементе каталога, а также содержимое файла в кластерах диска, не изменяются, поэтому всегда есть возможность полностью или частично восстановить удаленный файл. Полное восстановление возможно, если:

- не перезаписан соответствующий элемент каталога;

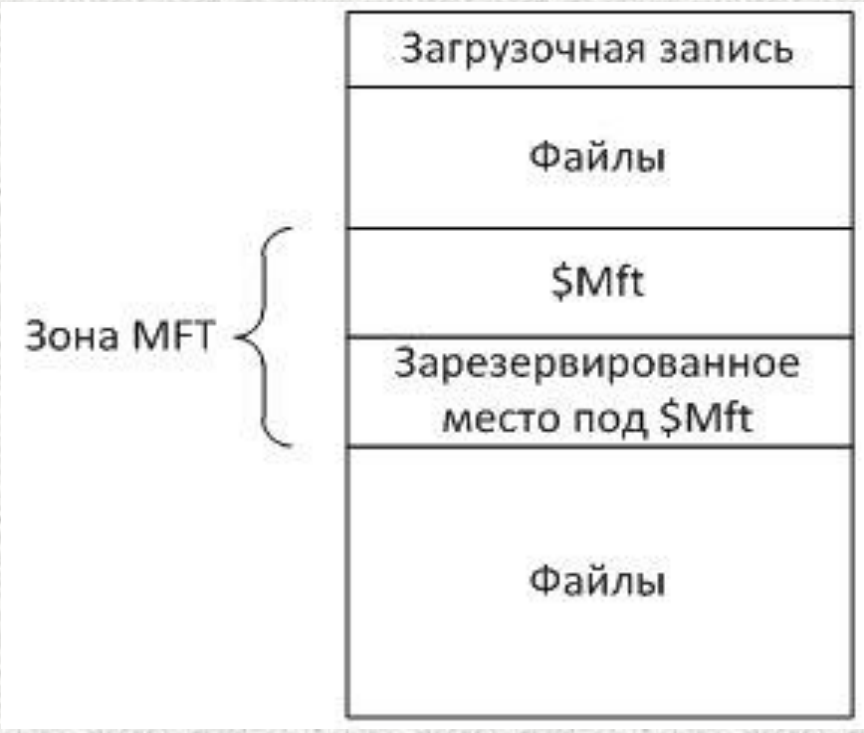
- имеется доступ к каталогу;

- кластеры, ранее занимаемые файлом, не выделены другим файлам или каталогам;

- удаленный файл был нефрагментированным.

Для восстановления удаленных файлов используются специальные программы (утилиты **Undelete**, **Recuva** , дисковые редакторы **Acronis**, **DiskExplorer**, **DMDE** и т.д.).

3. Файловая система NTFS



Основным отличием NTFS от файловой системы FAT является хранение основных системных структур данных в виде обычных файлов.

В первом блоке раздела находится загрузочная запись ($Boot), содержащая программу загрузки и информацию о разделе (тип файловой системы и адреса основных системных файлов). Загрузочная запись обычно занимает 8 Кбайт (16 секторов).

NTFS использует два вида каталогов – главный и обычный. Каждый раздел диска имеет один главный каталог, в котором регистрируются все файлы раздела - MFT (Master File Table), который хранится в файле $MFT. Размер одной записи MFT – 1 Кбайт. Для хранения MFT на диске выделяется участок размером 12% объема диска (зона MFT). Адрес $MFT хранится в загрузочной записи.

Стандартный размер кластера – 4 Кбайт.

NTFS. Таблица Master File Table

1. В записи MFT хранится вся информация о файле (имя, дата и время создания, размер, положение на диске отдельных фрагментов, и т.д). Если не хватает одной записи MFT, то используются несколько, причем не обязательно подряд. При этом первая запись называется базовой.

2. Каждая запись MFT имеет уникальный номер – индекс, общее количество записей – до 2^{48}.

3. Первые 16 записей выделены для описания системных метафайлов, причем самая первая запись описывает структуру самого MFT.

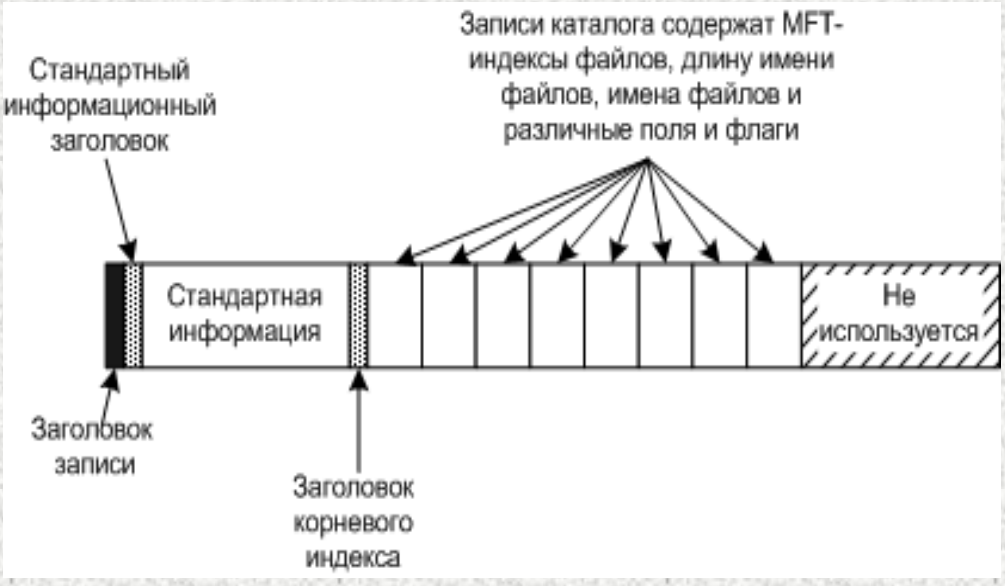
4. В записи MFT можно размещать содержимое небольших по размеру файлов (несколько сотен байтов) без выделения места в области данных, что позволяет существенно экономить дисковое пространство. В этом случае файл называется *непосредственным*.

NTFS. Основные системные файлы

Имя файла	Назначение
$MFT	централизованный каталог всех файлов раздела
$MFTmirr	копия первых 4-х записей MFT
$LogFile	файл поддержки журналирования
$Volume	служебная информация - метка тома, версия файловой системы и т.д
$AttrDef	список стандартных атрибутов файлов на томе
$.	корневой каталог
$Bitmap	карта свободного места тома
$Boot	загрузочная запись
$Quota	файл, в котором записаны права пользователей на использование дискового пространства
$Upcase	файл - таблица соответствия строчных и прописных букв в именах файлов на текущем томе; нужен потому, что в NTFS имена файлов записываются в кодировке Unicode, что составляет более 65 тысяч различных символов, искать большие и малые эквиваленты которых достаточно сложно.

NTFS. Логическая структура записи MFT

а) для слабо фрагментированного файла



б) для небольшого каталога

Особенности NTFS

Для **больших каталогов** вместо простого списка файлов используется бинарное дерево, обеспечивающее быстрый поиск файла:

а) имена файлов упорядочиваются;

б) применяется метод половинного деления для определения местоположения искомого файла.

Например, если в каталоге зарегистрированы 1000 файлов, то для последовательного поиска (FAT) придется осуществить в среднем 500 сравнений (наиболее вероятно, что файл будет найден на середине поиска), а системе на основе дерева (NTFS) – не более 12-ти ($2^{10} = 1024$).

Для обеспечения **повышенной надежности** в NTFS используются:

а) журналирование - ведение журнала транзакций, под которыми понимается последовательность действий, совершаемых целиком и корректно или не совершаемых вообще;

б) возможность перемещения или фрагментации по диску всех своих служебных областей при возникновении любых неисправностей поверхностей диска (кроме первых 16 элементов MFT)

Работа с именованными потоками

Файлы NTFS могут содержать несколько потоков данных - основной (неименованный) и дополнительные (именованные). При этом файловые процессоры в поле «Размер файла» обычно выводят размер основного потока.

1. Пусть имеется файл ***primer.txt*** *размером 15 байтов, содержащий строку:*

   Hello,students!

2. Создаем в файле именованный поток:

   Echo This is alternate stream> primer.txt:potok1

3. Просмотрим текущий каталог и определим размер файла :

   Dir (размер файла не изменился, команда не видит второй поток)

4. Просмотрим содержимое файла и потока:

   type primer.txt (вывод основного потока)

   type primer.txt:potok1 (ошибка, команда не видит второй поток !)

   more< primer.txt:potok1 (вывод на экран строки «This is alternate stream»)

5. Создадим в файле новый поток, содержащий графический файл 3630 Кбайт:

   type foto.jpg > primer.txt:potok.jpg

6. Просмотрим текущий каталог и определим размер файла :

   Dir (размер файла не изменился)

7. Извлечем графические данные из потока:

   mspaint primer.txt:*potok.jpg*

Работа с именованными потоками (продолжение)

1. Именованные потоки не распознаются большинством команд ОС и многими программами (например, антивирусными) .
2. Для получения информации по всем потокам файла необходимо применять специальные утилиты (например, **nfi.exe, sdir.exe, lads.exe**).
3. В ОС Windows именованные потоки используются только в разделах NTFS.
4. При удалении небольшого файла на диске NTFS реально может освободиться несколько Гигабайтов памяти.

Сравнение FAT и NTFS

Ограничения / возможности	NTFS	FAT16 и FAT32
Размеры диска	2^{64} байт	2^{43} байт (8 Тб)
Размер тома	теоретически до 2^{64} Байт; разметка диска в стиле MBR позволяет создавать разделы до 2 ТБ; разметка GPT – до 9.4×10^{21} байт);	на диске FAT32 – до 4 177 920 кластеров. При максимальном размере кластера 32 Кбайт размер тома может быть до 127.53 Гб.
поддержка ссылок разных типов	поддерживает жесткие и мягкие ссылки	не поддерживает ссылки
Максимальный размер файла	теоретически до 2^{64} байт, практически —до 16 Тб	FAT16 поддерживает файлы размером не более 2 Гигабайт FAT32 поддерживает файлы размером не более 4 Гигабайт
Максимальное количество файлов	4 294 967 295 ($2^{32}-1$)	в FAT32 не более 268 435 444 ($2^{28}-12$)

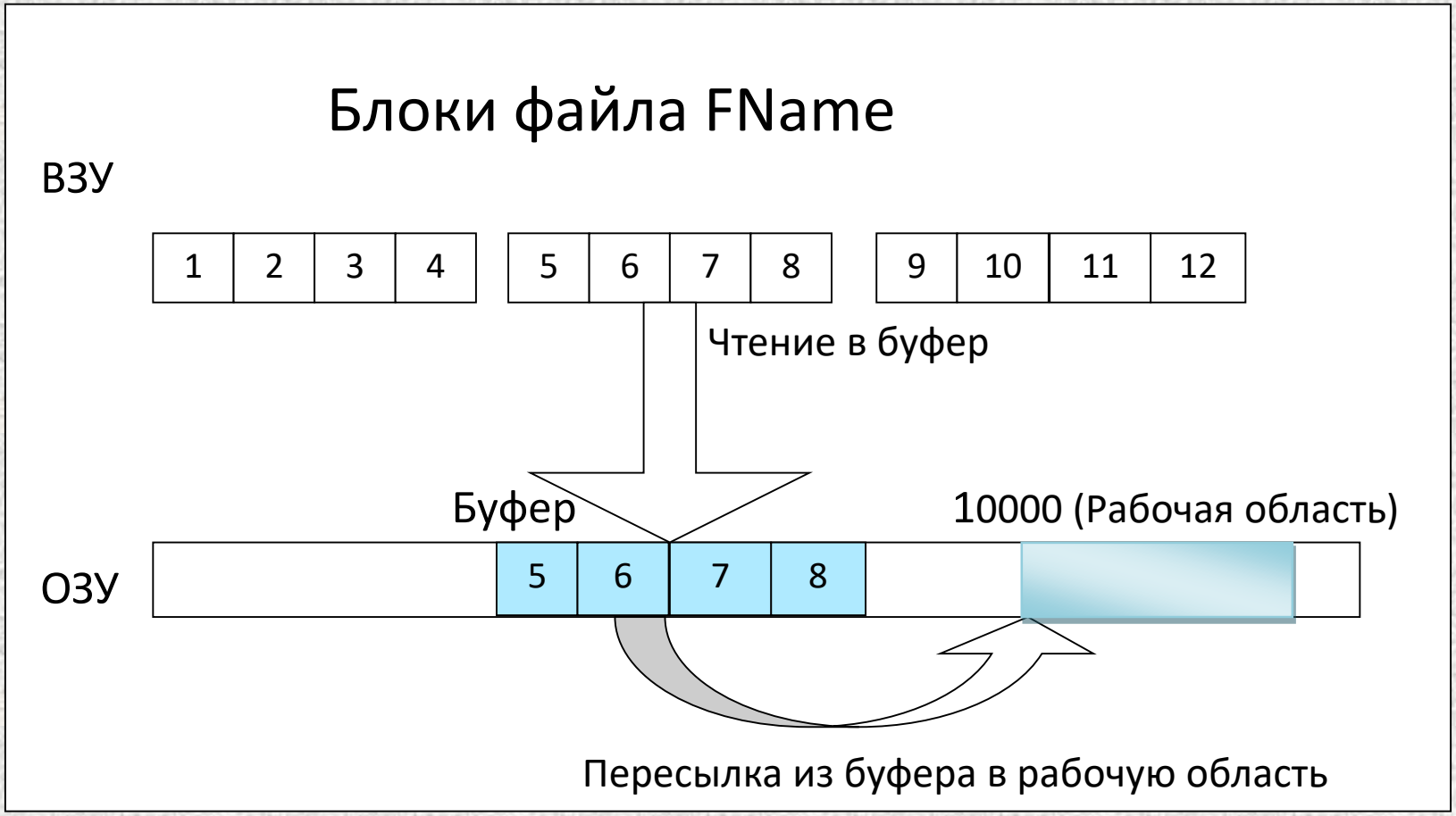
Часть 2

Основные принципы функционирования файловой системы

Структура и порядок выполнения запроса к файловой системе

Структуру файловой системы будем изучать на примере многопользовательской ОС семейства UNIX.

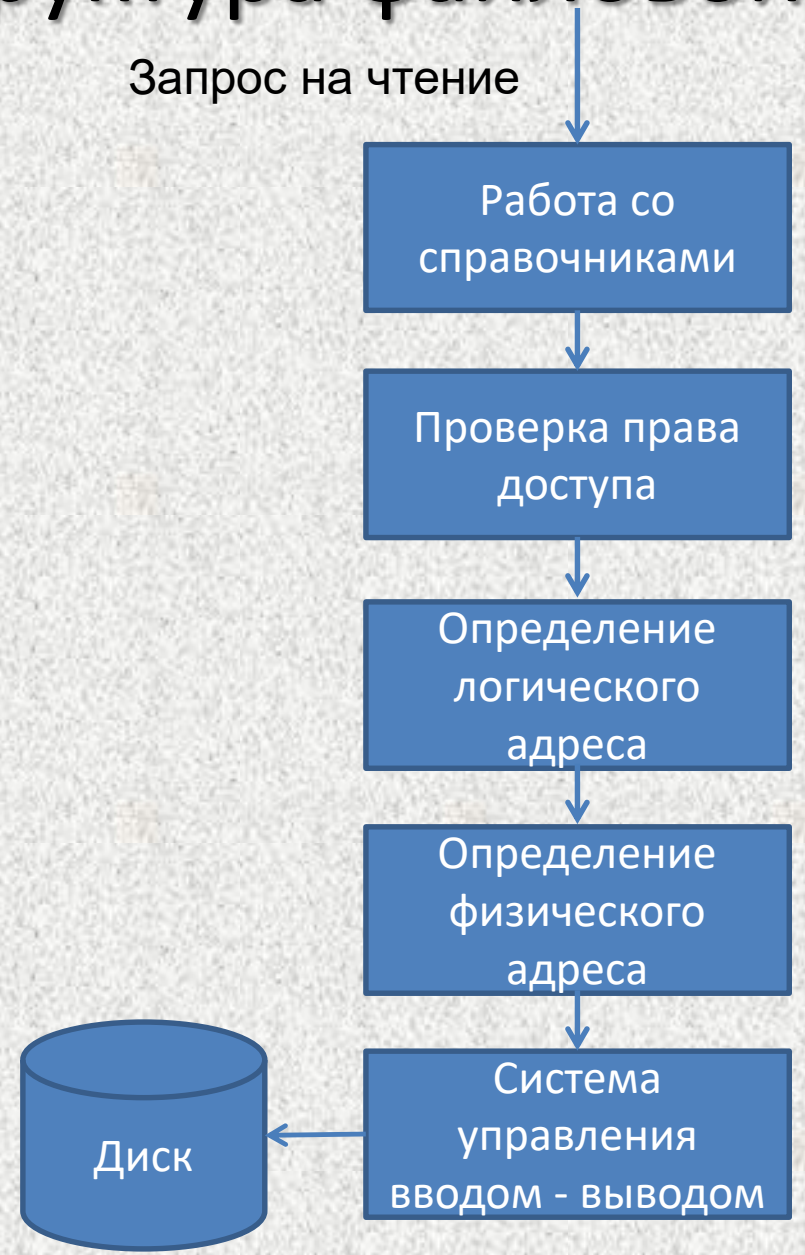
Типовой запрос: *считать из файла **Fname** запись **№ 6** в область памяти по адресу **10000***



Алгоритм выполнения запроса

1. Проверить наличие файла в указанном каталоге. При отсутствии вывести сообщение об ошибке.
2. Выбрать из индексного дескриптора указанного файла все его характеристики и скопировать их в ОЗУ.
3. Проверить право доступа к файлу с указанной операцией (чтение, запись, выполнение). При отсутствии прав вывести сообщение об ошибке.
4. Найти логический номер блока внутри файла.
5. Определить номер физического блока диска, в котором находится требуемая запись.
6. Считать блок с найденным номером с диска в буфер.
7. Выделить из буфера требуемую запись и переслать ее в рабочую область программы (в сегмент данных).

Структура файловой системы



Работа со справочниками (каталогами)

Базовый справочник – единый справочник всех файлов в одном разделе диска. Содержит все метаданные по каждому файлу за исключением имени, вместо которого используется уникальный идентификатор файла (ИДФ). В ОС Linux базовый справочник представлен массивом индексных дескрипторов (i-узлов.)

Символьный справочник (каталог) – связывает ИДФ и имя файла.

Главный символьный справочник содержит список зарегистрированных пользователей и идентификаторы файлов, содержащих их домашние каталоги.

Функции уровня «Работа со справочниками»:

- ведение символьных справочников;
- ведение базового справочника;
- возможность присваивать различные имена одному файлу;
- возможность присваивать одно имя различным файлам;
- ведение таблицы активных имен.

Проверка права доступа к файлам

Функции:

- хранение и изменение прав доступа;
- проверка прав доступа

Права доступа к файлам:

- полный доступ (write);
- чтение (read);
- исполнение (execute);
- запуск от имени владельца (s);
- запрет удаления чужих файлов (t);
- отсутствуют.

Категории пользователей:

владелец;
группа;
остальные

Методы проверки:

- матрица управления доступом;
- список категорий пользователей в базовом справочнике;
- пароли.

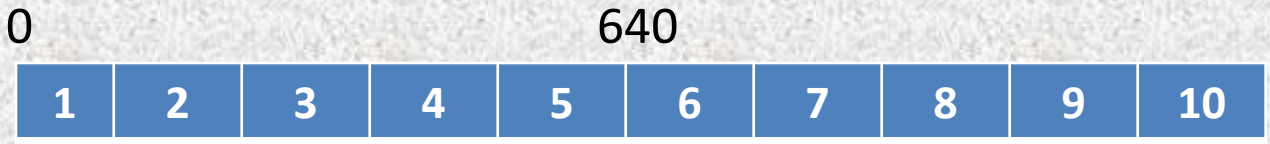
Пример матрицы управления доступом

ИДФ	3	4	5	6	7	8	9	10	11
Иванов	W	W	N	N	W	R	W	R	N
Петров	R	N	W	N	E	W	E	W	E
Сидоров	N	N	N	W	E	N	E	A	E
Попов	N	N	N	N	E	N	E	N	W

Определение адресов

Логический адрес записи – адрес записи относительно начала файла. Если к записи обращаемся по ключу, то значение ключа предварительно преобразуется в номер записи.

Логический адрес определяется расчетным путем с учетом формата и размера записей. Например, для файла с записями фиксированного размера 128 байтов логический адрес записи № 6 будет равен: 5* 128 = 640 байтов.

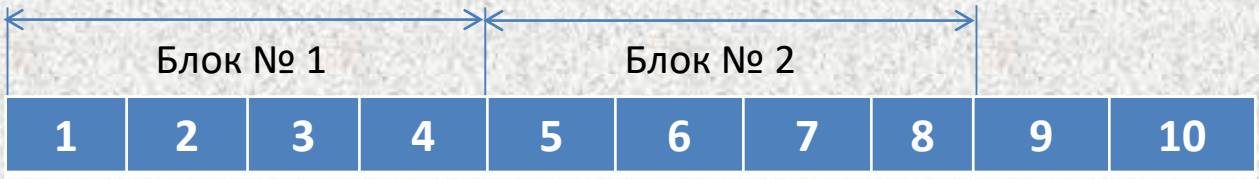


Логический адрес записи преобразуется в *физический адрес* блока диска.

Алгоритм преобразования:

1. По размеру блока и логическому адресу записи определяется логический номер блока внутри файла;
2. Из схемы размещения блоков файла на диске (FAT или i-узлы) определяется физический номер блока диска (номер дискового кластера).

Например, при размере блока 512 байт и размере записи 128 байт искомая запись № 6 будет находится в блоке № 2 файла.



Система управления вводом-выводом

Основные функции:

- обеспечение общего интерфейса к драйверам устройств,
- именование устройств,
- буферизация,
- распределение памяти на блок-ориентированных устройствах,
- распределение и освобождение выделенных устройств,
- уведомление об ошибках.

Буферизация. Способы управления буферами

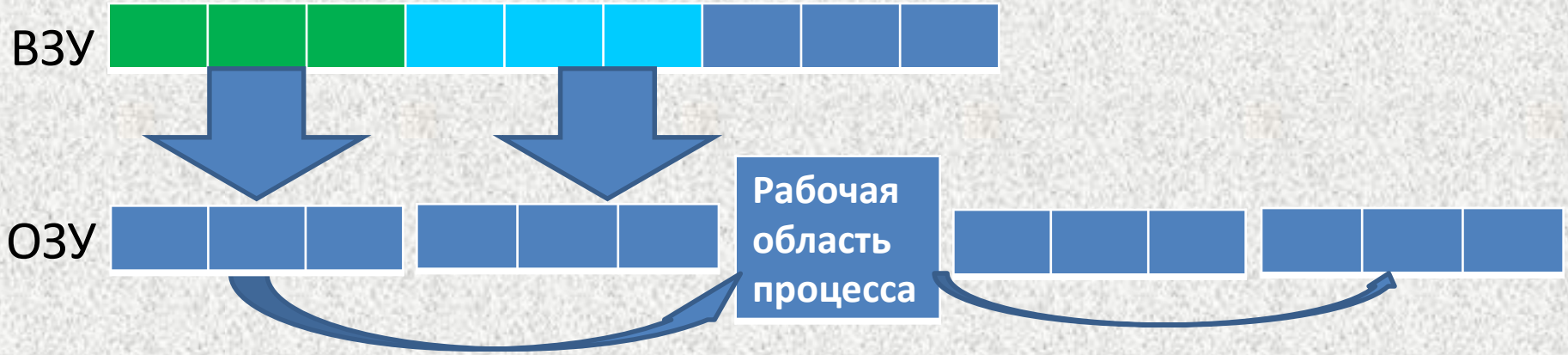
Буфер – участок оперативной памяти, выделенный для временного хранения блоков данных. Возможны два способа управления буферами – по запросу и с упреждением.

Буферизация по запросу основана на том, что все действия по работе с внешним устройством управляются программистом в коде своей программы (запрос буфера, загрузка данных в буфер, синхронизация процессов исполнения и ввода-вывода, освобождение буфера). Недостаток – сложность, достоинство – возможность обработки файлов с любым способом организации.

Буферизация с упреждением полностью управляется файловой системой:

- используется для обработки последовательных, индексно-последовательных и библиотечных файлов, т.е. когда известна последовательность обработки блоков;
- для операций обмена выделяется не менее 2-х буферов;
- загрузка буферов проводится до момента выдачи запроса на ввод записей.
- синхронизация процессов исполнения программы и обмена проводится файловой системой.

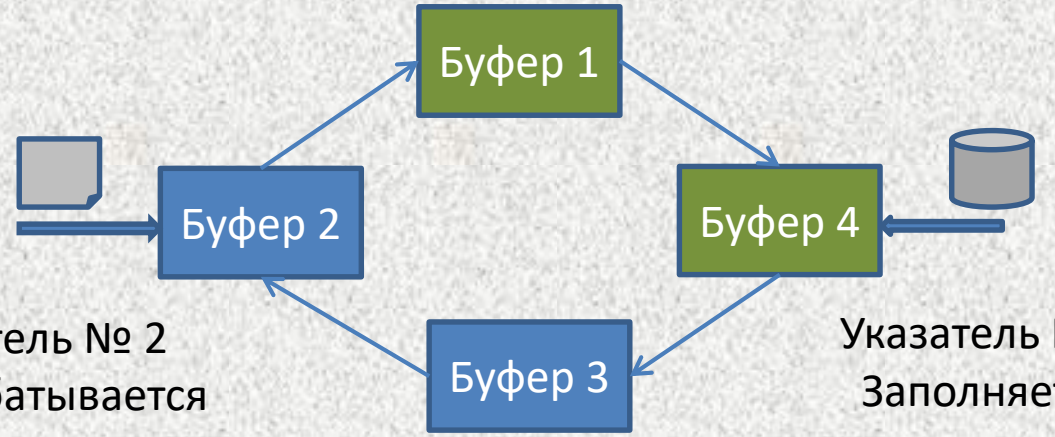
Простая буферизация



Входные буферы

Выходные буферы

Кольцевая буферизация



Указатель № 2
Обрабатывается процессом

Указатель № 1
Заполняется данными

- полный буфер

- пустой буфер

Отображаемые файлы

Отображаемый файл – объект в адресном пространстве процесса, значение которого представляет содержимое заданного файла, расположенного на диске. В основе механизма отображения лежит использование виртуальной памяти.

Каждый файл отображается в собственном сегменте адресного пространства процесса.

С одним отображаемым файлом могут одновременно работать несколько процессов.

Достоинства:

1. Отсутствует необходимость использования буферов, т.к. данные сразу попадают в страницы пользовательской памяти.
2. Уменьшается число системных вызовов.
3. Упрощается программирование за счет использования адресных указателей.
4. Могут использоваться для организации обмена данными между процессами.

Отображаемые файлы (продолжение)

Недостатки:

1.Нельзя увеличить размер отображаемого файла.

2.Если один процесс работает с отображением файла, а другой – с реальным файлом на диске, то возникают проблемы с их согласованием, т.к. изменения, внесенные в отображаемый файл, будут сохранены только при вытеснении соответствующей страницы на диск.

3.Размер реального файла может превышать максимальный размер сегмента.

Реализация в Object Pascal:

Функции Object Pascal для работы с отображаемыми файлами: *CreateFileMapping()* –создание отображаемого файла; *MapViewOfFile()* – проецирование данных файла в адресное пространство процесса; *UnMapViewOfFile()* – прекращение отображения файла в адресное пространство процесса;

Реализация в C#:

Метод *MemoryMappedFile.CreateFromFile* - создание отображаемого файла;

Метод *MemoryMappedFile. OpenExisting* – подключение к существующему отображаемому файлу;

Метод *MemoryMappedFile.CreateViewStream* – получение последовательного доступа к отображаемому файлу;

Метод *MemoryMappedFile.CreateViewAccessor* – получение произвольного доступа к отображаемому файлу;

Тема 5. Управление памятью

Основные функции системы управления памятью

Оперативная память является важнейшим ресурсом, требующим управления со стороны мультипрограммной операционной системы.

Представляет собой набор нумерованных ячеек памяти (байтов). Номер байта называется **адресом**.

Основные функции:

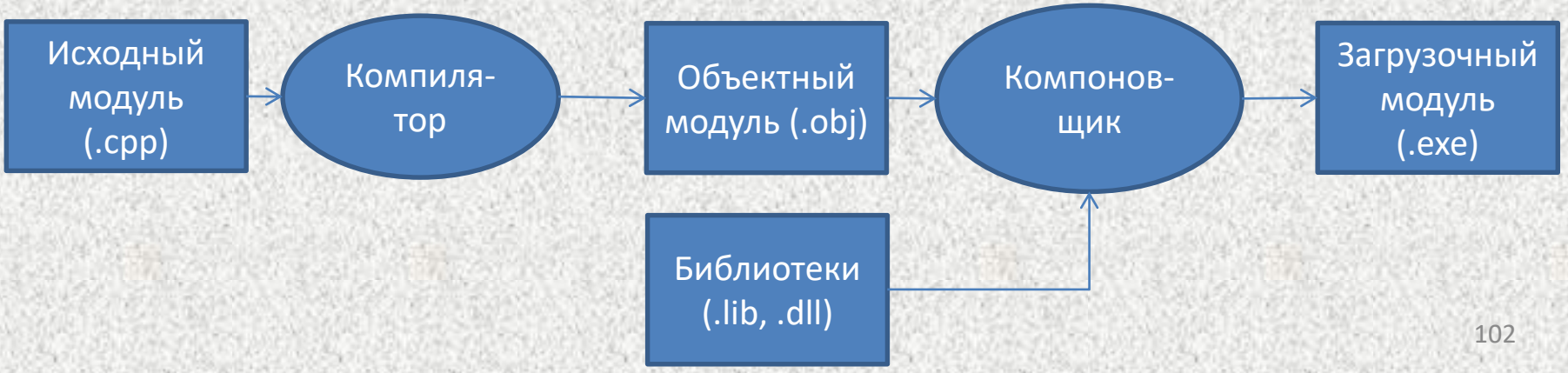
1. определение стратегии выделения памяти;
2. учет свободной и занятой памяти;
3. распределение памяти между конкурирующими процессами и защита адресных пространств процессов;
4. отображение адресов программы на конкретную область физической памяти;

Понятие логического адреса

Современные программы являются перемещаемыми, т.е. могут выполняться в любом участке ОЗУ. Для этого все адреса в программах делаются логическими, т.е. не привязанными к физическим адресам ОЗУ.

Компилятор преобразует исходную программу в набор машинных двоичных команд (объектный модуль) с **относительной** адресацией. Адрес первой команды принимается равным нулю и относительно него отсчитываются адреса всех команд и данных. Для каждой переменной, используемой в программе, выделяется участок памяти также с относительной адресацией.

Компоновщик подключает к программе другие объектные модули, требуемые для выполнения, в результате чего получается загрузочный (исполняемый) модуль также с **относительной** адресацией.



Преобразование адресов

Преобразование логических адресов в физические проводится перед загрузкой исполняемого модуля в ОЗУ с помощью **настраивающего загрузчика**. Начальный адрес выделенного участка памяти в ОЗУ (A_o) записывается в один из регистров процессора (базовый регистр) и каждый относительный адрес преобразуется в физический следующим образом:

$$A_{физ} = A_o + A_{отн}$$

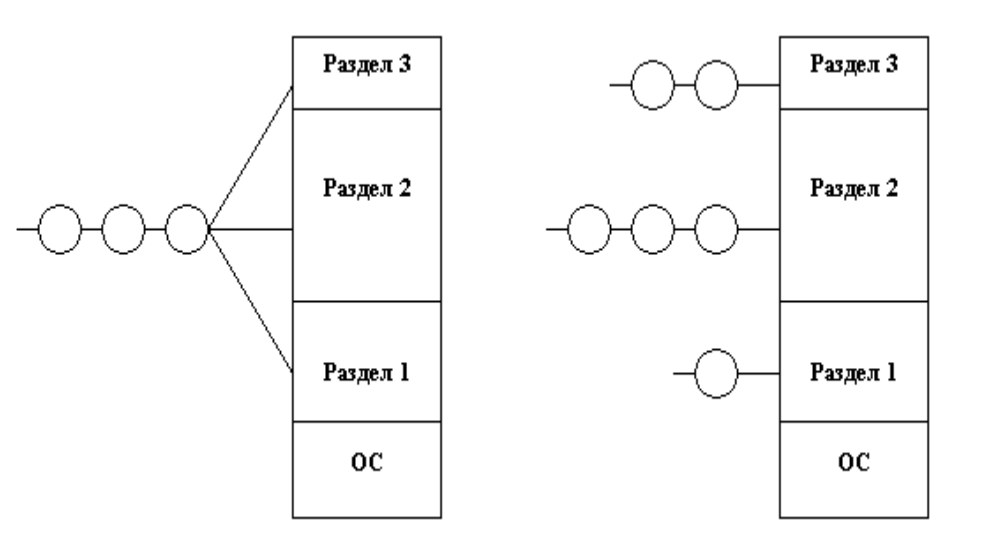
В приведенном примере $A_o = 1000$;

Классификация методов распределения памяти

В основу классификации положена история развития систем управления памятью.

Выделение фиксированными разделами

1. При загрузке ОС вся оперативная память делится на несколько разделов фиксированного размера.
2. Каждый раздел может иметь свою очередь задач (в этом случае каждому заданию заранее назначается собственный раздел ОЗУ) или может существовать одна глобальная очередь для всех разделов.



Выделение фиксированными разделами (продолжение)

Достоинство : простота управления.

Недостатки:

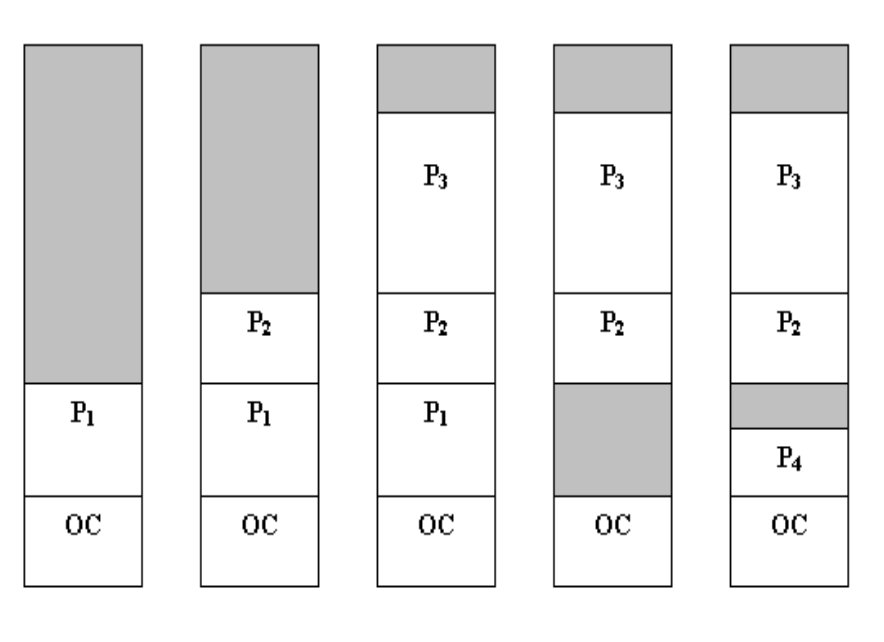
а) число одновременно выполняемых задач ограничено числом разделов;

б) необходимость программирования в условиях ограничений на размер ОЗУ;

в) очень большой уровень *фрагментации* из-за того, что процесс не полностью занимает выделенный ему раздел или вследствие неиспользования некоторых разделов, которые слишком малы для задач, находящихся в очереди).

Фрагментация памяти – появление в ОЗУ свободных участков, недоступных для размещения задач.

Выделение динамическими разделами



Динамические разделы имеют переменный размер, который определяется требованиями задач.

По истечении некоторого времени память представляет собой набор занятых и свободных участков.

Смежные свободные участки могут быть объединены в один.

Алгоритм работы программы – распределителя памяти:

1. анализ запроса на выделение свободного участка (раздела);
2. выборка свободного участка среди имеющихся в соответствии с одной из стратегий;
3. загрузка задачи в выбранный раздел;
4. внесение изменений в таблицы свободных и занятых областей ОЗУ.

Выделение динамическими разделами (продолжение)

Выделение динамических разделов проводится на основе механизма базирования с использованием **таблицы свободной памяти**.

Эта таблица может упорядочиваться по адресам свободных участков или по их размеру. В зависимости от этого задачам будут выделяться разные участки памяти. Например, для задачи размером 28 Кбайт в первом случае будет выделен участок по адресу 180 К, а во втором – по адресу 400 К.

Адрес участка	Размер
100 К	24 К
180 К	120 К
320 К	16 К
400 К	48 К
520 К	80 К

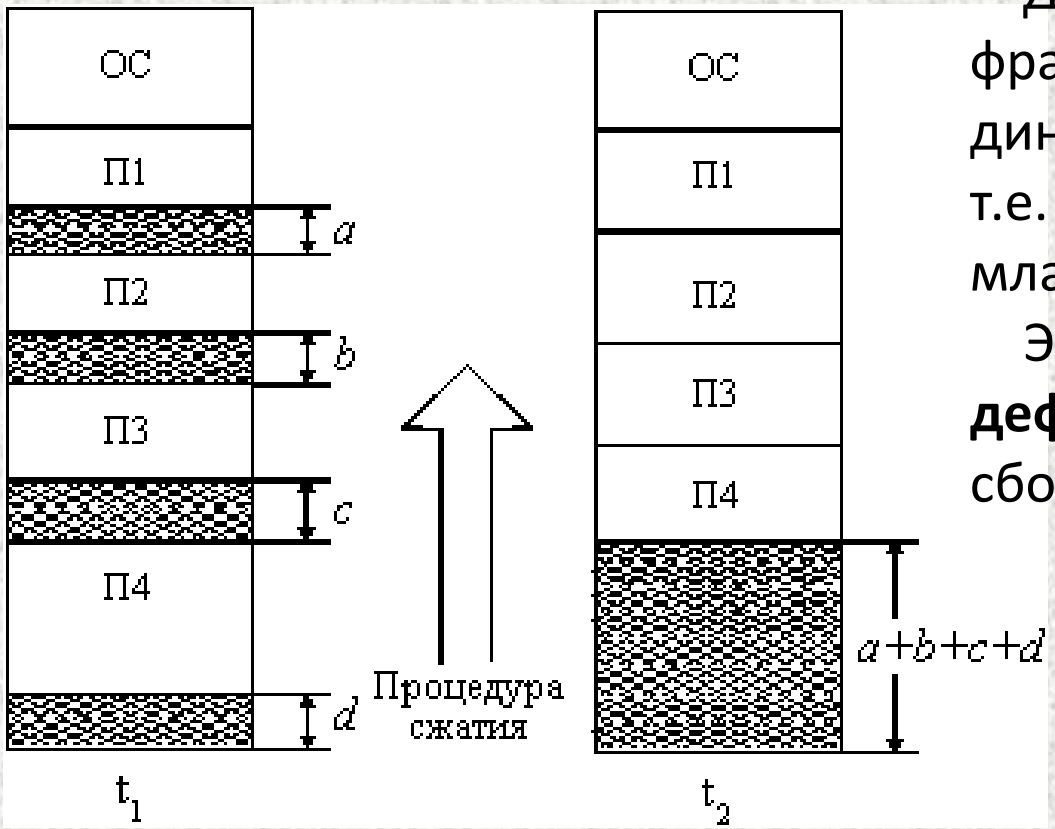
а) сортировка по адресу

Адрес участка	Размер
320 К	16 К
100 К	24 К
400 К	48 К
520 К	80 К
180 К	120 К

б) сортировка по размеру

Перемещаемые разделы

Распределение памяти динамическими разделами уменьшает фрагментацию по сравнению с фиксированными разделами, но при длительной работе системы в ОЗУ возникает большое число небольших по размеру свободных участков, недоступных для загрузки задач.



Для борьбы с такой фрагментацией используется динамическое перемещение задач, т.е. все задачи сдвигаются в сторону младших адресов ОЗУ.

Эта процедура называется **дефрагментацией памяти** или сборкой мусора.

Перемещаемые разделы (продолжение)

Некоторые системы управления памятью имеют механизм **«свертывания - развертывания»** задач.

При поступлении в очередь приоритетной задачи в случае отсутствия свободного участка достаточного размера проводится принудительное вытеснение (свертывание) любой низкоприоритетной задачи на диск и загрузка в освобожденное место приоритетной задачи. По окончании ее выполнения свернутая задача копируется с диска в ОЗУ (развертывается) и продолжает работу. В общем случае развертывание может проводиться в любой свободный участок памяти.

Синонимами термина «свертывание-развертывание» являются «**выгрузка**» или «**своппинг**».

В ОС больших ЭВМ часто используется распределение ОЗУ в режиме **«передний план - фон».** Вся память делится на 2 раздела, в фоновом разделе выполняются громоздкие задачи, требующие значительных ресурсов, а в разделе переднего плана – задачи, работающие в интерактивном режиме. В фоновый раздел управление передается только в случае отсутствия задач в разделе переднего плана.

Страничное распределение памяти

Основные принципы:

1.Вся память делится на **страницы фиксированного размера S,** называемые **кадрами или фреймами** (обычно S=2…4 Кбайт); каждая страница имеет свой физический номер, нумерация начинается с нуля.

2.Адресное пространство задач делится на **логические** страницы, размер которых совпадает с размером физических страниц ОЗУ.

3.Задачи могут загружаться в ОЗУ в произвольные страницы, в.т.ч. **несмежные**.

4.Для каждой задачи создается собственная таблица страниц, в которой хранится информация о расположении логических страниц задачи в физических страницах ОЗУ (**таблица отображения страниц задачи**).

5.Каждый логический адрес в задаче **аппаратно** преобразуется в два целых числа (пару): **номер страницы** и **смещение** в пределах страницы.

Страничное распределение памяти (продолжение)

Номер логической страницы p	Смещение d

Номер физической страницы n	Смещение d

№ логической страницы	№ физической страницы
0	…..
1	…..
p	n
…..	…..

Для ссылки на таблицу страниц обычно используется специальный регистр. При переключении процессов диспетчер должен найти его таблицу страниц, указатель на которую входит в контекст процесса.

При любом обращении к команде или переменной проводится **динамическое преобразование адресов:**

1. Выполняемый процесс обращается по адресу *v* = (*p*, *d*)
2. Механизм отображения ищет номер страницы *p* в таблице отображения и определяет, что эта страница находится в физической странице *n*, формируя **реальный адрес *v1* = (*n*, *d*).**
3. Вычисляется физический адрес команды:

Афиз = физ. адрес страницы + смещение = n *S + d.

Страничное распределение памяти (пример)

Пусть в ОЗУ со страничной организацией памяти расположены 3 задачи размером 1,2 и 3 Кбайт соответственно. Размер страниц – 1 Кбайт.

Задача 1

№ лог. страницы	№ физ. страницы
0	6

Задача 2

0	5
1	8

Задача 3

	0	2
0		
1К	1	4
2К	2	7

Таблицы отображения страниц

№ страницы	Содержимое страницы
0	ОС
1	ОС
2	Задача 3, стр. 0
3	свободно
4	Задача 3, стр 1
5	Задача 2, стр 0
6	Задача 1, стр. 0
7	Задача 3, стр 2
8	Задача 2, стр 1
9	свободно

Физические страницы ОЗУ

Пусть в задаче № 3 по адресу 20 находится команда:

move EAX, [1064]

Какой физический адрес будет у этой команды после загрузки задачи в ОЗУ ?

Адрес будет равен:

2048 + 20 = 2068;

Из какого адреса будет проводиться загрузка регистра EAX ?

4096 + 40 = 4136

Сегментное распределение памяти

При страничной организации адресное пространство процесса делится механически на равные части, что не позволяет дифференцировать способы доступа к разным частям программы.

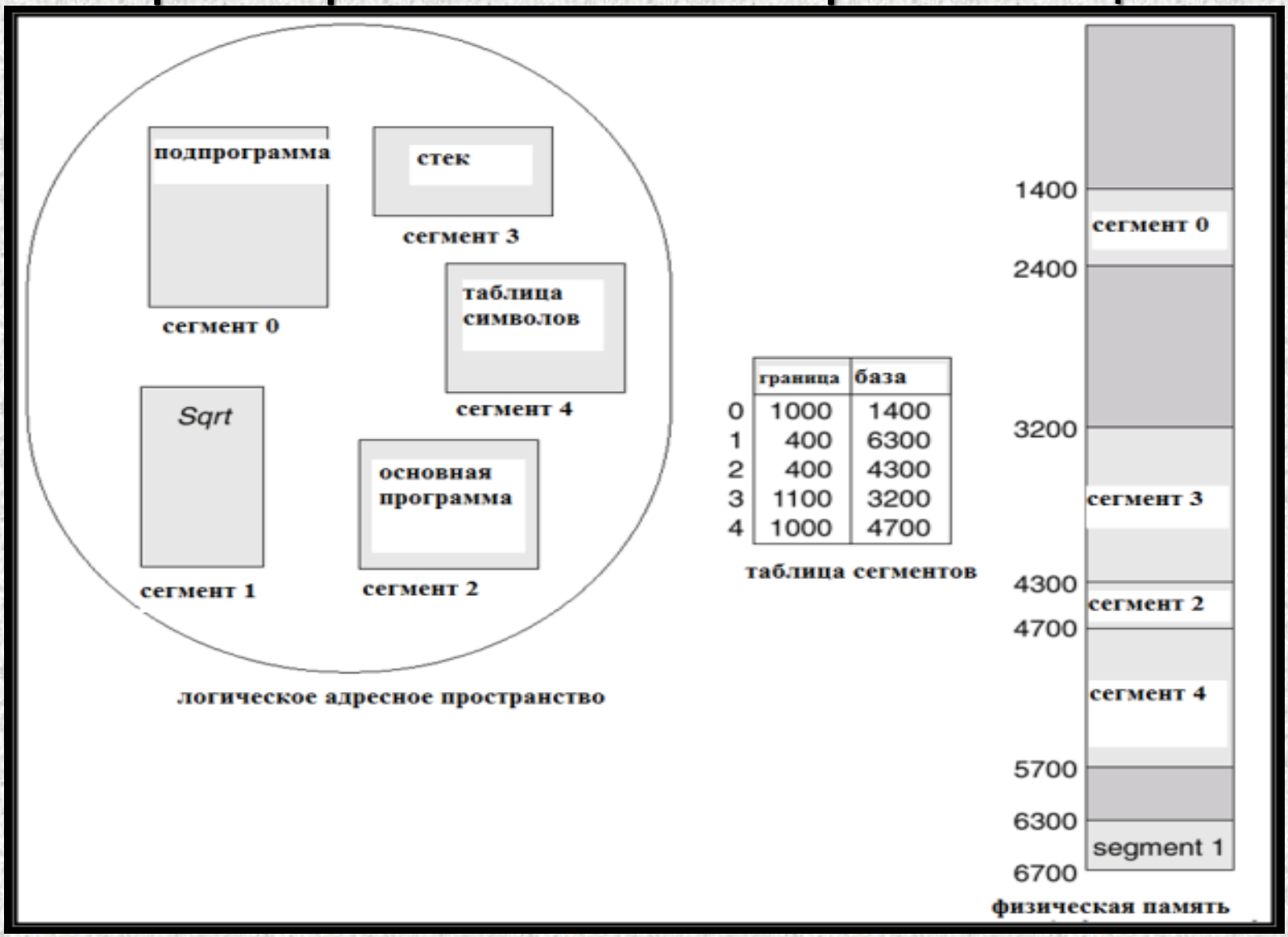
Современные системы программирования создают исполняемый модуль в виде набора отдельных фрагментов (например, код программы, данные и т.д.). Для каждого фрагмента программы ОС могут выделять свой **сегмент** в памяти. Сегмент – логическая единица распределения памяти, предназначенная для размещения в памяти одного фрагмента программного кода или данных. Например, в виде сегментов памяти могут быть представлены:

- основная программа;
- процедуры и функции;
- стек;
- набор локальных переменных;
- набор глобальных переменных;
- отображаемый файл;
- и т.д.

Основные принципы сегментного распределения

1. Логическое адресное пространство представляет собой набор сегментов.
2. У каждого сегмента имеются имя (номер), размер и другие параметры (уровень привилегий, разрешенные виды обращений).
3. Система программирования специфицирует каждый адрес двумя значениями: **номером сегмента и смещением в пределах сегмента**, что отличается от схемы страничного распределения, где система программирования задает только один адрес, который разбивается аппаратурой на номер страницы и смещение.
4. Для отображения логических адресов в физические при сегментной организации памяти служит т**аблица сегментов.**
5. Длина сегментов различна и поэтому ее необходимо хранить в явном виде в таблице сегментов.
6. Логический адрес при сегментной организации – пара «номер сегмента, смещение».
7. При программировании на Ассемблере программист может задавать значения сегментных регистров самостоятельно.

Пример сегментной организации



Если размер смещения в виртуальном адресе выходит за пределы размера сегмента, то возникает прерывание.

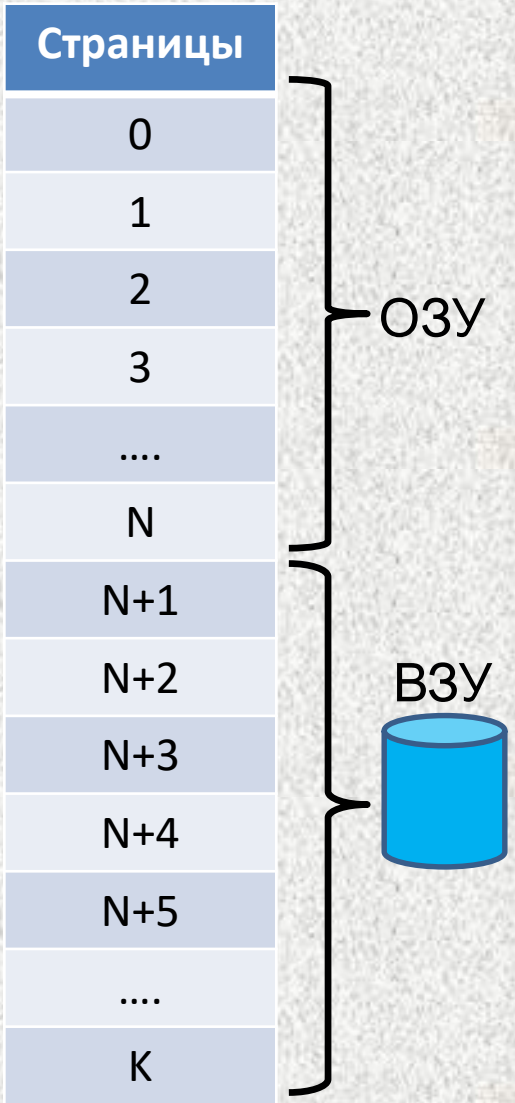
Виртуальная память

1. Виртуальная память – совокупность физической оперативной памяти со страничной или сегментно-страничной организацией и части дискового пространства, организованного по образу оперативной памяти **с единой нумерацией страниц**. Эта часть диска называется **файлом подкачки**.

2. Поддерживаются все механизмы страничной памяти: разбиение ОЗУ на страницы, динамическое преобразование адресов, загрузка задач в несмежные страницы, использование таблиц отображения страниц задач и т.д.

3. В физической памяти достаточно хранить небольшую часть страниц задач. Остальные находятся на диске и вводятся в ОЗУ по мере необходимости, ненужные страницы выводятся на диск.

4. Обмен данными между ОЗУ и диском всегда осуществляется целыми страницами.



Алгоритмы ввода страниц

При обращении задачи к виртуальной странице, отсутствующей в основной памяти, ОС должна: а) выделить физическую страницу в ОЗУ, б) переместить в нее копию требуемой страницы из внешней памяти, в) модифицировать соответствующий элемент таблицы страниц.

Ввод страниц может проводиться двумя способами:

1. **по запросу** : ввод новой страницы проводится, когда имеется обращение из активной страницы, уже размещенной в ОЗУ.
2. **с упреждением**: ввод страниц проводится с опережением на основе различных методов прогнозирования. Часто для уменьшения накладных расходов вводится несколько соседних страниц совместно с той, на которую имеется запрос, при этом уменьшается число обращений к диску.

Алгоритмы вывода (замещения) страниц

Общая схема замещения: а) найти некоторую занятую страницу в ОЗУ; б) переместить при необходимости ее содержимое на ВЗУ; в) записать в этот страничный кадр нужную страницу из ВЗУ; г) модифицировать элемент соответствующей таблицы страниц; д)продолжить выполнение задачи.

Число перемещений страниц может быть уменьшено за счет использования **бита модификации** и **атрибута «read only»**, которые хранятся в таблице страниц. При этом, если в страницу не вносились изменения, то она не переписывается на диск, т.к. ее копия там уже имеется.

Алгоритмы замещения делятся на **глобальные и локальные**. Глобальные алгоритмы могут выводить на диск страницы любого процесса, локальные – только страницы своего процесса. Недостатки глобальных алгоритмов: а) возможность замедления других процессов; б) некорректно работающее приложение может нарушить работу всей системы, пытаясь захватить больше памяти. Локальные алгоритмы:

1. FIFO - каждая страница сопровождается меткой времени, выводится «старейшая» страница.
2. Вывод страницы, не использовавшейся дольше всех (LRU Least Recently Used).
3. Вывод наиболее редко используемой страницы (NFU Not Frequently Used).

Защита памяти

Одна из функций системы управления памятью - защита адресного пространства процессов от несанкционированного доступа со стороны других процессов.

В алгоритмах, использующих **смежное** распределение памяти, основной способ защиты – использование **ключей защиты памяти** (КЗП). КЗП – целое беззнаковое число, генерируемое операционной системой и записываемое в слово состояния программы PSW и в каждый блок памяти, выделенный процессу. При любом обращении к ОЗУ проводится проверка этих чисел на совпадение. Если они совпадают, то доступ разрешается, иначе возникает прерывание типа «чужая память». Этот способ использовался в алгоритмах смежного распределения памяти.

В алгоритмах **несмежного** распределения памяти основной способ защиты – использование **таблицы отображения страниц**, т.к. любой процесс может обращаться только к тем страницам, которые зарегистрированы в его таблице отображения.

Таблицы отображения страниц процессов часто называют **таблицами локальных дескрипторов**. Число таких таблиц равно числу запущенных процессов. Операционная система имеет свою собственную **таблицу глобальных декрипторов**.

Составная виртуальная память

Большой объем виртуальной памяти современных ЭВМ дает возможность выполнять поток задач под управлением разных операционных систем. При этом в начальном разделе хранится **главная операционная система (монитор виртуальных машин)**, которая управляет реальными ресурсами ЭВМ и обеспечивает работу всех других ОС, в другом разделе, например, ОС LINUX, в третьем – ОС QNX. В каждом разделе могут выполняться свои задачи. В этом случае можно говорить о **мульти-программировании на уровне ОС**.

Современные ЭВМ широко используют концепцию виртуализации вычислений. Примерами программного обеспечения для виртуализации в среде Windows являются системы **Hiper-V** (Microsoft), **VMWare WorkStation** (VMWare**)** и **VirtualBox** (Oracle).

Тема 6. Прохождение программ в среде ОС

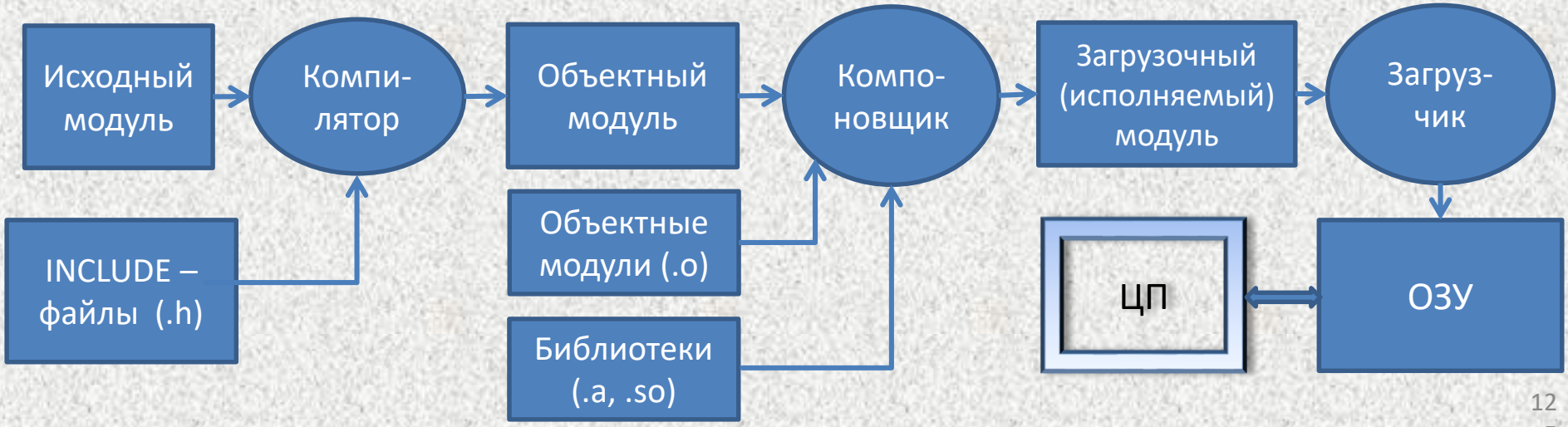
Схема прохождение программы

Вспомним основные этапы прохождения программ в среде ОС.

1. **Компилятор** преобразует исходную программу в набор машинных двоичных команд (объектный модуль) с относительной адресацией. Для каждой переменной, используемой в программе, выделяется участок памяти также с относительной адресацией. В общем случае этот класс программ называется **трансляторами**.

2. **Компоновщик** подключает к программе другие объектные модули, требуемые для выполнения, в результате чего получается загрузочный (исполняемый) модуль также с относительной адресацией.

3. **Загрузчик** преобразует относительную адресацию исполняемого модуля в абсолютную (физическую) с учетом используемого алгоритма распределения памяти , загружает этот модуль в выделенные участки памяти и передает процессору адрес первой команды программы.



Трансляция программы

Основные функции транслятора:

- контроль синтаксических ошибок,
- перевод программы с языка высокого уровня на язык машинных двоичных команд.

Известны два типа трансляторов – интерпретаторы и компиляторы. **Интерпретатор** переводит каждую строку программы в набор машинных команд и этот набор сразу же исполняется. Далее обрабатывается следующая строка и т.д. Достоинства - экономия памяти и удобство отладки, недостаток – низкая скорость выполнения программ. Примеры интерпретаторов – Basic, PHP, HTML, Python.

Компилятор переводит в машинные команды исходный модуль целиком , после чего сформированный объектный модуль передается на дальнейшую обработку. Компилятор требует больше памяти, но подготовленные им программы работают быстрее.

Далее будем говорить в основном о компилирующих системах разработки.

Основные этапы работы компилятора

1. Препроцессорная обработка (подключение include – файлов; например, #include <stdio.h>);
2. Лексический анализ (выделение ключевых слов и конструкций языка программирования);
3. Синтаксический анализ (выявление ошибок в синтаксисе операторов);
4. Генерация команд Ассемблера (не обязательный этап);
5. Генерация машинного кода.

Типы ошибок, обнаруживаемых компилятором:

1. Лексические – связаны с распознаванием лексем языка (например, wile вместо while, dbl вместо double и т.д.)
2. Синтаксические – вызваны нарушением синтаксиса конструкций языка (пропущена точка с запятой, неравное число открывающихся и закрывающихся скобок или кавычек, использование неописанных переменных и т.д.).
3. Наведённые – вызваны использованием в операторах программы ранее обнаруженных ошибочных строк.

Структура объектного модуля

Все компиляторы, работающие в одной ОС, должны создавать объектные модули **одинаковой структуры** для того, чтобы можно было объединять в одной программе модули, написанные на разных языках программирования.

ESD
Код
RLD
Признак конца

Модуль содержит переработанную компилятором программу в виде нескольких сегментов (машинный код, статические данные, динамические данные, стек) и служебную информацию.

Связь между функциями внутри программы и с другими программами организуется с помощью **внешних символов**, которые выносятся в словарь External Symbol Dictionary (ESD).

Внешние символы – это имя программы, имена вызываемых функций и т.д. Каждый символ хранится в отдельной строке словаря, куда во время компоновки заносятся адреса соответствующих функций.

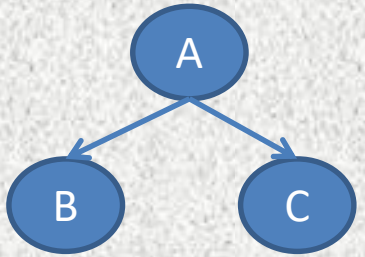
Словарь перемещений (Relocation Dictionary, RLD) содержит информацию о всех переменных и других адресных константах, используемых в программе (имена и адреса) .

Словари используются компоновщиком на этапе формирования исполняемого модуля.

Структура словарей объектного модуля

Структура ESD для программы, состоящей из трех модулей. Поле «Адрес» заполняется на этапе компоновки.

Имя	Тип	Адрес
A	внешнее имя	
B	внешний символ	
C	внешний символ	
printf	внешний символ	
……	……	



Структура RLD:

Имя	Относительный Адрес
alfa	A1
beta	A2
gamma	A3
……	---

Управление компилятором gcc

Командная строка для вызова:

gcc [ключ1 ключ2...] имя_файла1 имя_файла2...

где "ключ" - последовательность символов, начинающаяся с символа "-" и указывающая режим работы компилятора; «имя_файла» - имя обрабатываемого файла.

После успешной компиляции входных файлов компилятор вызывает компоновщик для формирования исполняемого модуля.

Компилятор имеет достаточно большое число ключей, но для выполнения лабораторной работы могут потребоваться следующие :

-c - обработать исходную программу без компоновки загрузочного модуля;

-o – указывает имя выходного файла;

-Iпуть_доступа – указывает путь к include-файлам (типа .h).

-Lпуть_доступа – указывает путь к библиотечным файлам, подключаемым на этапе компоновки (типа .a).

Например, вызов компилятора для обработки файла **main.c** с подключением Include – файлов, находящихся в каталоге **.\include**

gcc -c –I.\include main.c

Некоторые ключи компилятора gcc

-B Compile to .asm (-S), then assemble to .obj

-G Optimize for size/speed; use -O1 and -O2 instead

-O Optimize jumps

-Hxxx Generate and use precompiled headers

-I Set the include file search path

-L Library file search path

-M Create a linker map file

-c Compile to object file only, do not link

-e Specify target executable pathname

-g Stop batch compilation after n warnings (Default = 255)

-j Stop batch compilation after n errors (Default = None)

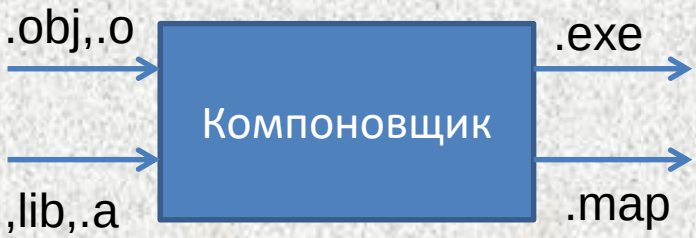
-n Set output directory for object files

-o Set output filename

-w Display all warnings

Компоновка исполняемого файла

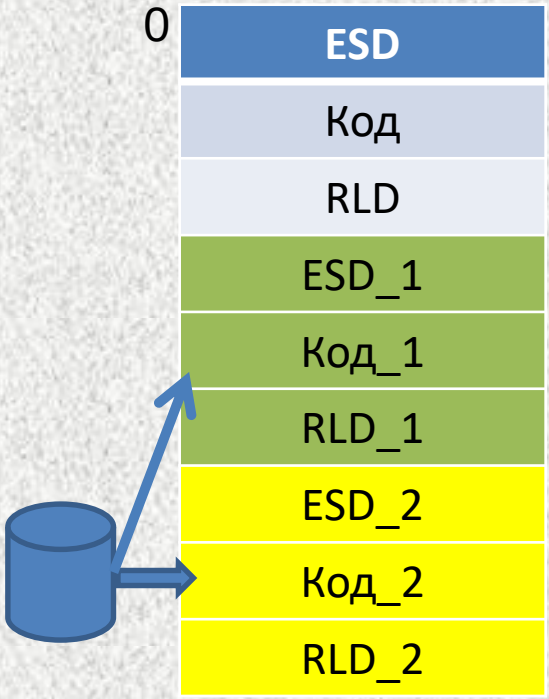
Цель – подключение к программе всех требуемых для ее исполнения **ранее откомпилированных модулей** (процедур и функций) и формирование единого исполняемого модуля с относительной адресацией.



Основные этапы работы компоновщика:

- разрешение всех внешних ссылок,
- коррекция адресных констант.

Алгоритм работы компоновщика основан на анализе словарей подключаемых объектных модулей: из ESD определяются имена подключаемых модулей, а на основе RLD проводится формирование кода с единой адресацией.



Размер исполняемого файла всегда больше размера исходной программы за счет подключения дополнительных модулей.

Структура сформированного загрузочного модуля выводится компоновщиком в текстовый файл с расширением **.map**, который называется картой памяти.

Способы компоновки

Возможны три способа компоновки – статическая, динамическая и оверлейная.

Статическая компоновка (раннее связывание) предполагает, что сборка исполняемого файла проводится до его загрузки в память с использованием статических библиотек (в Windows - файлы типа **.lib**, в Linux – файлы типа **.a**). Этот способ был рассмотрен на предыдущем слайде и имеет следующие недостатки:

а) значительный размер исполняемых файлов;

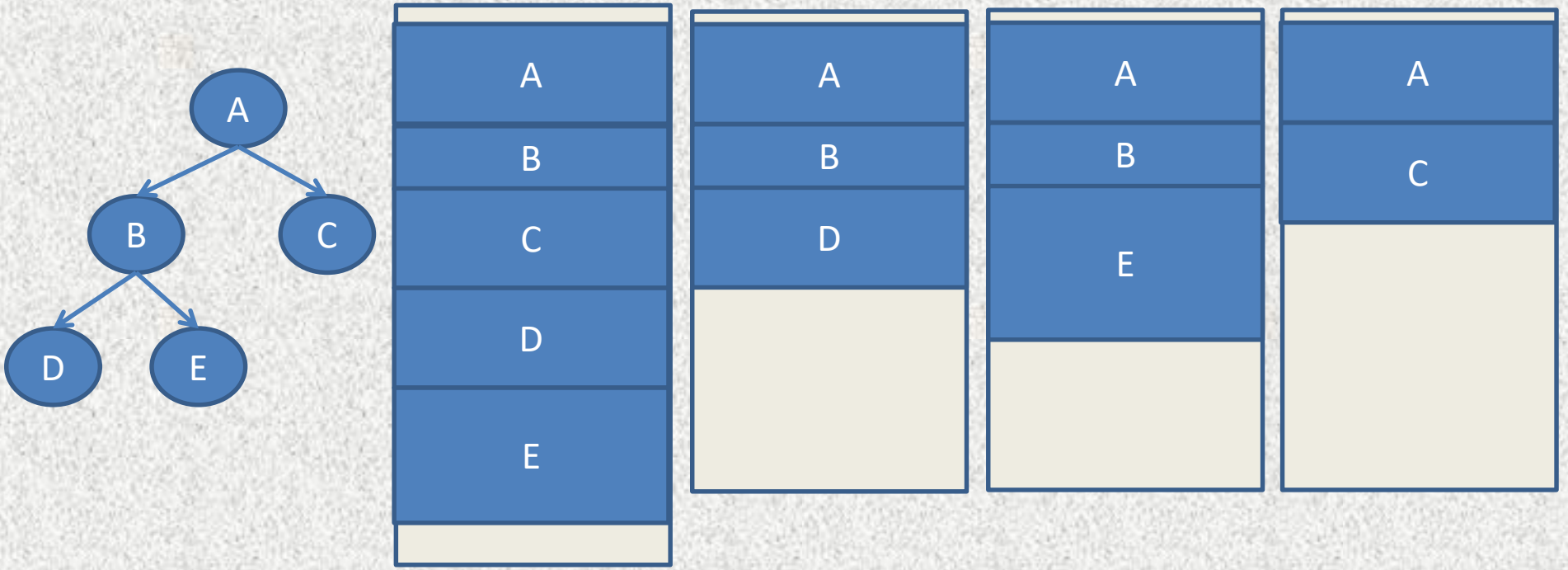
б) неэффективное использование оперативной памяти, т.к. многие загруженные в память процессы содержат одинаковые фрагменты кода (например, код процедуры вывода данных на экран).

Динамическая компоновка (позднее связывание) основана на том, что в состав исполняемого модуля включаются только ссылки на библиотечные функции, а загрузка библиотек в оперативную память и непосредственное извлечение функций из библиотек проводятся на этапе выполнения программы. Для этого метода можно использовать только специальные библиотеки динамической компоновки (в Windows - файлы типа **.dll**, в Linux – файлы типа **.so**). При обращении программы к библиотечной функции проводится проверка наличия этой библиотеки в памяти. Если она загружена, то программе разрешается доступ к функции, иначе процесс блокируется и проводится загрузка библиотеки в ОЗУ.

Способы компоновки (продолжение)

Оверлейная компоновка основана на загрузке отдельных модулей программы с перекрытием в памяти. Для этого компоновщик включает в состав исполняемого модуля ряд специальных команд, с помощью которых ОС проводит загрузку в ОЗУ отдельных модулей программы.

Размер требуемой памяти для модуля простой структуры **Vпр = A+B+C+D+E**, для модуля оверлейной структуры **Vo = max {(A + B + D), (A + B + E), {(A + C) }**



В настоящее время оверлейная компоновка используется только в системах с ограниченным объемом памяти.

Управление компоновщиком Linux

Командная строка для вызова :

ld -dynamic-linker /lib64/ld-linux-x86-64.so.2 -o myprogram

-Map=abcdmap /usr/lib64/crt1.o /usr/lib64/crti.o /usr/lib/gcc/x86_64-redhat-linux/4.8.5/crtbegin.o main.o func1.o -lc /usr/lib/gcc/x86_64-redhat-linux/4.8.5/crtend.o /usr/lib64/crtn.o

Ключи компоновщика:

-l*имя* – добавляет статическую библиотеку с именем **libимя.a** в список файлов для компоновки; например, опция -lm добавляет в список библиотеку математических функций libm.a;

-L*каталог* – добавляет указанный каталог в список каталогов для поиска библиотек;

-Map=*имя* – записывает в текстовый файл с указанным именем информацию об относительных адресах, именах и размерах всех объектных модулей, включенных в состав исполняемого модуля, а также диагностические сообщения, возникшие на этапе компоновки; этот файл называется картой памяти.

На вход компоновщика подаются файлы пользователя func1.o и main.o, а также служебные файлы (crt1.o, crti.o, crtbegin.o, crtend.o, crtn.o).

Управление компоновщиком Windows

Командная строка для вызова использует **позиционные** параметры:

ilink32 [ключи] startup objfiles, [exefile], [mapfile], [libraries], [deffile], [resfile]

Здесь ***startup*** – объектный модуль инициализации, имя которого определяет тип формируемого исполняемого модуля; например *c0x32.obj* - для формирования консольного Windows приложения; *c0w32.obj* - для формирования графического Windows приложения;

objfiles – список имен объектных модулей, разделенных пробелами (.obj);

exefile – имя исполняемого файла (.exe);

mapfile – имя файла, содержащего карту памяти исполняемого модуля, в которую выводится информация об относительных адресах всех объектных модулей, включенных в состав загрузочного модуля (.map);

libraries - список имен библиотечных файлов, разделенных пробелами (.lib);

deffile – имя файла определения модуля (.def), в случае отсутствия задаются свойства по умолчанию;

resfile – имя файла ресурсов (.res), в случае отсутствия в исполняемый файл включаются стандартные ресурсы Windows (курсоры, иконки, BMP-файлы, диалоговые окна, шрифты и т.д.).

Управление компоновщиком Windows (продолжение)

Основные ключи компоновщика:

-m - создать карту памяти;

-x - не создавать карту памяти;

-s - создать подробную карту памяти;

-Lпуть – указывает путь к библиотечным файлам.

Пример вызова компоновщика

ilink32 -s –L.\lib c0x32 main mod1,main,kart,cw32 mylib import32

Здесь на основной вход подаются модуль инициализации **c0x32** и последовательные объектные файлы **main** и **mod1**, на библиотечный вход – библиотека компилятора **cw32**, личная библиотека **mylib** и библиотека импорта функций API **import32**.

В результате компоновки будут сформированы исполняемый файл **main.exe** и текстовый файл карты памяти **kart.map**.

Свойства исполняемого модуля и требуемые ресурсы будут подключены по умолчанию.

Пример фрагмента карты памяти Windows

Start Length Name Class

0001:00401000 000009318H _TEXT CODE

0002:0040B000 00000261CH _DATA DATA

0003:0040D61C 000000870H _BSS BSS

0004:00000000 00000009CH _TLS TLS

Detailed map of segments

0001:000000B0 0000014F C=CODE S=_TEXT G=(none) M=D:\KVG\C0X32.OBJ ACBP=A9

0001:00000200 0000001A C=CODE S=_TEXT G=(none) M=D:\KVG\A.OBJ ACBP=A9

0001:0000021C 00000010 C=CODE S=_TEXT G=(none) M=D:\KVG\ B.OBJ ACBP=A9

0001:0000022C 00000010 C=CODE S=_TEXT G=(none) M=D:\KVG\ C.OBJ ACBP=A9

Выводы:

1. Исполняемый файл содержит 4 сегмента (CODE, DATA, BSS, TLS).
2. Размер сегмента кода – 37656 байтов
3. В составе исполняемого модуля находятся модули:
   - c0x32.obj (335 байтов),
   - a.obj (26 байтов),
   - b.obj, c.obj (по 16 байтов каждый)

Прохождение программ в среде .NET

До появления Интернета большинство программ писалось, компилировалось и предназначалось для конкретного процессора и ОС. С появлением Интернета, когда в глобальную сеть связывались разнотипные процессоры и ОС, возникли проблемы переносимости и межъязыкового взаимодействия программ.

Платформа .NET поддерживает языки C#, VB.NET, C++, F#, а также различные диалекты других языков, привязанные к .NET, например, Delphi.NET. При компиляции код на любом из этих языков компилируется в PE-файл (Portable Executable file), содержащий команды псевдокода языка CIL (Common Intermediate Language). PE-файл имеет расширение **.exe** и может выполняться только на компьютерах, где установлен Framework .NET.

При запуске на выполнение приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется общеязыковой средой исполнения (Common Language Runtime, CLR) . При этом в текущий момент времени будет компилироваться лишь та часть приложения, к которой непосредственно идет обращение. Если мы обратимся к другой части кода, то она будет скомпилирована из CIL в машинный код. Ранее скомпилированная часть приложения сохраняется до завершения работы программы, что в итоге повышает ее производительность.

Существует также версия .NET Core, которая является кроссплатформенным аналогом .NET Framework с открытым исходным кодом и предназначена для работы в Windows, MacOS и Linux.

Прохождение программ в среде .NET (продолжение)

Программы в .NET Framework создаются в форме сборок (модулей), представляющих собой самодостаточный компонент для развертывания, тиражирования и повторного использования. Сборка характеризуется следующими свойствами:

- формируется компилятором языка программирования;
- является единицей повторного использования кода;
- может существовать в виде выполняемого файла (с расширением EXE) или файла динамической библиотеки (с расширением DLL);
- в качестве идентификатора использует номер версии.

Сборка может иметь в своем составе один или несколько файлов с программным кодом, а также служебную информацию, которая называется манифестом. Манифест содержит следующие метаданные о сборке: идентификатор автора и версия сборки, список файлов с указанием по каждому из них контрольной суммы, права на запуск и использование, а также зависимости от других сборок, т.е. имена и версии сборок, которые используются данной сборкой.

В технологии .NET Framework легко решается проблема объединения программ, написанных на разных языках программирования. Для этого каждая из программ с помощью собственного компилятора переводится на промежуточный язык CIL и все откомпилированные файлы включаются в одну сборку.

Организация ввода-вывода в архитектуре x86

Классификации устройств ввода-вывода

По быстродействию:

а) центральные;

б) устройства ввода-вывода (внешние или периферийные).

По назначению:

- устройства ввода (клавиатура, мышь, сканер, сенсорная панель, джойстик, стилус);
- устройства вывода (монитор, принтер, плоттер);
- универсальные устройства (магнитные или оптические диски, USB флэш-накопители, сетевой адаптер, сенсорный монитор).

По способу обмена данными:

- блок – ориентированные (хранят информацию в блоках, каждый из которых имеет собственный адрес; имеется возможность поиска заданного блока по адресу);
- байт – ориентированные (обмен проводится последовательностью байтов; адресация отсутствует; не позволяют проводить операцию поиска).

Взаимодействие ОС и внешних устройств

ОС работает с **контроллером** устройства, а не самим устройством. Именно контроллер знает все особенности конкретного устройства, преобразует поток битов в блоки байтов, проводит контроль и исправление ошибок.

Контроллер имеет собственную встроенную память, состоящую из двух частей. Первая часть доступна для всех устройств, входящих в состав компьютера и используется для хранения блоков принимаемых или передаваемых данных и сигналов управления, а вторая используется для обработки данных и недоступна другим устройствам.

В составе ОС имеется **система управления вводом – выводом**. Это комплекс программ, предназначенный для управления обменом данными между всеми устройствами, входящими в состав ПК.

С контроллером непосредственно работает **драйвер** – программа, управляющая устройством или классом устройств. Драйверы могут входить в состав ОС или поставляться отдельно производителями устройств. Современные ОС содержат большое число встроенных драйверов.

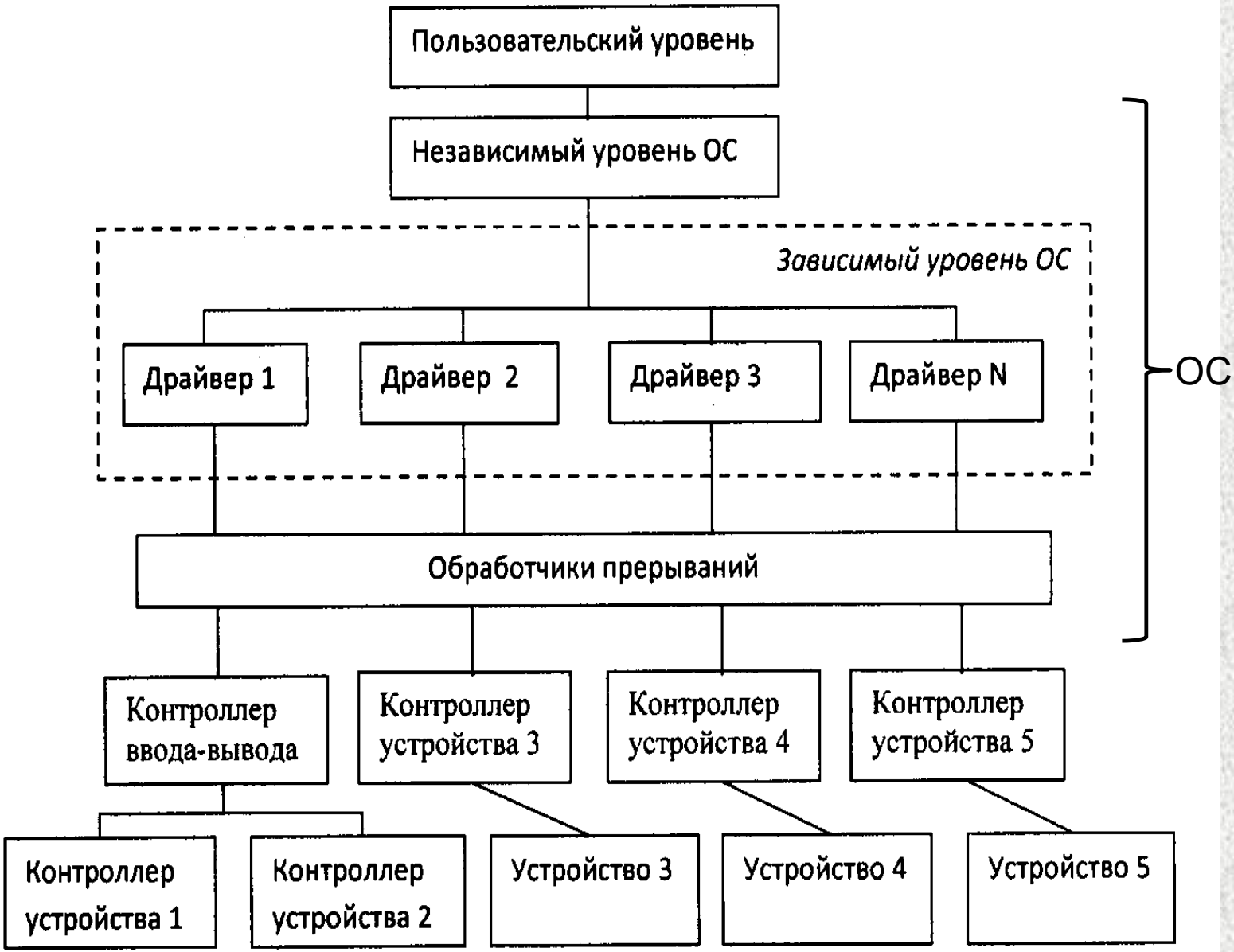
Программная поддержка устройств ввода-вывода

Основная идея организации программного обеспечения ввода-вывода состоит в разбиении его на несколько уровней, причем нижние уровни обеспечивают экранирование особенностей аппаратуры от верхних, а те, в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ключевые принципы:

- **независимость от устройств** . Код программы не должен зависеть от того, читает ли она данные с гибкого диска или с жесткого диска).
- **единообразное именование**. Для именования устройств должны быть приняты единые правила.
- **возможность обработки ошибок**. Ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это ему не удается, то исправлением ошибок должен заняться драйвер устройства.

Иерархия устройств и ПО ввода-вывода



Функции независимого слоя ОС

- обеспечение общего интерфейса к драйверам устройств;
- именование устройств;
- защита устройств;
- обеспечение независимого размера блока;
- буферизация;
- распределение памяти на блок-ориентированных устройствах;
- распределение и освобождение выделенных устройств;
- уведомление об ошибках.

Порты ввода – вывода

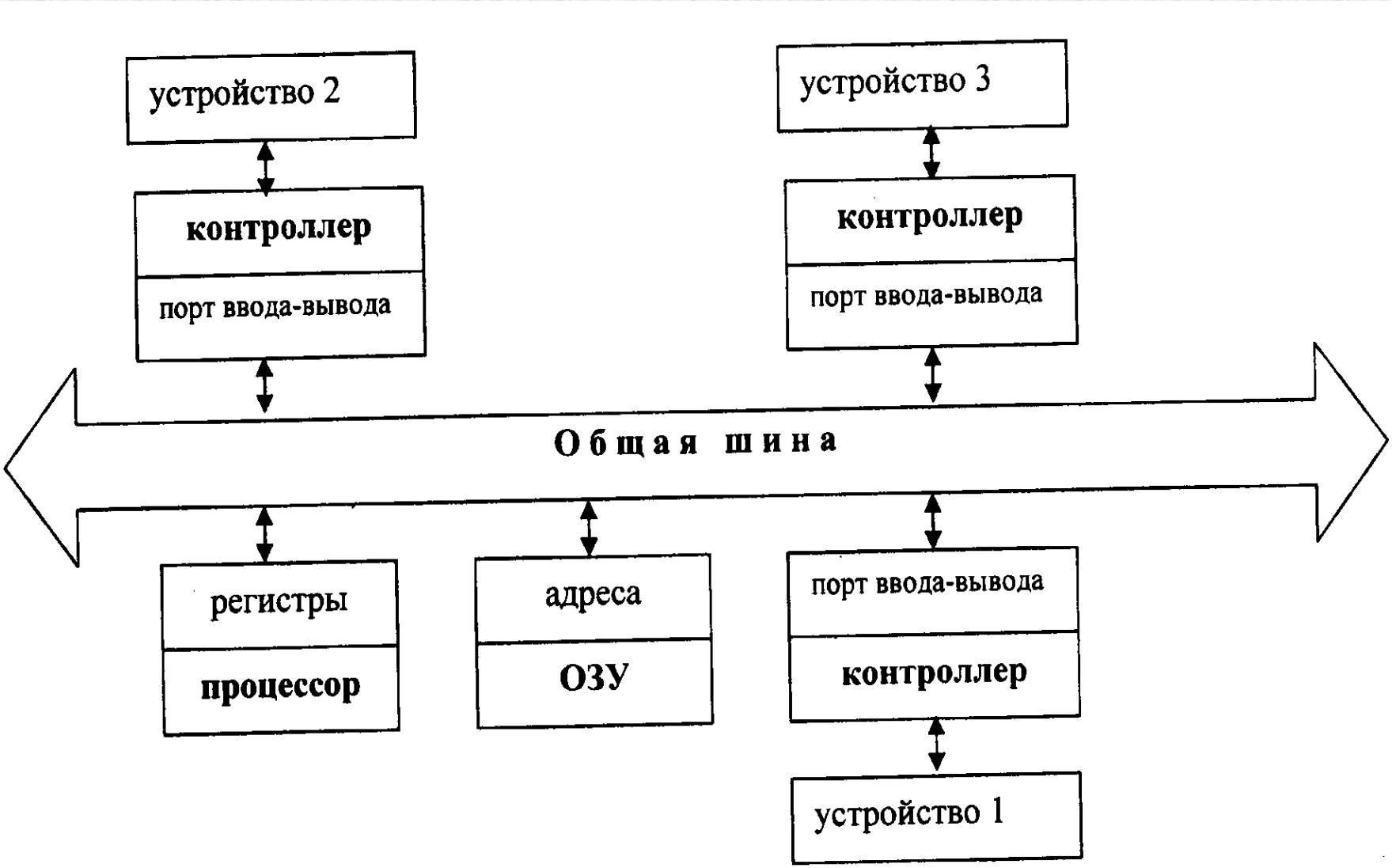
Программно – доступные элементы памяти контроллеров внешних устройств называются **портами ввода – вывода**. Для работы с портами используется пространство логических номеров, называемое **адресным пространством ввода - вывода**.

Часть этого адресного пространства ввода - вывода, выделяемая одному устройству, называется **ресурсами** этого устройства.

Каждый порт в этом пространстве имеет свой уникальный номер – адрес, для хранения которого выделяется 2 байта. Порт контроллера может занимать в пространстве ввода/вывода несколько номеров.

Порт представляет собой пронумерованный путь, по которому передаются данные между процессором и программно-доступными регистрами контроллеров. Для передачи из процессора используется команда OUT, в процессор – команда IN.

Порты ввода-вывода (продолжение)



Пример адресов портов ввода-вывода

Номер порта	Назначение порта
00 00 – 00 1F	Контроллер прямого доступа в память (DMA)
00 60, 00 64	Контроллер клавиатуры
00 61	Динамик
01 F0 – 01 F7	Контроллер первичного жесткого диска
01 70 – 01 77	Контроллер вторичного жесткого диска
03 F0– 03 F5	Контроллер гибкого диска
03 F8 – 03 FF	Первый асинхронный адаптер (COM1)
02 F8 – 02 FF	Второй асинхронный адаптер (COM2)
03 78 – 03 7F	Первый параллельный порт (LPT1)
B0 00 B0 1F	Первый универсальный USB хост-контроллер

Структурная модель контроллера

Каждый контроллер имеет по крайней мере четыре регистра памяти:

- **регистр состояния** - доступен только для чтения и содержит набор битов, информирующих ОС о состоянии устройства (бит занятости, бит готовности, бит ошибки и т.д.);
- **регистр управления** - используется для инициализации и управления режимами работы устройства, а также для хранения кода выполняемой команды;
- **регистр входных данных** - предназначен для хранения данных, передаваемых на устройство;
- **регистр выходных данных** - предназначен для хранения данных, читаемых с устройства.

Для доступа к содержимому этих регистров ОС может использовать один или несколько портов ввода – вывода.