**АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ**
**ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ**
**«Международный Колледж Бизнеса и Дизайна»**
**(АНО ПОО «Международный Колледж Бизнеса и Дизайна»)**

УТВЕРЖДАЮ
Директор АНО ПОО «МКБиД»
_____ Н.Н.Репин
«28» августа 2023 г.

# КОНСПЕКТ ЛЕКЦИЙ

по учебной дисциплине ОП 01 Операционные системы и среды

программы подготовки специалистов среднего звена
по специальности: 09.02.07 «Информационные системы и программирование»

2023 год

**Тема: Введение. Основные понятия. Операционные системы для автономного компьютера. ОС как виртуальная машина. ОС как система управления ресурсами. Эволюция ОС.**

**Цель: изучить основных понятий ОС, классификации и виды ресурсов, рассмотреть периоды развития ОС, вникнуть в смысл использования абстракции в операционных системах.**

1)**введение**

Представите себе простой сценарий начала вашего утра:

1) Проснулись,сходили в уборную
2) Сварили кофе, выпели его
3) Собрались, оделись, сели на автобус ,купили билетик, закомпостировали его и поехали, кто куда, кто в технарь, кто на работу
4) Приехали, вызвали лифт и поехали

**Как вы думаете среди этих сценариев был ли хотя бы один сценарий взаимодействия человека с ОС?**

*Нет, так как мы говорим об операционных системах современных компов, ноутбуков, планшетов и телефонов.*

Даже пример с использованием электронного лифта ,не является, взаимодействием с ОС, в том виде котором мы представляем, НО НА УРАВНЕ АБСТРАКЦИЙ,НА УРОВНЕ ИДЕЙ, было взаимодействие с неким подобием ОС.

**Идея** ОС в том, что бы любой пользователь на интуитивном уровне смог выполнить определенные задачи, не вдаваясь программную реализацию.

**ДЛЯ ЧЕГО Я ВООБЩЕ БЕРУ ЧАЙНИК,ЛИФТ, АВТОБУС, ДЛЯ ТОГО, ЧТОБЫ ПОКАЗАТЬ ВАМ НА НЕ СВЯЗНЫХ ВЕЩАХ ЧЕМ ОТЛИЧАЕТЯ ОС ДЛЯ КОМПОВ,НОУТОВ И.Т.Д ОТ ОС АБСТРАКТОНОЙ.**

**ОС(для понимания)-**это и есть некий такой интерфейс (абстракция), позволяющий нам не задумываться, как все происходит внутри, какие процессы обрабатываются внутри процессора в данный момент, что происходит с памятью, мы занимаемся своими задачами , а ос позволяет не думать нам о сложных внутренностях.

ПРИМЕР: когда мы смотрим любое видео на сайте, ваш браузер получает данные по интернету и выводит их на экран, при этом ваш браузер понятия не имеет ,как там работает сетевая карта, что нужно делать там процессору, что бы все это работало, как процессор должен выводить через системную шину пиксели на экран, чтобы они еще и двигались, браузеру это не нужно знать, нам тем более, в этом и есть прелесть ОС и абстракции, как идее в целом.

Далее мы будем использовать такое понятие, как абстракция, интерфейс-это будут ключевые слова, которыми можно описать ОС.**АБСТРАКЦИЯ**-также это инструмент для упрощения сложности, наращивания не только для ОС ,но и для информатики в целом.

При разработке ОС широко применяется абстрагирование, которое является важным методом упрощения и позволяет сконцентрироваться на взаимодействии высокоуровневых компонентов системы, игнорируя детали их реализации. В этом смысле ОС представляет собой интерфейс между пользователем и компьютером.

С помощью простых и ясных абстракций, скрываются от программиста все ненужные подробности организации системы, работы таймера, управления памятью и т.д. Более того, на современных вычислительных комплексах можно создать иллюзию неограниченного размера оперативной памяти и числа процессоров. Всем этим занимается операционная система.

Таким образом, операционная система представляется пользователю виртуальной машиной, с которой проще иметь дело, чем непосредственно с оборудованием компьютера.

**Операционная система**-это компьютера представляет собой комплекс взаимосвязанных программ, который действует как интерфейс между приложениями и пользователями с одной стороны, и аппаратурой компьютера с другой стороны**. В со-соответствии с этим определением ОС выполняет две группы функций:**

1) предоставление пользователю или программисту вместо реальной аппаратуры компьютера *расширенной виртуальной машины*, с которой удобней работать и которую легче программировать;

2) повышение эффективности использования компьютера путем рационального *управления его ресурсами* в соответствии с некоторым критерием.

Любая операционная система представляет собой разделительный барьер (ОСЬ), посредника между разработчиками программного обеспечения и микроархитектуры.

*В первом случае*, функцией ОС является предоставление пользователю виртуальной машины, которую легче программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальную машину. Современный разработчик прикладных программ может обойтись без досконального знания аппаратного устройства компьютера. Он может даже не знать системы команд процессора.

*Во втором случае*, функцией ОС является распределение ресурсов между процессами, конкурирующими за эти ресурсы. ОС должна управлять всеми ресурсами таким образом, чтобы обеспечить максимальную эффективность ее функционирования. Критерием эффективности может быть, например, пропускная способность или реактивность системы.

Ясно, что средний программист не в состоянии учитывать все особенности работы оборудования (в современной терминологии – заниматься разработкой драйверов устройств), а должен иметь простую высокоуровневую абстракцию, скажем, представляя информационное пространство диска как набор файлов. Файл можно открывать для чтения или записи, использовать для получения или сброса информации, а потом закрывать.

**В зависимости от режимов процессора т.е. должен быть полный доступ(Kernel mode) и частичный доступ (User mode***). Подробней будет в следующей лекции*

**Режимы работы центрального процессора(ЦП):**

1. **Kernel mode (режим ядра или привилегированный режим)-** режим в котором работает ядро ОС или сама ОС, это тот режим когда можно выполнять на железе и на процессоре в частности все, что там физически можно выполнять
2. **User mode (пользовательский режим) –**это тот режим которая предоставляем ОС своим пользователям(людям или программам или всему ,что запускается на этой ОС) так же это режим в котором у нас нет доступа абсолютно ко всем ресурсам, это все та же абстракция, мы можем пользоваться ресурсами, но в порядке котором защищает как железо так и другие процессы.

**Что делать с памятью? Нельзя запретить просто так доступ к памяти или к какой-то части памяти ,нужна некая абстракция для работы с памятью, чтобы несколько процессов могли работать физически с одной памятью, при этом не поломать работу друг друга.**

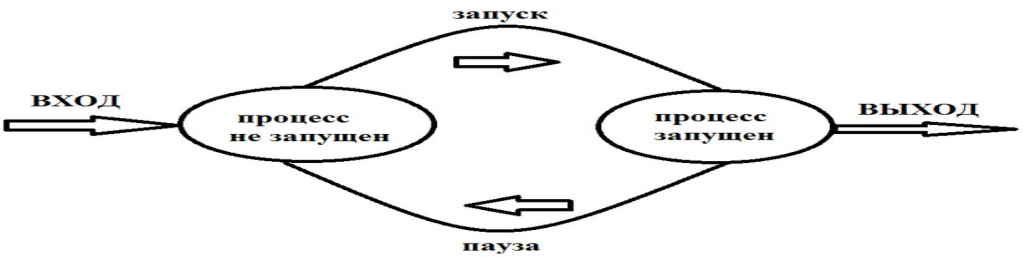
*Прежде чем дальше говорить о памяти рассмотрим что такое процесс.*

*для понимания что такое ПРОГРАММА-ПРОЦЕСС аналогия АВТОМОБИЛЬ-ПОЕЗДКА(автомобиль-это просто набор деталей, когда он стоит, а когда он завелся и движется – тогда это процесс езды-это поездка.. )*

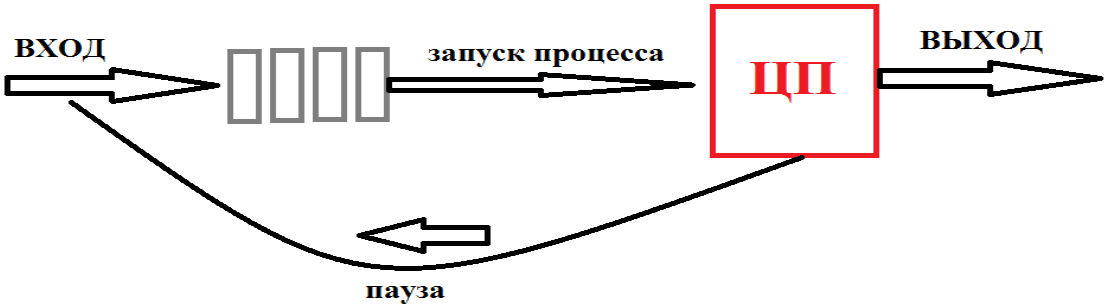
**Процесс**-это программа в исполнении. **Программа**-это просто набор символов.

В конечном итоге это набор инструкций для процессора, а когда инструкции выполняются, (физически выполняются на компе), тогда мы называем это процессом.

ПРОСТАЯ МОДЕЛЬ СОСТОЯНИЯ ПРОЦЕССА




ОЧЕРЕДЬ ПРОЦЕССОВ




**Управление ресурсами включает решение следующих задач**:

**Под ресурсом понимается** любой объект, который может быть использован вычислительным процессом (распределен в процессе вычислений) так же *Ресурсом* является любой объект, который может распределяться внутри системы.

Операционная система не только предоставляет пользователям и программистам удобный интерфейс к аппаратным средствам компьютера, но и является механизмом, распределяющим ресурсы компьютера.

К числу основных ресурсов современных вычислительных систем могут быть отнесены такие ресурсы, основная память, наборы данных, диски, накопители, принтеры, сетевые устройства.

Ресурсы распределяются между процессами. Процесс (задача) представляет собой базовое понятие большинства современных ОС и часто кратко определяется как программа в стадии выполнения. **Программа** — это статический объект, представляющий собой файл с кодами и данными. **Процесс** — это динамический объект, который возникает в операционной системе после того, как пользователь или сама операционная система решает «запустить программу на выполнение», то есть создать новую единицу вычислительной работы.

Например, ОС может создать процесс в ответ, на команду пользователя run name.exe, где name. exe — это имя файла, в котором хранится код программы.

Управление ресурсами вычислительной системы с целью наиболее эффективного их использования является назначением операционной системы.

Например, мультипрограммная операционная система организует одновременное выполнение сразу нескольких процессов на одном компьютере, поочередно переключая процессор с одного процесса на другой, исключая простои процессора, вызываемые обращениями процессов к вводу-выводу.

ОС отслеживает и разрешает конфликты, возникающие при обращении нескольких процессов к одному и тому же устройству ввода-вывода или к одним и тем же данным.

**Управление ресурсами включает решение следующих общих, не зависящих от типа ресурса задач:**

1. - *планирование ресурса* — то есть определение, какому процессу, когда и в каком количестве (если ресурс может выделяться частями) следует выделить данный ресурс;
2. - *удовлетворение запросов на ресурсы*;
3. - *отслеживание состояния и учет использования ресурса* — то есть поддержание оперативной информации о том, занят или свободен ресурс и какая доля ресурса уже распределена;
4. - разрешение конфликтов между процессами.

Задача организации эффективного совместного использования ресурсов несколькими процессами является весьма сложной  В ПЕРВУЮ ОЧЕРЕДЬ ПОЛУЧАЮТ СИСТЕМНЫЕ ЗАДАЧИ,СВОИ РЕСУРСЫ ДАЛЕЕ УЖЕ ПО ТРЕБОВАНИЮ И ОЧЕРЕДИ.

**Классификация ресурсов**

1) Ресурсы могут быть ***разделяемыми***, когда несколько процессов могут их использовать одновременно (в один и тот же момент времени) или параллельно (в течение некоторого интервала времени процессы используют ресурс попеременно)
2) а могут быть и ***неделимыми***.


Рис. 1.1 – Классификация ресурсов

**Основные виды ресурсов:**

1) *аппаратные* – процессоры, память, внешние устройства;

2) *информационные* – данные и программы.

В мультипрограммной системе образуются очереди заявок от одновременно выполняемых

программ к разделяемым ресурсам компьютера: процессору, странице памяти, к принтеру, к диску.

Операционная система организует обслуживание этих очередей по приоритету.

***Ресурс может быть выделен задаче, в следующих случаях:***

**1.)** *ресурс свободен*, и в системе нет запросов от задач более высокого приоритета к запрашиваемому ресурсу;

**2.)** текущий запрос и ранее выданные запросы *допускают совместное использование ресурсов*;

**3.)** ресурс используется задачей низшего приоритета и *может быть временно отобран* (разделяемый ресурс).

***Схема освобождения ресурса:***

После окончания работы с ресурсом задача с помощью специального вызова супервизора посредством соответствующей директивы *сообщает операционной системе об отказе от ресурса, либо операционная система самостоятельно забирает ресурс, если управление возвращается супервизору* после выполнения какой-либо системной функции. Супервизор ОС, получив управление по этому обращению, освобождает ресурс и проверяет, *имеется ли очередь к освободившемуся ресурсу*. При наличии очереди в соответствии с принятой дисциплиной обслуживания и в *зависимости от приоритета заявки* он выводит из *состояния ожидания ждущую ресурс задачу и переводит ее в состояние готовности к выполнению*. После этого *управление либо передается данной задаче, либо возвращается той, которая только что освободила ресурс*.

**Супервизор**— программа, управляющая процессом, памятью и работой оборудования операционной системы.

Таким образом, управление ресурсами составляет важную часть функций любой операционной системы, в особенности мультипрограммной

**Функциональные компоненты ОС для автономного компьютера**

Программы ОС группируются согласно выполняемым функциям и называются подсистемами ОС. Все подсистемы разделяются на два больших класса по следующим признакам:

**1)**по типам локальных ресурсов, которыми управляет ОС; соответствующие подсистемы – *подсистемы управления ресурсами*;

**2)** *по специфические задачи*, применимым ко всем ресурсам; соответствующие подсистемы – *подсистемы, общие для всех ресурсов*.

**Основные подсистемы управления ресурсами** – это подсистемы:

**1)** управления процессами;

**2)** управления памятью;

**3)** управления файлами и внешними устройствами.

**Общие для всех ресурсов –** это подсистемы:

**1)** прикладного программного и пользовательского интерфейсов; (понимаются средства, предоставляемые операционной системой для написании приложений

**2)** защиты данных и администрирования.

Безопасность данных обеспечивается:

**1)**средствами отказоустойчивости ОС (защита от сбоев и отказов аппаратуры и ошибок программного обеспечения);

**2)**средствами защиты от несанкционированного доступа (защита от ошибочного или злонамеренного поведения пользователей системы.

Функции защиты тесно связаны с функциями администрирования, так как именно администратор определяет права и возможности пользователей, отслеживает события, от которых зависит безопасность системы, и поддерживает отказоустойчивость (например, посредством утилит регулярно выполняя операции резервного копирования).

Функции операционной системы автономного компьютера обычно группируются либо в соответствии с типами локальных ресурсов, которыми управляет ОС, либо в соответствии со специфическими задачами, применимыми ко всем ресурсам. Иногда такие группы функций называют подсистемами.

*Наиболее важными функциями ОС по управлению локальными ресурсами автономного компьютера являются*:

1. управление процессами (программами);
2. управление памятью;
3. управление файловой системой;
4. управление устройствами и операциями ввода/вывода;
5. обеспечение защиты данных и администрирование данных;
6. обеспечение интерфейса прикладного программирования; -
7. формирование интерфейса пользователя.

С 90-х годов популярнейшей операционной системой является Windows от Microsoft, а также UNIX-системы, среди которых стоит отдельно выделить Linux и Mac OS (OS X)

разработки Apple. Но все же ОС, которая предназначенная для среднестатистического пользователя-это скорее Windows, с его графической оболочкой, что не маловажно.

**Эволюция ОС**

**1(лампы)**1945г – первые ЭВМ без ОС.

**2(транзисторы)**1955г –нач 60х Усоверш процесс загрузки перфокарт. первый прообраз ОС – системы пакетной обработки .Разделение програмистов, пользователейлей, проектировщиков, специалистов по обслуживанию.

**3(интегральные микросхемы)**1965г Удешевление, появление первых ПК. Появились многозадачные. Появились магнитные ленты, диски. Возможности:

1.Мультипрограммирование

2.Прерывания,

3.реализацияя защитные механизмы,

4. развитие параллелизма в.

4.Разграничение прав пользователя. OS 360

**4(ПК)**1980-ый и до наших дней. совершенствование ПК. Сетевые и распределённые системы.

MS-DOS, UNIX

С 1985 – 1995 Windows по сути является лишь графической оболочкой MS-DOS и усовершенствуются

**Контрольные вопросы:**

1. что такое ОС, какие группы функций она выполняет?
2. Что такое абстракция, как она применяется в ОС?
3. Что такое процесс, программа?
4. Назовите классификацию ресурсов?
5. В каких случаях ресурс может быть выделен задаче?
6. Назовите основные подсистемы управления ресурсами
7. Решение, каких задач включает в себя система управления ресурсами?
8. Назовите и опишите этапы эволюции ОС?

**Список использованных источников:**


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- https://studfiles.net/preview/1872551/


## ЛЕКЦИЯ 2

**Тема: Ядро и вспомогательные модули ОС. Ядро в привилегированном режиме. Многословная структура ОС.**

**Цель: изучить, что такое ядро ОС, из чего оно состоит и какие функции выполняет в ОС. Понять принцип работы многослойной структуры ОС и режимы работы ЦП.**

Когда дело касается ОС, тогда самое важное – это ядро. То что составляет операционную систему-называется ядром. Ядро существует в не пользовательских процессов.

**Ядро Linux** — ядро операционной системы, соответствующее стандартам POSIX, составляющее основу операционных систем семейства Linux. Разработка кода ядра была начата финским студентом Линусом Торвальдсом в 1991 году, на его имя зарегистрирована Торговая марка «Linux».

Код написан в основном на Си с некоторыми расширениями gcc и на ассемблере (с использованием AT&T-синтаксиса GNU Assembler).

Распространяется как свободное программное обеспечение на условиях GNU General Public License, кроме несвободных элементов, особенно драйверов, которые используют прошивки, распространяемые под различными лицензиями

**POSIX** (англ. ***p**ortable **o**perating **s**ystem **i**nterface* —**переносимый интерфейс операционных систем**) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API).

Стандарт создан для обеспечения совместимости различных UNIX-подобных операционных систем и переносимости прикладных программ на уровне исходного кода, но может быть использован и для не-Unix систем.

**Ядро** – часть ОС, рационально размещаемая в ОЗУ, которая работает в привилегированном режиме , модули ОС выполняющие основные ее функции: управление процессами, памятью, устройствами ввода/вывода, данные модули ядра образуют основу ОС, без них она не работоспособна, и не сможет выполнить ни одну из своих функций.

**Наиболее общим подходом к структуризации ОС является разделение всех ее модулей на 2 группы:**

1) **ЯДРО-** модули выполнения основной функции ОС.

Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования - API (Application Programming Interface). По этим инструкциям ядро управляет функциями ОС.

Ядро включает модули, выполняющие основные функции ОС:

1) управление и синхронизация процессов

2)управление памятью

3) управление вводом-выводом

4) файловая система

5) управление прерываниями

**Функции, входящие в состав ядра можно разделить на два класса.**

**1 класс**. Функции для решения внутрисистемных задач организации вычислительного процесса (переключение контекстов процессов, загрузка/выгрузка страниц, обработка прерываний). Эти функции недоступны для приложений.

**2 класс**. Функции для поддержки приложений (доступны приложениям). Эти функции создают для приложений так называемую прикладную программную среду и образуют интерфейс прикладного программирования -API. Приложения обращаются к ядру с запросами - системными вызовами. Функции API обслуживают системные вызовы - предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения.

**Функции модулей ядра** - наиболее часто используемые функции ОС. Скорость выполнения этих функций определяет производительность всей системы в целом. Большинство модулей ядра являются *резидентными* т.е. ( постоянно находятся в ОП)

Остальные модули ОС выполняют полезные, но менее обязательные функции. Например, к таким вспомогательным модулям могут быть отнесены *программы архивирования, дефрагментации диска* и т.п. Вспомогательные модули ОС оформляются либо в виде приложений, либо в виде библиотек процедур и функкций.

2**) МОДУЛИ** выполнения вспомогательной функции ОС.

**Модуль**— функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом или поименованной непрерывной её части.

Модули ядра выполняют такие базовые функции ОС, как управление процессами памятью, ввода-вывода.

*Модули, выполняющие вспомогательные функции:*

1) утилиты
2) библиотеки
3) компиляторы

Ядро работает в привилегированном режиме, и большая часть его модулей постоянно находится в памяти (резидентные).

Разделение ОС на ядро и вспомогательные модули облегчает ее *расширяемость* (Для добавления новой функции, достаточно разработать новое приложение, при этом не требуется модифицировать важные функции ядра ОС.

Внесение изменений в функции ядра - сложнее, но зависит от структурной организации самого ядра. ЯДРО МОЖЕТ БЫТЬ : монолитное(в ядро записаны все программы) и микроядро( записывается только самые важные функции) )

*Вспомогательные модули подразделяются на следующие группы:*

1) *Утилиты* (Сжатие, архивирование, проверка, дефрагментация и пр.)
2) *Системные обрабатывающие программы* (редакторы, отладчики, компиляторы и пр.)
3) *Программы дополнительных услуг* (игры, калькулятор и пр.)
4) *Библиотеки процедур* (математических функций и пр.)
5) *Вспомогательные модули ОС загружаются в оперативную память только на время выполнения* (транзитные модули) т.е находятся в ОП по требованию

Обратим внимание, вспомогательные модули ОС обращаются к функциям ядра, как и обычные приложения, посредством системных вызовов. Разделение ОС на ядро и модули обеспечивает легкую расширяемость. Дополнительные модули ОС обычно загружаются в оперативную память только на время выполнения, т.е. являются транзитивными.

Вспомогательные модули, в отличие от модулей ядра, являются транзитными.

*Обратим внимание* на то, что многие модули ОС оформлены как обычные приложения. Решение о том, является ли какая-либо программа частью ОС или нет, принимает производитель ОС. Некоторая программа может существовать определенное время как пользовательское приложение, а потом стать частью ОС (*например, Web-браузер компании Microsoft – сначала поставлялся как отдельное приложение, затем стал частью ОС Windows*).

**Многослойная структура ОС**

Аппаратура компьютера должна поддерживать как минимум два режима работы: режим пользователя и привилегированный режим.

ОС должна работать с исключительными полномочиями, для того чтобы играть роль арбитра в споре приложений за ресурсы компьютера (кому и в каком количестве отдать ресурсов).

Для этого используют многослойную структуру.

Таким образом, вычислительную систему, работающую под управлением ОС на основе ядра, можно рассматривать как систему, из трех иерархически расположенных слоев:

1)нижний слой образует аппаратура,

2)промежуточный - ядро,

3)утилиты, обрабатывающие программы и приложения, составляют верхний слой.

При такой организации приложения не могут напрямую взаимодействовать с аппаратурой, а только через операционную систему и утилиты.

ядро является Осью связующим звеном между аппаратурой и верхним слоем(утилитами ,библиотеками и сис. программами)

Многослойную структуру вычислительной системы принято изображать в виде системы концентрических окружностей, иллюстрируя тот факт, что каждый слой может взаимодействовать только со смежными слоями. Действительно, при такой организации ОС приложения не могут непосредственно взаимодействовать с аппаратурой, а только через слой ядра.

**Режимы работы центрального процессора(ЦП):**

Так как ядро выполняет все основные функции ОС, то чаще всего именно ядро - та, часть ОС, которая работает в привилегированном режиме, а приложения – в пользовательском режиме.

Чаще всего железо не имеет никакой защиты т.е. мы можем запустить плохие деструктивные инструкции на процессоре, а он не будет этому противится. То что мы не можем чаще всего этого сделать (пользователи компа) и то что наши программы не могут взять и уничтожить жесткий диск, хотя физически это возможно, можно послать этот сигнал-и уничтожить ОС, одна из функций ОС-это ЗАЩИТА , но самой ОС нужен полный доступ, полный контроль над железом и по этому придумали два режима работы процессора


3. **Kernel mode (режим ядра или привилегированный режим)-** режим в котором работает ядро ОС или сама ОС, это

тот режим когда можно выполнять на железе и на процессоре в частности все, что там физически можно выполнять

*Режим ядра*— это привилегированный режим работы, в котором код имеет прямой доступ ко всем аппаратным ресурсам и всей памяти, включая адресные пространства всех процессов режима пользователя

Необходимо обратить внимание, на то, что работа системы с привилегированным ядром замедляется за счет замедления выполнения системных вызовов.

Системный вызов привилегированного ядра инициирует переключение процессора из пользовательского режима в привилегированный, а при возврате к приложению - переключение из привилегированного режима в пользовательский.

4. **User mode (пользовательский режим) –**это тот режим которая предоставляем ОС своим пользователям(людям или программам или всему ,что запускается на этой ОС) так же это режим в котором у нас нет доступа абсолютно ко всем ресурсам, это все та же абстракция, мы можем пользоваться ресурсами, но в порядке котором защищает как железо так и другие процессы.

*Режим пользовател*я — менее привилегированный по сравнению с режимом ядра режим работы процессора. Он не имеет прямого доступа к аппаратуре. Выполняющийся в этом режиме код непосредственно имеет дело лишь с объектами своего адресного пространства.

Каждое приложение пользовательского режима работает в своем адресном пространстве и защищено тем самым от вмешательства других приложений. Код ядра имеет доступ к областям памяти всех приложений, но сам полностью от них защищен. Приложения обращаются к ядру с запросами на выполнение системных функций.

**Контрольные вопросы:**

1. Что такое Ядро, какие функции выполняет?
2. На какие основные группы модулей подразделяется ОС?

3. Что такое вспомогательные модули, за что они отвечают в ОС?
4. Что такое модули ядра, что они отвечают в ОС?
5. Назовите функции входящие в состав ядра, на какие классы подразделяются?
6. Опишите принцип работы многослойной структуры
7. Назовите и опишите два режима работы ЦП

**Список использованных источников:**


1. Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
2. Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
3. Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
4. https://studopedia.su/13_98782_mnogosloynaya-struktura-os.html
5. https://studfiles.net/preview/1774747/page:10/
6. https://studfiles.net/preview/1774747/page:9/


## ЛЕКЦИЯ 3

**Тема: архитектура ядер ОС. Микро ядерная архитектура. Концепция. Преимущества и недостатки микро ядерной архитектуры.**

**Цель: изучить основные направления в дизайне ядер (архитектур) ОС. Изучение концепции микро ядерной и гибридной архитектуры ОС.**

Микроядерные Операционные Системы (ОС) привлекают внимание разработчиков и исследователей последние 30 лет. Действительно,

1) с одной стороны эти системы обеспечивают жесткое разделение исполнимых модулей, вынесение всех служб из ядра в пространство пользователя, способствует повышение отказоустойчивости и безопасности системы.
2) С другой стороны издержки, возникающие из-за поддержки такой распределенной структуры (вызванные частыми переключениями контекста) требуют дополнительных вычислительных ресурсов.

Это противоречие заставляет ученых и разработчиков искать новые архитектурные решения для минимизации негативных и усиления позитивных сторон микроядерных ОС

**Ядро** – наиболее защищенная часть ОС. В нем сосредоточены основные функции данной ОС

**ядро ОС выполняет в компьютере следующие функции:**

**1.** Оно все время находится в оперативной памяти, так как координирует работу системы;

**2.** Ядро является центральной частью любой операционной системы, поэтому управляет деятельностью всех составляющих ОС;

**3.** Оно содержит в себе драйверы устройств, программы для управления памятью, планировщик заданий;

**4.** Отвечает за осуществление системных вызовов.

Все эти задачи могут быть реализованы различными путями. Поэтому и типов ядер ОС бывает несколько. Выделяют множество разновидностей ядер ОС.

**Основных направлении в дизайне ядер(архитектуры) ОС :**

1) **МОНОЛИТНОЕ ЯДРО-** говорит о том, что все ОС должна быть одной большой программой и максимально все ,что можно туда записать, должно быть записано.

Та грань между ОС и не ОС, довольно абстракта и зависит от того, как дизайнер той ОС решили все скомпоновать.

ПРИМЕР: драйвера для принтер ,графические драйвера, для каких то ОС-будет является частью самой ОС, а для других будут является просто пользовательскими приложениями.

Так вот в монолитном ядре мы пытаемся как можно больше полезных функций и приложений записать в режиме ядра, когда ОС имеет доступ к ресурсам компа и это позволяет сделать ядро довольно быстры, так как это одно программа, записано все вместе, рядом.

Недостаток: если в части ОС произойдёт ошибка, то все ядро выйдет из строя, решением этой проблемы является микроядро.

МОНОЛИТНОЕ ЯДРО Монолитное ядро предоставляет богатый набор абстракций оборудования. Все части монолитного ядра работают в одном адресном пространстве. Это такая схема операционной системы, при которой все компоненты её ядра являются составными частями одной программы, используют общие структуры данных и взаимодействуют друг с другом путём непосредственного вызова процедур. Монолитное ядро — старейший способ организации операционных систем. Примером систем с монолитным ядром является большинство UNIX-систем.

**Достоинства:** высокая скорость работы, простая разработка модулей.

**Недостатки**: Поскольку всё ядро работает в одном адресном пространстве, сбой в одном из компонентов может нарушить работоспособность всей системы. Примеры: Традиционные ядра UNIX (такие как BSD), Linux; ядро MS-DOS, ядро KolibriOS.

2) **МОДУЛЬНОЕ ЯДРО** – подвид монолитного ядра. Здесь более гибкая связь между ядром и приложениями, это все не записано в ядро, а уже имеет подразделение.

Является более совершенной модификацией монолитного ядра. Оно не представляет собой одну программу, а состоит из отдельных модулей. Нарушение работы одного из них никоим образом не сказывается на работе остальных. При подключении к процессору нового аппаратного обеспечения, в случае с монолитным ядром ОС, потребовалась бы его полная перестройка. В случае же с модульными ядрами достаточно лишь подгрузить к основному набору новый модуль. Это можно сделать как непосредственно во время работы на ПК или ноутбуке, так и с его перезагрузкой.

3) **МИКРОЯДРО-** вместо того, чтобы запихивать все, мы добавим, только самое необходимое для работы ОС.

С точки зрения пользователя те же самые дрова для принтера и графики будут, казаться частью ОС, но на самом деле будут такими же программами, как и те программы которые напишем мы. Это позволит минимизировать риск, если в части ядра произойдёт ошибка, это не значит, что ядро перестанет работать.

ПРИМЕР: в Linux графическая часть не входит в ядро, так что если, что то пошло не так, графа ляжет, а ядро будет не затронуто.

Недостаток: это скорость работы. Теперь большее расстояние между частями ОС им приходится общаться между собой посылая сигналы или же какого либо связующего интерфейса.

Представляет только основные функции управления процессами и минимальный набор для работы с оборудованием. Классические микроядра дают очень небольшой набор системных вызовов.

**Достоинства:** устойчивость к сбоям и ошибкам оборудования и компонентов системы, высокая степень ядерной модульности, что упрощает добавление в ядро новых компонентов и процесс отладки ядра. Для отладки такого ядра можно использовать обычные средства. Архитектура микроядра увеличивает надежность системы.

**Недостатки**: Передача информации требует больших расходов и большого количества времени.

Экзоядро. Такое ядро ОС, которое предоставляет лишь функции взаимодействия процессов, безопасное выделение и распределение ресурсов. Доступ к устройствам на уровне контроллеров позволяет решать задачи, которые нехарактерны для универсальной ОС.

4) **НАНОЯДРО-** выполняет обработку аппаратных прерываний посылаемых устройствами ПК

**Прерывание**— сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания.

Такое ядро выполняет только единственную задачу- обработку аппаратных прерываний, образуемых устройствами ПК. После обработки наноядро посылает данные о результатах обработки далее идущему в цепи программному обеспечения при помощи той же системы прерываний.

5) **ГИБРИДНОЕ ЯДРО**- это современная модификация микроядра объединяет их достоинства и минимизирует недостатки.

Позволяет некоторой части функций проникать в само ядро не нарушая его архитектуры и не выводит его из строя, но при этом оставаться защищенным.

Модификация микроядер, позволяющая для ускорения работы впускать несущественные части в пространство ядра. На архитектуре гибкого ядра построены последние операционные системы от Windows, в том числе и Windows 7, 10 используют гибридную архитектуру. .

Смешанное ядро, в принципе, должно объединять преимущества монолитного ядра и микроядра: казалось бы, микроядро и монолитное ядро — крайности, а смешанное — золотая середина. В них возможно добавлять драйверы устройств двумя способами: и внутрь ядра, и в пользовательское пространство. Но на практике концепция смешанного ядра часто подчёркивает не только достоинства, но и недостатки обоих типов ядер.

В ходе своей работы использует так называемые «несущественные» части системы, за счет чего увеличивается скорость процессов. Используется абстракция.

Все перечисленные подходы к построению ОС безусловно имеют как преимущества, так и недостатки. Поэтому в большинстве современных операционных системах используют различные комбинации подходов к построению. Обычно за основу берут один из подходов и дополняют в него элементы других подходов, стараясь свести к минимумы недостатки.

**Микро ядерная архитектура**

В привилегированном режиме остается работать только маленькая часть ОС, называемая микроядром. Все остальные функции ядра оформляются в виде приложений, работающих в пользовательском режиме.

Микроядро работает в привилегированном режиме и обеспечивает взаимодействие между программами, планирование использования процессора, первичную обработку прерываний, операции ввода-вывода и базовое управление памятью. Остальные компоненты системы взаимодействуют друг с другом путем передачи сообщений через микроядро.

**Основное достоинство микроядерной архитектуры** – Операционные системы, основанные на микроядерной архитектуре, удовлетворяют большинству требований, предъявляемым к современным ОС:

1) **обладают переносимостью**

(весь машинно-зависимый код изолирован в микроядре необходимо мало изменений при переносе системы на новый процессор, к тому же все изменения сгруппированы вместе)

2) **высокая степень расширяемости**

(для того, чтобы добавить новую подсистему требуется разработать новое приложение, для чего не требуется затрагивать микроядро; с другой стороны, пользователь легко может удалить ненужные подсистемы, удалять из ядра было бы сложнее)

3) **достаточная надежность**

(каждый сервер ОС выполняется в своем адресном пространстве и таким образом защищен от других серверов, кроме того, если происходит крах одного сервера, он может быть перезапущен без останова или повреждения других серверов).

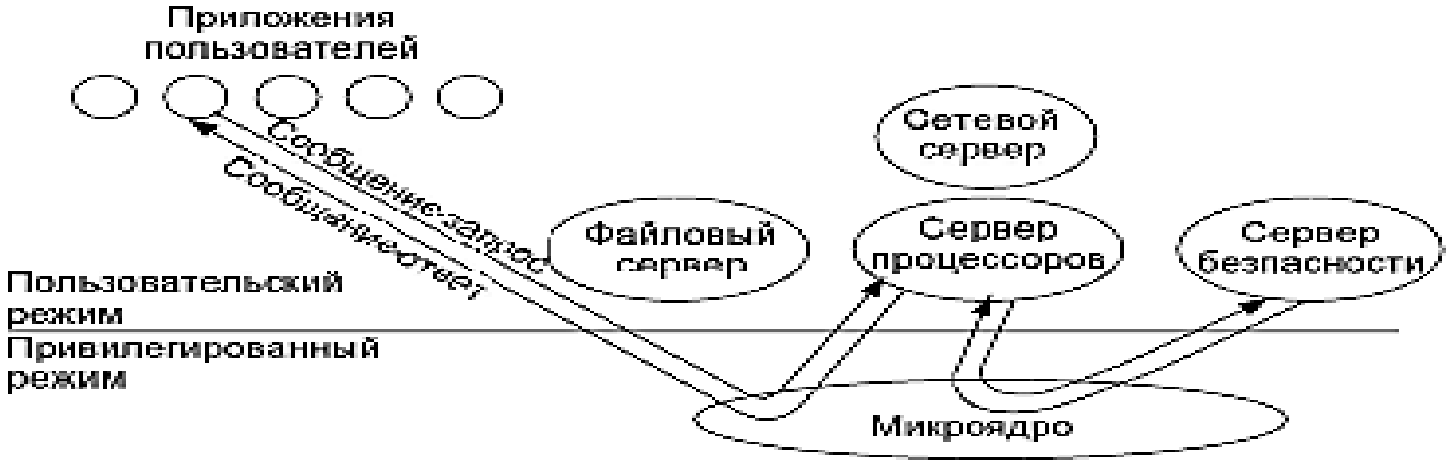
**4)** **поддержка распределенных вычислений**

(серверы ОС могут работать на разных компьютерах - асимметричная ОС; возможен перенос однопроцессорных ОС на распределенные компьютеры)

! **Основной (и по сути единственный) недостаток микроядерной архитектуры** -

**1)** снижение производительности.

Для того чтобы микроядерная операционная система по скорости не уступала операционным системам на базе монолитного ядра, требуется очень аккуратно (продуманно) проектировать разбиение системы на компоненты, стараясь минимизировать взаимодействие между ними. Таким образом, основная сложность при создании микроядерных операционных систем необходимость очень аккуратного проектирования.




Реализация системного вызова в микроядерной архитектуре

**Гибридная архитектура ОС**

Гибридное ядро сочетает в себе элементы

-микро ядерной и

-монолитной архитектуры.

В таких ситуациях значительная часть функций, традиционно выполняемых ядром операционной системы, реализована в её монолитном ядре, однако некоторые реализованы в виде отдельных программ, исполняемых в своих собственных адресных пространствах, что соответствует идеям микро ядерной архитектуры.

Например, из монолитного ядра можно вынести драйверы устройств, но оставить все остальные компоненты.

*К гибридным ядрам относят MacOS X*, поскольку значительная часть системных служб и драйверов данных систем реализуется в виде процессов пользовательского режима, то есть активно используется архитектура микроядра.

Большинство гибридных ядер поддерживают архитектуру подгружаемых модулей ядра, но есть **важное отличие от монолитно модульных ядер** - подгружаемые модули ядра и прочие компоненты гибридных ядер располагаются в вытесняемой памяти и взаимодействуют друг с другом путем передачи сообщений, как положено в микроядерных операционных системах. Кроме того, в **отличии от микроядер**, как правило, большинство компонентов ядра работают в одном адресном пространстве. Таким образом гибридные ядра имеют отличия как от монолитных и модульных ядер, так и от микроядер.

**Контрольные вопросы:**

1. Назовите основные направления в дизайне ядер ОС
2. Опишите каждую архитектура, назовите их достоинства и недостатки
3. Назовите различи я в концепциях миро ядерной и гибридной архитектуры.

**Список использованных источников:**


1. Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
2. Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
3. Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
4. http://linuxguru.ru/how-it-works/vidy-arxitektury-yader-operacionnyx-sistem/
5. https://www.intuit.ru/studies/courses/2192/31/lecture/968?page=4
6. http://mf.grsu.by/UchProc/livak/po/lections/lec_arhit.htm

# ЛЕКЦИЯ №4

**ТЕМА: Понятия «процесс» и «поток». Модели состояний процесса**

**ЦЕЛЬ: рассмотреть понятия процесс и поток, изучить принцип построения и работы модели состояний процессов**

Напомним из предыдущих лекций!

**Программа**-это набор инструкций для процессора(набор кода),чаще всего это файл записанный где то в памяти и его можно запустить, а вот когда мы его запускаем ОС дает этой программе свободу, дает ей доступ к железу, компу, к ресурсам, тогда эта программа может работать, так вот в процесса когда эта программа запущена, потребляет ресурсы, память, тогда это есть **процесс** ( программа в исполнении)

*Ну или как мы уже говорили на примере машина и поездка, процесс-это поездка, программа-это сама машина(груда железа).*

Мы много говорили о функциях ОС, что их много, что доступ к компу, доступ к железу- это создание некой абстракции, это одна из основных задач ОС, но естественно этого не достаточно.

1. Если ОС дает доступ к ресурсам компьютера, каким то программа, то просто так остановится на этом сегодня уже нельзя.

Когда только появлялись компы, тогда был один процесс в один момент времени, и нам не нужно было думать о том, что какой процесс к чему имеет доступ. У нас был всего лишь один процесс в любой момент времени и можно было не волноваться, что он там заденет какой то другой процесс. Когда этот процесс завершался, можно было забыть обо всем, и запустить новый процесс и проблем не было.

Сейчас же появились и желания и возможности запускать сразу несколько процессов на выполнение, и стала задача, как их распределять.

Термин «процесс» впервые появился при разработке операционной системы Multix и имеет несколько определений, которые используются в зависимости от контекста, согласно которым **процесс — это:**

**1.** программа на стадии выполнения

**2.** «объект», которому выделено процессорное время

**3.** асинхронная работа

**Примером** контекстной зависимости применения термина «процесс» является ОС Android. В этой системе процесс и приложение не являются тесно связанными сущностями:

1. программы для андроид могут выглядеть выполняющимися при отсутствии процесса;
2. несколько программ могут использовать один процесс;
3. либо одно приложение может использовать несколько процессов;
4. процесс может присутствовать в системе даже если его приложение бездействует.

Отметим, что в этом нет никаких противоречий с теорией операционных систем — объектные библиотеки и загружаемые модули тоже являются процессами.

Каждый процесс имеет как минимум один поток. Также каждый процесс имеет свое собственное виртуальное адресное пространство и контекст выполнения, а потоки одного процесса разделяют адресное пространство процесса.

**Пото́к выполне́ния** -наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы.

**Поток** — определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса, это изменение сразу же становится видно другим потокам этого процесса.

Реализация потоков выполнения и процессов в разныхоперационных системах отличается
друг от друга, но в большинстве случаев поток выполнения находится внутри процесса.
Несколько потоков выполнения могут существовать в рамках одного и того же процесса и
совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов.

**Диспетчеризация**

**Диспетчер –** отправляет процессы на выполнение, выделяет время ЦП и переключает ЦП с одного процесса на другой.(наш диспетчер задач)

С его помощью можно в режиме реального времени отслеживать выполняющиеся приложения и запущенные процессы, оценивать загруженность системных ресурсов компьютера и использование сети.

В любой момент времени любой процесс может находиться в каком-либо состоянии: как минимум это ожидание ввода/вывывода, выполнение, готовность к выполнению, и еще можно придумать ряд дополнительных промежуточных состояний.

**Рассмотрим две модели состояний процессов**.

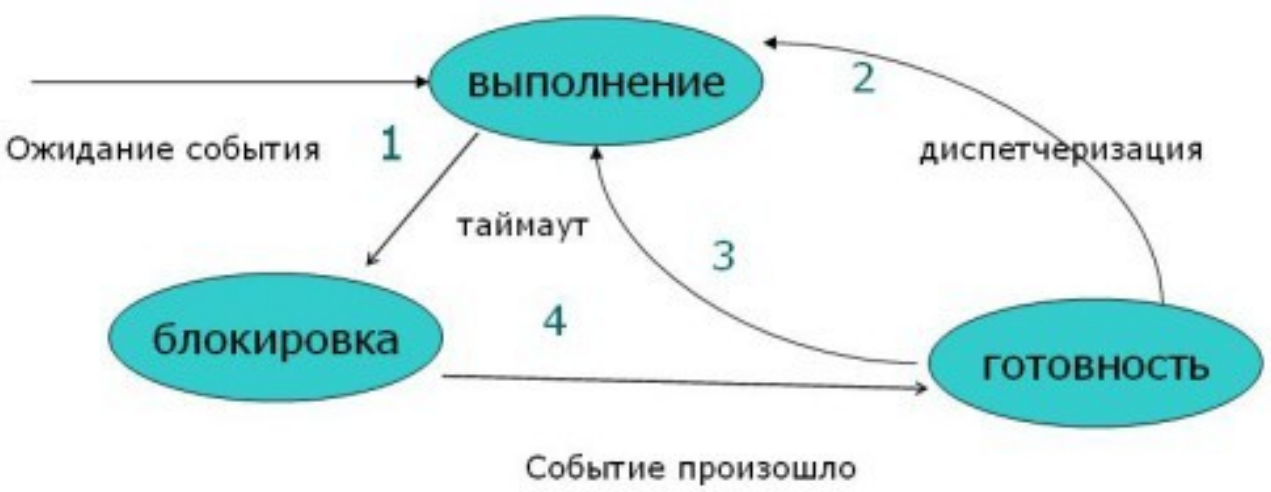
**- кольцевая модель состояний процесса**

**-расширенная модель состояний процесса**

Конечная цель любой операционной системы – выполнить какую-либо работу, задачу, ответить на запрос пользователя...

Можно выделить три основные состояния процесса

1. Выполнения (исполняется на ЦП);
2. Готовности (временно остановлен);
3. Блокировки (ожидает внешнего события);




Кольцевая модель состояний процесса

**Когда процесс может перейти в состояние готовности?**

Предположим, что наш процесс выполнялся до ввода данных. До этого момента он был в состоянии выполнения, потом перешел в состояние ожидания(блокировки)— ему нужно подождать, пока мы введем нужную для работы процесса информацию. Затем процесс хотел бы уже перейти в состояние выполнения, так как все необходимые ему данные уже введены, но он попадает в состояние блокировки: так как он не единственный процесс в системе, пока он был в состоянии ожидания(блокировки), его «место под солнцем» занято — так как

процессор выполняет другой процесс. После того , когда данные от пользователя или системы уже поступили процесс переходит на стадию ГОТОВНОСТИ: ждать ему нечего, а выполняться он тоже не может.

Из состояния готовности процесс может перейти только в состояние выполнения, после того, как квант времени исчерпается или же прейдет очередь определенного процесса. В состоянии выполнения может находится только один процесс на один процессор. Если у вас n-процессорная машина, у вас одновременно в состоянии выполнения могут быть n процессов.

Из состояния выполнения процесс может перейти либо в состояние ожидания(блокировки), либо в состояние готовности.

*Почему процесс может оказаться в состоянии ожидания(блокировки)*, мы уже знаем — ему просто нужны дополнительные данные или он ожидает освобождения какого-нибудь ресурса, например, устройства или файла. *В состояние готовности процесс* может перейти, если во время его выполнения, квант времени выполнения «вышел».

**Выполнение** — это активное состояние, во время которого процесс обладает всеми необходимыми ему ресурсами. В этом состоянии процесс непосредственно выполняется процессором.

**блокировка** — это пассивное состояние, во время которого процесс заблокирован и не может быть выполнен, потому что ожидает какое-то событие, например, ввода данных или освобождения нужного ему устройства.

**Готовность** — это тоже пассивное состояние, процесс тоже заблокирован, но в отличие от состояния ожидания, он заблокирован не по внутренним причинам (ведь ожидание ввода данных — это внутренняя, «личная» проблема процесса — он может ведь и не ожидать ввода данных и свободно выполняться — никто ему не мешает), а по внешним, независящим от процесса, причинам.

**Любой процесс находится в нескольких состояниях, в самом простом варианте можно выделить три состояния:**

1. Предположим, что любой процесс начинает свою “жизнь” с состояния **«Выполнение»**.

2. Если был сделан запрос на ввод/вывод, то он осуществляется с некоторой задержкой, ибо ввод/вывод работает медленней чем ЦП, значит появляется некоторое **ожидание** события (т.е. ожидание ввода/вывода).

Процесс в это время находится в заблокированном состоянии **«Блокировка».** Ему диспетчер не выделяет времени ЦП, процесс не может выполнять какую-либо работу пока не выполнится ввод/вывод.

3. Далее, когда произошел ввод/вывод процесс уже может быть обработаться, его состояние изменяется в состояние **«Готовность»** (процесс должен показать диспетчеру о своей готовности выполняться)**.**

Диспетчер при очередном переключении между процессами видит, что есть процесс в состоянии «Готовность» и переключает его в состоянии «**Выполнение**».

В этом состоянии «Выполнение», диспетчер передает процессу квант времени ЦП – начинается непосредственное выполнение. Этот круг замкнутый.

4. Еще одно состояние, когда процесс попадает в «Готовность» — когда квант процессорного времени, отведенный на выполнение истекает, то происходит так называемый **таймаут,** процесс из состояния **«Выполнение»** переходит в **«Готовность»** и диспетчер в это время передает управление другому процессу.

Диспетчеру системы нужно знать, что происходит с общими событиями: с вводом/выводом, синхронизацией, др.

**Контрольные вопросы:**

1. что такое процесс?
2. Что такое поток?
3. Что такое диспетчер, чем он отличается от диспетчеризации?
4. Опишите работу модели трех состояний
5. Когда процесс может перейти в состояние готовности?
6. Почему  процесс может оказаться в состоянии ожидания?


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Учебно-методические материалы для студентов кафедры АСОИУ
  Источник: http://www.4stud.info/rtos/lecture2.html
- Иерархии процессов
  Источник: http://os-pc.ru/proc/83-ierarkhii-protsessov.html
- Планирование и диспетчеризация процессов и потоков. Вытесняющие и невытесняющие алгоритмы планирования
  Источник: http://5fan.ru/wievjob.php?id=24512


## ЛЕКЦИЯ №5

**ТЕМА: Модель состояний процесса из 5-ти составляющих. Планировщик заданий. Иерархия процессов в системах.**

**ЦЕЛЬ: рассмотреть операции, которые можно производить над процессами. Изучить принцип построения и работы модели состояний процессов из 5-ти элементов. Понять принцип построения иерархии процессов.**

Как мы уже говорили есть кольцевая модель, а есть модифицируемая версия(гибрид). Оно повторяет предыдущую модель только несколько модифицируемая.

Перейдем к рассмотрению второй модели состояний процесса.

**Вторая модель состояний процесса**

появилось два дополнительных состояния: рождение процесса и смерть процесса.


Новая модель состоит из **пяти состояний**, эта модель очень близка к сегодняшним ОС.

1. **Новый процесса** — это пассивное состояние, когда самого процесса еще нет, но уже готова структура для появления процесса (некий шаблон для процесса),но еще не загрузился память(создано пустое адресное пространство).

2. Если новый процесс принимается ОС, если все соблюдается, все права доступа, то процесс помещается в состояние «**Готовность**»: процесс полностью готов к выполнению, т.е. может получить управление и непосредственно начать работать. Все загружено в память, инициализированы данные, стек, куча.

3. «**Выполнение**» — процесс исполняется.

4. «**Блокировка**» — процесс ожидает внешнего события (ввода/вывода).

5. **Завершенный процесса** — процесс удаляется из пула выполненных процессов, он закончил работу. Процесс помечается как «завершенный». Самого процесса уже нет, но может случиться, что его «место", то есть структура данных , осталась в списке процессов. Такие процессы называются зобми.

**Диспетчер** будет выполнять работу по очистке процесса. На данном этапе проходит работа по освобождению памяти, закрытию ресурсов процесса (вв/выв, файлов…).

В операционной системе есть специальная программа — **планировщик**, которая следит за тем, чтобы все процессы

выполнялись отведенное им время. Так же мы сами можем создавать процессы и задавать и время выполнения.

**планировщик заданий** - **это** оснастка MMS(сервис мультимедийных сообщений), позволяющая назначать автоматически выполняемые **задания**, запуск которых производится в определенное время или при возникновении определенных событий.

**Например,** у нас есть три процесса.

1) Один из них находится в состоянии выполнения.

2) Два других — в состоянии готовности.

Планировщик следит за временем выполнения первого процесса, если «время вышло», планировщик переводит процесс 1 в состояние готовности, а процесс 2 — в состояние выполнения. Затем, когда, время отведенное, на выполнение процесса 2, закончится, процесс 2 перейдет в состояние готовности, а процесс 3 — в состояние выполнения.

В ОС время перехода процесса из одного состояния в другое должно быть детерминированно.

детерминированный **процесс** — это Процесс, исход которого полностью определен алгоритмом, значениями входных переменных и начальным состоянием системы.

Функции контроля за временем т.е. дедлайн (deadline) возлагаются на планировщика

1. В командной строке введите **Taskschd.msc**. или через панель управления СИСТЕМА И БЕЗОПАСНОСТЬ=>АДМИНИСТРИРОВАНИЕ=> ПЛАНИРОВЩИК ЗАДАНИЙ

«Планировщик заданий», включая создание, удаление, выполнение запросов, изменение, запуск и завершение назначенных заданий на локальном или удаленном компьютере. Это средство находится в папке %SYSTEMROOT%\System32.

*Создаем задачу(запуск скрипта стрельбы по мишени при обнаружении опасности)*

*1.**триггер**(при создании можно указать условия ее запуска) запускать, когда мне грозит опасность т.е. уничтожение моего персонажа.*

*2.**действия** (при создании необходимо указать действия, которые будет выполняться при ее запуске)*

*2.1 может быть отправка смс мне с текстом*

*2.2 запуск программы*

*2.3 отправка почты*

*Я указываю запуск моего скрипта защиты и добавляю сценарий после которого мне придет на почту смс с текстом завершении задачи*

*3.**условия** (указать условия, которые вместе с триггерами будут определять необходимость выполнения задачи)*

*3.1 запускать только при подключении к сети*

*4.параметры(дополнительные параметры выполнения задачи)*

*4.1 выполнять задачу по требованию(когда грозит опасность персонажу)*

*4.2 при сбое перезапустить скрипт*

*4.3 задаю время выполнения для задачи(играю 1 неделю, скрип работает 1 неделю)*

**Операции над процессами**

Над процессами можно производить следующие операции:

1. **Создание процесса** — это переход из состояния рождения в состояние готовности
2. **Уничтожение процесса** — это переход из состояния выполнения в состояние смерти
3. **Восстановление процесса** — переход из состояния готовности в состояние выполнения
4. **Изменение приоритета процесса** — переход из выполнения в готовность
5. **Блокирование процесса** — переход в состояние ожидания из состояния выполнения
6. **Пробуждение процесса** — переход из состояния ожидания в состояние готовности
7. **Запуск процесса** (или его выбор) — переход из состояния готовности в состояние выполнения

**Для создания процесса операционной системе нужно:**

1. Присвоить процессу имя
2. Добавить информацию о процессе в список процессов
3. Определить приоритет процесса (могу задать наивысшие правы выполнения)
4. Сформировать блок управления процессом
5. Предоставить процессу нужные ему ресурсы

**Иерархия процессов**

В некоторых системах, когда процесс порождает другой процесс, родительский и дочерний процессы продолжают быть определенным образом связанными друг с другом. Дочерний процесс может и сам создать какие-либо процессы, координируя иерархию процессов, которые он создал, при этот быть зависимым от родительского процесса.

Следует заметить, что в отличие от растений и животных, использующих половую репродукцию, у процесса есть только один родитель, но ноль, один, два или более детей.

В UNIX процесс, все его дочерние процессы и более отдаленные потомки образуют группу процессов. Когда пользователь отправляет сигнал с клавиатуры, тот достигает всех участников этой группы процессов, связанных на тот момент времени с клавиатурой (обычно это все действующие процессы, которые были созданы в текущем окне). Каждый процесс по отдельности может захватить сигнал, игнорировать его или совершить действие по умолчанию, которое должно быть уничтожено сигналом.

В качестве другого примера, поясняющего ту роль, которую играет иерархия процессов, давайте рассмотрим, как UNIX инициализирует саму себя при запуске.

В загрузочном образе присутствует специальный процесс, называемый **init** *В начале своей работы он считывает файл, сообщающий о количестве терминалов. Затем он разветвляется, порождая по одному процессу на каждый терминал. Эти процессы ждут, пока кто-нибудь не*

*зарегистрируется в системе.* Если регистрация проходит успешно, процесс регистрации порождает оболочку для приема команд. Эти команды могут породить другие процессы, и т. д. Таким образом, все процессы во всей системе принадлежат единому дереву, в корне которого находится процесс init.

В отличие от этого в Windows не существует понятия иерархии процессов, и все процессы являются равнозначными. Единственным намеком на иерархию процессов можно считать присвоение родительскому процессу, создающему новый процесс, специального маркера (называемого дескриптором), который может им использоваться для управления дочерним процессом. Но он может свободно передавать этот маркер какому-нибудь другому процессу, нарушая тем самым иерархию. А в UNIX процессы не могут лишаться наследственной связи со своими дочерними процессами.

Процесс не может взяться из ниоткуда: его обязательно должен запустить какой-то процесс.

Процесс, запущенный другим процессом, называется дочерним (child) **процессом или потомком**.

Процесс, который запустил новый процесс называется родительским (parent), **родителем** или просто — предком.

**У каждого процесса есть два атрибута** — PID (Process ID) - идентификатор процесса и PPID (Parent Process ID) — идентификатор родительского процесса.

**Процессы создают иерархию в виде дерева**. Самым «главным» предком, то есть процессом, стоящим на вершине этого дерева, является процесс **init (PID=1).**

**Контрольные вопросы:**

7. Что такое планировщик заданий?
8. Какие операции можно произвести над процессами?
9. Что нужно для создания процесса ОС?
10. Опишите иерархию процессов
11. Назовите главные элементы «Модели состояний и 5-ти элементов»
12. Опишите процесс работы модели состояний


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Учебно-методические материалы для студентов кафедры АСОИУ
  Источник: http://www.4stud.info/rtos/lecture2.html
- Иерархии процессов
  Источник: http://os-pc.ru/proc/83-ierarkhii-protsessov.html
- Планирование и диспетчеризация процессов и потоков. Вытесняющие и невытесняющие алгоритмы планирования
  Источник: http://5fan.ru/wievjob.php?id=24512


## ЛЕКЦИЯ №7

**ТЕМА: Алгоритмы планирования. Классы планирования, уровни и их метрики.**

**ЦЕЛЬ: изучить основные задачи планирования, понять когда нужно применять определенный класс планирования. Знать метрики планирования.**

## ПЛАНИРОВАНИЕ ПРОЦЕССА

### 5.1 Основные понятия планирования процессов

**Планирование** - обеспечение поочередного доступа процессов к одному процессору.

**Планировщик** - отвечающая за это часть операционной системы.

**Алгоритм планирования** - используемый алгоритм для планирования какой либо задачи либо процесса.

Ситуации, когда необходимо планирование:

1. Когда создается процесс
2. Когда процесс завершает работу
3. Когда процесс блокируется на операции ввода/вывода, семафоре, и т.д.
4. При прерывании ввода/вывода.

**Алгоритм планирования без переключений** (неприоритетный) - не требует прерывание по аппаратному таймеру, процесс останавливается только когда блокируется или завершает работу.

**Алгоритм планирования с переключениями** (приоритетный) - требует прерывание по аппаратному таймеру, процесс работает только отведенный период времени, после этого он приостанавливается по таймеру, чтобы передать управление планировщику.

Необходимость алгоритма планирования зависит от задач, для которых будет использоваться операционная система.

**Основной задачей планирования процессов** в ОС является обеспечение высокой производительности ОС.

Существуют разные метрики, которыми оценивается эта производительность. Эти метрики зачастую противоречивы.

**Что концептуально требуется при проектировании планировщика ОС(какие цели должны быть достигнуты):**

1. **Максимизировать использование ЦП** (чтобы он максимально работал на задачах)
2. **Максимизировать пропускную способность**(число выполненных запросов в единицу времени)
3. **Минимизировать среднее время отклика** (среднее время от подачи запроса до завершения обработки ответа)

4. **Минимизировать среднее время ожидания** (среднее время от подачи запроса до начала его выполнения)
5. **Минимизировать энергии** (джоулей на инструкцию)

**КЛАССЫ ПЛАНИРОВАНИЯ**

1. **Пакетная система обработки**– ориентирован на длительные задачи, которые требуют больших вычислительных ресурсов, где не требуется частое прерывание.

Т.е. подразумевают обработку больших задач большими пакетами, нет ограничения на время выполнения.

Могут использовать *неприоритетный и приоритетный алгоритм*

например: для расчетных программ

**Его ЗАДАЧИ:**

1. *Пропускная способность* - количество задач в час увеличить
2. *Оборотное время* - минимизация времени на ожидание обслуживания и обработку задач.
3. *Использование процесса* - чтобы процессор всегда был занят.

2. **Интерактивные системы** – ориентирован на снижение времени отклика,

т.е. чтобы система казалась”отзывчивой”. (например перемещение мыши) ОС что-то делает. И всегда пользователю хочется, чтобы этот ответ происходил как можно быстрее.

Главное чтобы на поступающий в систему запрос был получен максимально быстро ответ. Запрос – это любое взаимодействие с компьютером.

Могут использовать только *приоритетный алгоритм*, нельзя допустить чтобы один процесс занял надолго процессор.

например: сервер общего доступа или персональный компьютер

**Его ЗАДАЧИ:**

1. Время отклика - быстрая реакция на запросы
2. Соразмерность - выполнение ожиданий пользователя (например: пользователь не готов к долгой загрузке системы)

3. **Реального времени** – специализированные класс, ориентированный на дедлайн – предельный срок завершения какой-либо работы. Главное, чтобы определенное действие завершалось к определенному сроку.

Классический пример СРВ – управление ядерным реактором, в котором превышение времени отклика приведет к аварийной ситуации.

Могут использовать неприоритетный и приоритетный алгоритм .

например: система управления автомобилем

**Его ЗАДАЧИ:**

1. Окончание работы к сроку - предотвращение потери данных
2. Предсказуемость - предотвращение деградации качества в мультимедийных системах (например: потерь качества звука должно быть меньше чем видео)

**УРОВНИ ПЛАНИРОВАНИЯ:**

Для каждого *уровня планирования процессов* можно предложить много различных алгоритмов. Выбор конкретного алгоритма определяется классом задач, решаемых вычислительной системой, и целями, которых мы хотим достичь, используя *планирование*.

1. **Долговременное(догосрочное)** – решает какие новые задачи будут добавлены (концептуальные вопросы).
2. **Среднесрочное** – решает нужно ли временно выгружать программу во вторичную память (какую и вообще нужно ли это).
3. **Краткосрочный** – решает, какому потоку дать следующий квант процессорного времени и какой длины. Координирует выполняющиеся потоки на разных ЦП.

**Процессы бывают:**

- ***Невытесняющие*** *(поп-preemptive)* алгоритмы основаны на том, что активному потоку позволяется выполняться, пока он сам, по собственной инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению поток

- ***Вытесняющие****(preemptive)* алгоритмы - это такие способы планирования потоков, в которых решение о переключении процессора с выполнения одного потока на выполнение другого потока принимается операционной системой, а не активной задачей.

Планирование процессов в ОС это процесс выбора – какой из процессов будет выполнятся следующим и как долго он будет выполнятся.

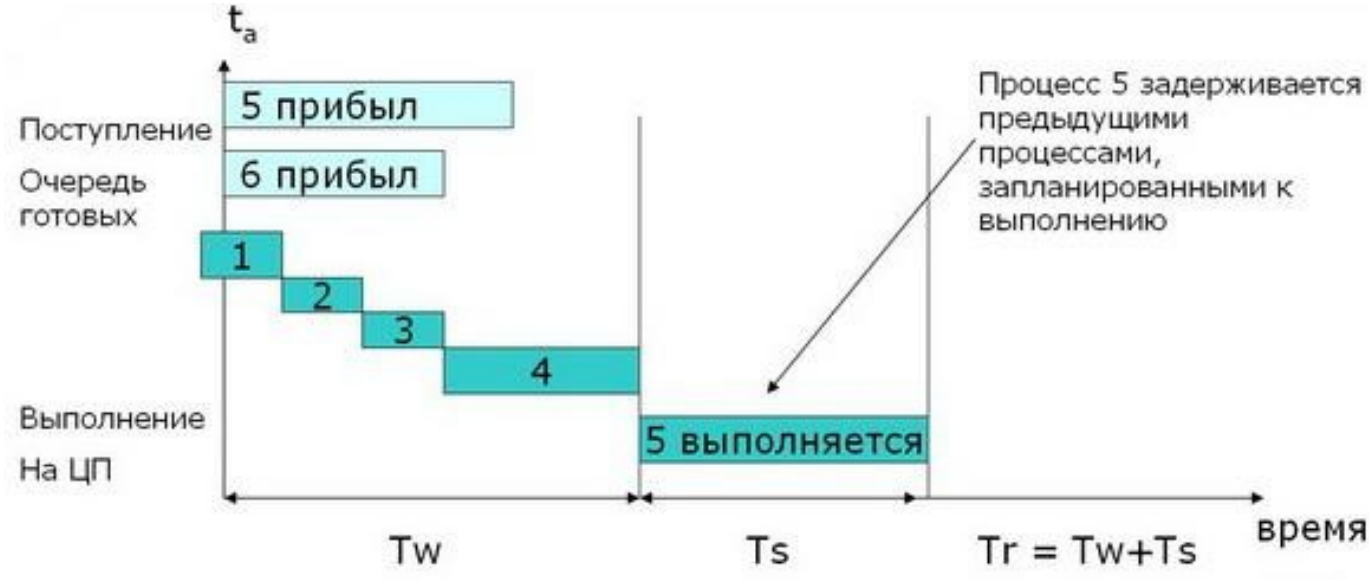
Не путать с переключением контекста, что является просто механизмом передачи управления. Переключения контекста это не есть операция планирования, это техническая операция.

1. Происходит прерывание;
2. Поток вызывает исключение или ловушку (trap);
3. После этого выбирается другой поток.

Т.е. в процессе переключения контекста нужно четко выбрать, кому передать управление.

**МЕТРИКИ ПЛАНИРОВАНИЯ**

На следующем рисунке рассмотрим пример метрики планирования.




Метрики планирования

**ta**— время поступления процесса (когда процесс становится готовым к выполнению);

**Tw** – время ожидания (которое тратит процесс в очереди на выполнение);

**Ts** – время выполнения ЦП;

**Tr** – время оборота (общее время на ожидание и выполнение).

На схеме **5** и **6** процессы поступили в очередь готовых процессов.

**5** задержался из-за 1-4 процесов. Для пятого процесса Tw – время ожидания Ts – время выполнения ЦП.

Время оборота это время от момента его поступления до момента, когда завершилось его выполнение Tr=Tw+Ts.

При планировании процессов главным остается вопрос: Как выбрать какой процесс будет работать дольше и дальше для этого мы рассмотри на следующей паре различные виды алгоритмов планирования?

**Контрольные вопросы:**

**1)** Какие классы планирования существуют?

**2)** Какие уровни планирования существуют?

**3)** Что требуется при проектировании планировщика ОС?

**4)** Какие метрики планирования вы знаете?

**5)** Перечислите алгоритмы планирования


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Алгоритмы планирования
  Источник: http://life-prog.ru/1_1057_algoritmi-planirovaniya.html
- Алгоритмы планирования
  Источник: http://baumanki.net/lectures/10-informatika-i-programmirovanie/353-operacionnye-sistemy/4804-algoritmy-planirovaniya.html


**ЛЕКЦИЯ №8**

**ТЕМА: Алгоритмы планирования в системах пакетной обработки. Планирование в интерактивных системах. Общее планирование реального времени.**

**ЦЕЛЬ: изучить основные виды планирования, понять принцип их работы в реальных проектах.**

**Существуют следующие алгоритмы планирования процессов:**

1. **FIFO**— классика – первым пришел, первым ушел
2. **Кратчайшая работа первой (SJF)**, т.е. следующей выбирается работа, которая требует наименьшего времени завершения
3. **Циклическое планирование**
4. **Планирование с приоритетом**
5. **Round-robin**
6. **Многоуровневая очередь**
7. **Многоуровневая очередь с обратной связью**

Рассмотрим названные алгоритмы планирования процессов.

***Планирование в системах пакетной обработки***

## 1) FIFO (First In, First Out (первым вошел, первым вышел)

Процессы ставятся в очередь по мере поступления.

***Преимущества:***

- Простота
- Справедливость (как в очереди покупателей, кто последний пришел, тот оказался в конце очереди)

***Недостатки:***

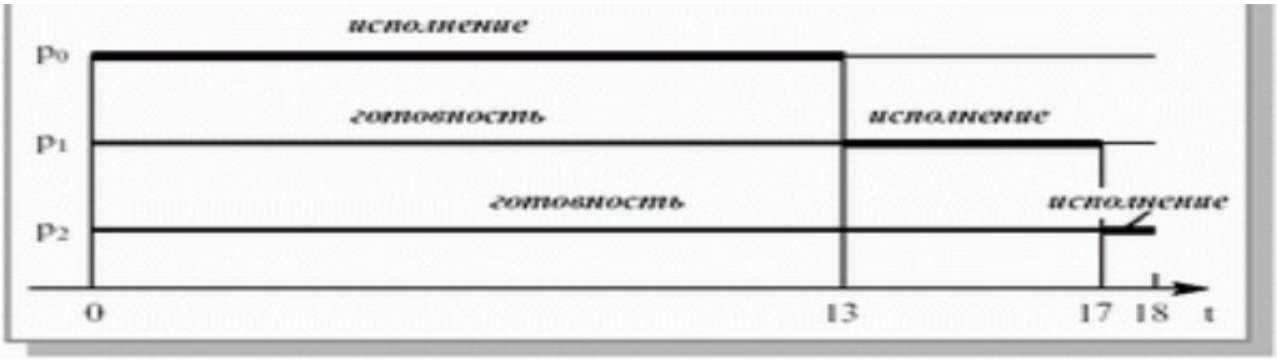
- Процесс, ограниченный возможностями процессора может затормозить более быстрые процессы, ограниченные устройствами ввода/вывода.

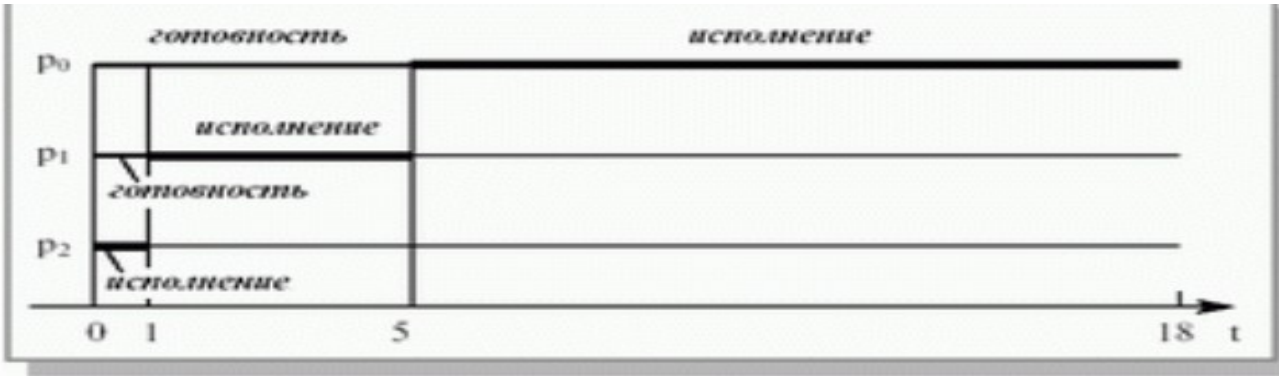
*В данном случае мы будем его понимать как невытесняющую многозадачность:*

1. Процессы планируются по мере их поступления;
2. Время выполнения не учитывается (никак совсем);
3. Другие процесс с меньшим временем Tr(временем обработки) вынуждены ждать (снижается отзывчивость системы);

4. Когда процесс переходит в состояние готовности он перемещается в конец очереди.

**Пример FIFO**




**Обобщения по FIFO**:

- Он больше других подходит для длительных, требовательных к времени ЦП процессов;
- Плохое использование ЦП и устройств ввода/вывода;
- Среднее время ожидание сильно варьируется.

Среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно зависят от порядка расположения процессов в очереди. Если у нас есть процесс с длительным временем , то короткие процессы, перешедшие в состояние готовность после длительного процесса, будут очень долго ждать начала выполнения.

Поэтому алгоритм FCFS практически неприменим для систем разделения времени – слишком большим получается среднее время отклика в интерактивных процессах.

## 2) Shortest-Job-First Кратчайшая работа первой (SJF)

**Суть** процесса — первым запланируется тот процесс, который требует **наименьшего времени для** своего **выполнения,** т.е. процесс имеющий **самое короткое время обработки.**

**Сложности:**

Нужно оценивать требуемое время обработки для каждого процесса.

**1.** Для пакетных заданий на основе предыдущего опыта или вручную (нет гарантии, что повторится)

**2.** Для интерактивных заданий на основе затраченного времени

Как только мы получаем метрику процессов, то короткий процесс помещается в начало очереди.

| | | | | | |
|---|---|---|---|---|---|
| 4 мин. | 6 мин. | 2 | 4 мин. | 2 | 2 |

| | | | | | |
|---|---|---|---|---|---|
| 2 | 2 | 2 | 4 мин. | 4 мин. | 6 мин. |

Нижняя очередь выстроена с учетом этого алгоритма

***Преимущества*:**

- Уменьшение оборотного времени
- Справедливость (как в очереди покупателей, кто без сдачи проходит в перед)
- Квантование времени не применяется.

Нижняя очередь выстроена с учетом этого алгоритма

**Недостатки:**

- Длинный процесс занявший процессор, не пустит более новые краткие процессы, которые пришли позже.

SJF алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим.

*При невытесняющем* SJF планировании процессор предоставляется избранному процессу на все требующееся ему время, независимо от событий происходящих в вычислительной системе.

*При вытесняющем SJF* планировании учитывается появление новых процессов в очереди готовых к исполнению (из числа вновь родившихся или разблокированных) во время работы выбранного процесса. Если CPU burst нового процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

Условно, имеется в виду невытесняющая политика планирования – сколько квант времени запрашивает процесс, столько ему и выделяется.

**Обобщение по «Кратчайшая работа первой»:**

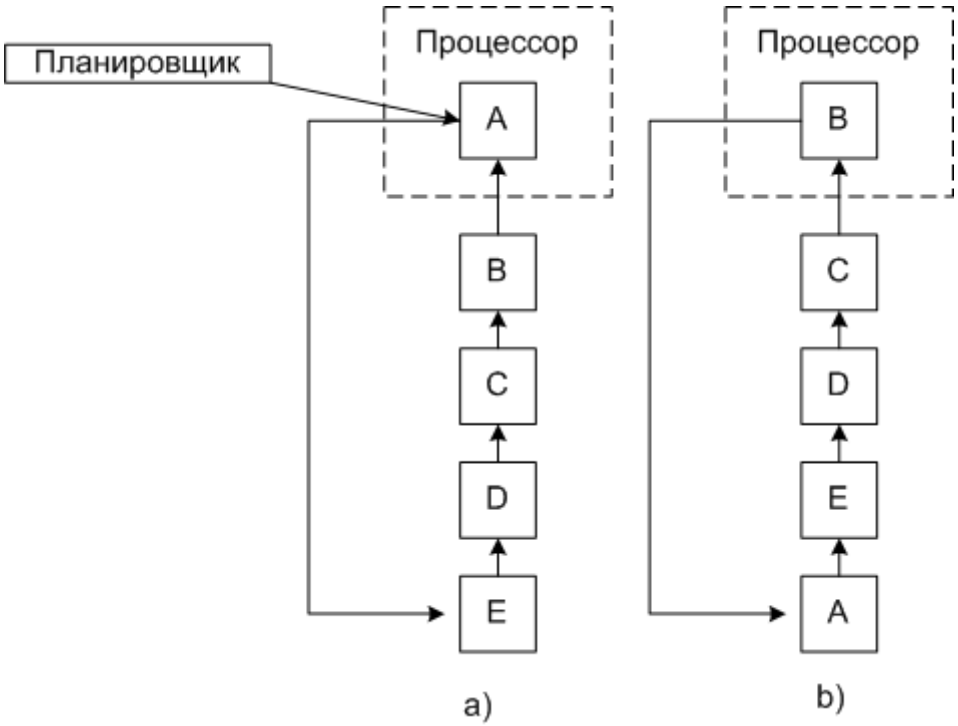
1. Процессы, уже выполняющиеся на ЦП вытесняются самым близким к завершению заданием;
2. Меньше общее время оборота процесса;
3. Меньше время ожидания ЦП.

### *5.3 Планирование в интерактивных системах*

## 3) Циклическое планирование

Самый простой алгоритм планирования и часто используемый.

Каждому процессу предоставляется квант времени процессора. Когда квант заканчивается процесс переводится планировщиком в конец очереди. При блокировке процессор выпадает из очереди.




Пример циклического планирования

***Преимущества:***

1) Простота
2) Справедливость (как в очереди покупателей, каждому только по килограмму)

***Недостатки:***

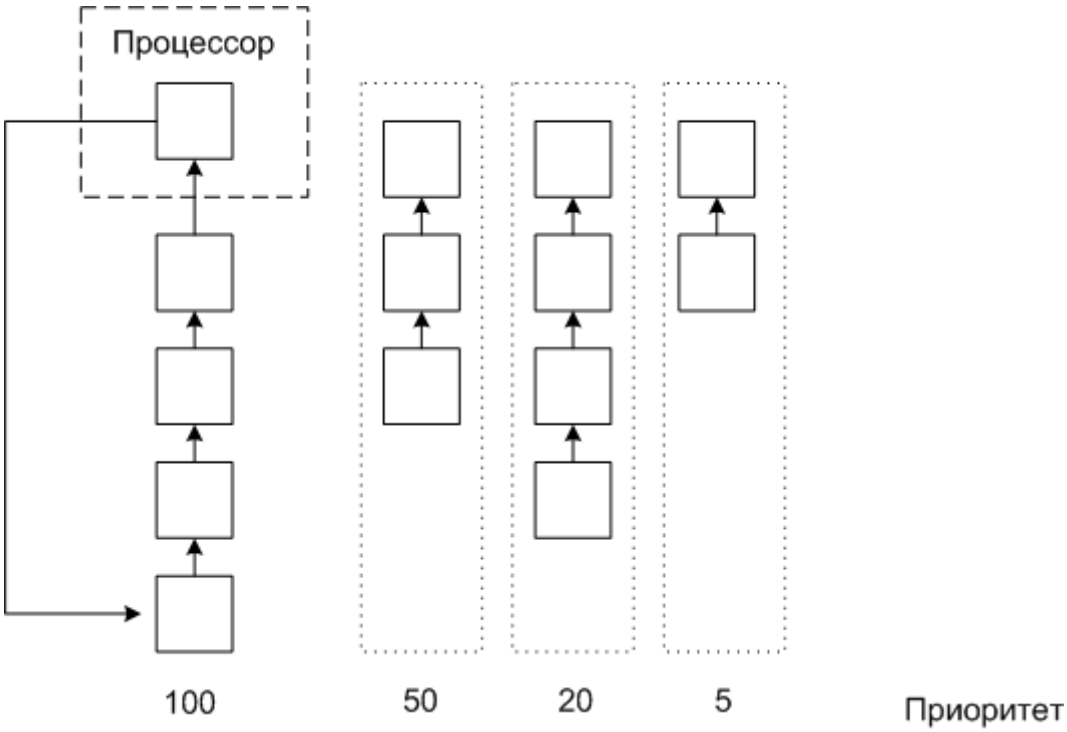
1) Если частые переключения (квант - 4мс, а время переключения равно 1мс), то происходит уменьшение производительности.
2) Если редкие переключения (квант - 100мс, а время переключения равно 1мс), то происходит увеличение времени ответа на запрос.

## 4) Планирование с приоритетами

Каждому процессу присваивается **приоритет(числовое значение)**, и управление передается процессу с самым высоким приоритетом.

Приоритет может быть динамический и статический.

Часто процессы объединяют по приоритетам в группы, и используют приоритетное планирование среди групп, но внутри группы используют циклическое планирование.




Приоритетное планирование 4-х групп

Тот же алгоритм кратчайшей работы следующей можно представить, как планирование с приоритетом, где приоритет – наименьшее время работы. Чем меньше до конца обработки процесса, тем выше приоритет.

**Суть:** каждому процессу сопоставляется некоторое число, которое характеризует, определяет **приоритет** этого процесса. Чем меньше это число, тем выше приоритет.

**Проблема старвации** – это проблема “зависания”,“голодания” – если процессу с высоким приоритетом приспичит выполнить очень длительную работу, то все остальные процессы будут “висеть” и ждать.

Время ЦП выделяется процессу с наивысшим приоритетом (вытесняющим или невытесняющим). Процесс с низким приоритетом может вообще никогда не выполниться, до него не дойдет очередь.

**Старвацию** ее можно представить в виде очереди и кто привилегированный лезет вне очереди.

Проявляется в алгоритме KPC(кратчайшая работа следующей). Низкоприоритетные запросы могут никогда не выполниться.

**Решение проблемы:**

Ввести понятие «**старения**»: по мере течения времени увеличивать приоритет процесса.

## 5)Round-robin

**суть метода.** Пусть имеется $N$ объектов, способных выполнить заданное действие, и $M$ задач, которые должны быть выполнены этими объектами. Подразумевается, что объекты $n$ равны по своим свойствам между собой, задачи $m$ имеют равный приоритет. Тогда первая задача ($m = 1$) назначается для выполнения первому объекту ($n = 1$), вторая — второму и т. д., до достижения последнего объекта ($m = N$). Тогда следующая задача ($m = N+1$) будет назначена снова первому объекту и т. п.

Проще говоря, происходит перебор выполняющих задания объектов по циклу, или по кругу (round), и по достижении последнего объекта следующая задача будет также назначена первому объекту.

Решение задач может быть дополнительно разбито на кванты времени, причем для продолжения решения во времени нумерация объектов (и, соответственно, назначенные задачи) сдвигается по кругу на 1, то есть задача первого объекта отдается второму, второго — третьему, и т. д., а первый объект получает задачу последнего, либо освобождается для приёма новой задачи. Таким образом, алгоритм Round-robin становится алгоритмом распределения времени или балансировки нагрузки.

Данный алгоритм планирования обозначает циклический алгоритм с вытесняющим планированием.

**1.** Каждый процесс получает фиксированный квант процессорного времени (фиксированную единицу процессорного времени).

**2.** После истечения кванта времени процесс принудительно вытесняется и помещается в очередь готовых к выполнению.

**3.** Процесс всегда планируются в том же порядке и каждый процесс получает одинаковый квант времени

**4.** Не сложно подсчитать: если квант времени равен **q** и **n**-процессов в очереди, то каждый получит **1/n** времени ЦП, кусками максимум по **q**

**5.** Никакой из процессов не ожидает больше, чем (n-1)*q единиц времени

**1.** Если q меньше, чем время, затрачиваемое на переключение контекста, тогда диспетчер будет неэффективным.

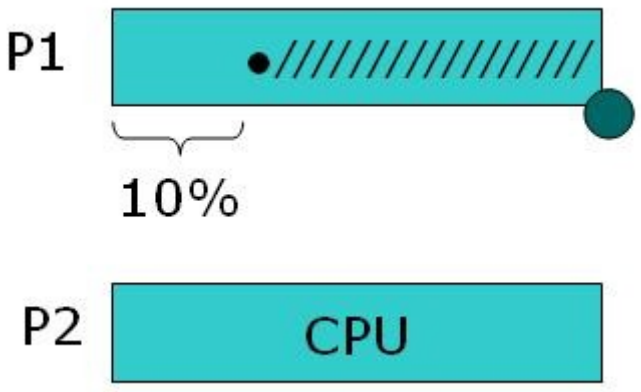
**При выполнении процесса возможны два варианта:**

**1) Время непрерывного использования процессора, требующееся процессу, (остаток текущего CPU burst) меньше или равно продолжительности кванта времени.** Тогда процесс по своей воле освобождает процессор до истечения кванта времени, на исполнение выбирается новый процесс из начала очереди и таймер начинает отсчет кванта заново.

**2) Продолжительность остатка текущего CPU burst процесса больше, чем квант времени**. Тогда по истечении этого кванта процесс прерывается таймером и помещается в конец очереди процессов готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале

**Недостаток Round-robin**

Процессы заблокированные в ожидании вв/выв, полностью не используют свой квант времени, поэтому процессы с интенсивным использованием ЦП получают преимущество.




**Недостаток Round-robin**

Проблема RR в том, что **не учитываются задержки**, и полезное время работы P1 составляет только 10%.

Есть 2 процесса P1 и P2

Процесс P1 ожидает ввод/вывод в точке (●), пока этот вв/выв не завершится часть отмеченного штриховкой времени процесс P1 потратит «впустую», он вытиснится только в зеленой точке по истечении кванта времени.

P2 в это время активно использует ЦП, например, считает.

**ПРИМЕР:** АЛГРИМ СЧИТАЕТ КОРЕНЬ, В ТО ВРЕМЯ ПОКА ПРОЦЕСС ВЫВОДА ОТВЕТА НА ЭКРАН,ПРОСТАЕВАЕТ.

## 6)Многоуровневые очереди

Для систем, в которых процессы могут быть легко рассортированы на разные группы, был разработан другой класс алгоритмов планирования. Для каждой группы процессов создается своя очередь процессов, находящихся в состоянии готовность (см. рисунок).

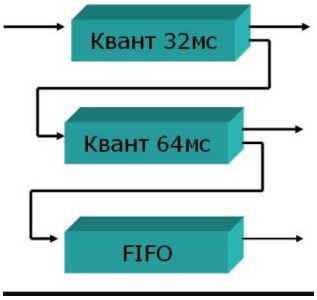
Этим очередям приписываются фиксированные приоритеты. Приоритет очереди процессов, запущенных студентами, — ниже, чем для очереди процессов, запущенных преподавателями. Это значит, что ни один пользовательский процесс не будет выбран для исполнения, пока есть хоть один готовый системный процесс, и ни один студенческий процесс не получит в свое распоряжение процессор, если есть процессы преподавателей, готовые к исполнению. Внутри этих очередей для планирования могут применяться самые разные алгоритмы. Так, например, для больших счетных процессов может использоваться алгоритм FCFS, а для интерактивных процессов – алгоритм RR. Подобный подход, получивший название многоуровневых очередей, повышает гибкость планирования: для процессов с различными характеристиками применяется наиболее подходящий им алгоритм.

Но в случае многоуровневых очередей нужно планирование не просто внутри каждой очереди, но и **планирование между очередями**. Получается «накрученный» планировщик, можно предложить много вариантов:

**1) Планирование с фиксированным приоритетом**

- Вначале обслужить все интерактивные процессы, потом все фоновые
- Возможна старвация

**2) Разделение времени**




Каждой очереди выделяется часть времени ЦП, которую она может распланировать между своими процессами. Например 80% времени ЦП на интерактивные процессы через RR, 20% на фоновые через FIFO.

**3) Многоуровневая очередь с обратной связью**

## Многоуровневая очередь с обратной связью

Планирование на основе затраченного времени, если процесс затратил определенный квант времени, то он помещается в определенную очередь – **динамически перепланируются очереди**.

**Многоуровневая очередь с обратной связью:**

-Если он выполнился достаточно быстро, то он попадает в первую очередь «быстрых» процессов.

-Если он средний по времени выполнения, то в среднюю.

-Если он требует много времени вычислительных ресурсов, то он помещается в последнюю очередь FIFO.

За счет этого процессы постоянно перемещаются между очередями, таким образом заранее не нужно смотреть, куда помещать процесс и сопоставлять ему какое-то свойство.

**Пример многоуровневой очереди с обратной связью**

Есть три очереди:

- Q0 –RR с квантом времени t=16мс
- Q1 –RR с квантом времени t=32мс

- Q2 –FIFO

## ***Общее планирование реального времени***

Используется модель, когда каждый процесс борется за процессор со своим заданием и графиком его выполнения.

Планировщик должен знать:

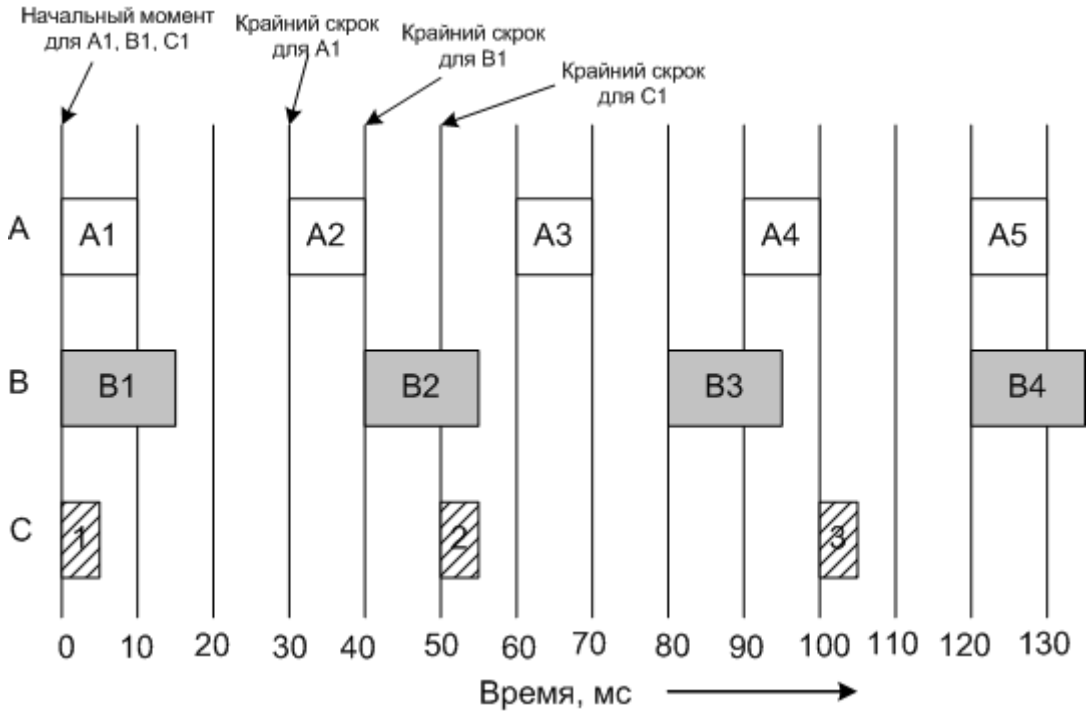
- частоту, с которой должен работать каждый процесс
- объем работ, который ему предстоит выполнить
- ближайший срок выполнения очередной порции задания

Рассмотрим пример из трех процессов.

Процесс **A** запускается каждые 30мс, обработка кадра 10мс

Процесс **B** частота 25 кадров, т.е. каждые 40мс, обработка кадра 15мс

Процесс **C** частота 20 кадров, т.е. каждые 50мс, обработка кадра 5мс




Три периодических процесса

Проверяем, можно ли планировать эти процессы.

10/30+15/40+5/50=0.808<1

Условие выполняется, планировать можно.

Можно планировать эти процессы **статическим** (приоритет заранее назначается каждому процессу) и **динамическим** методами.

**Контрольные вопросы:**

6) Расскажите суть алгоритм планирования FIFO
7) Расскажите суть алгоритма кратчайшей работы первой
8) Планирование с приоритетом.
9) В чем суть метода Round-robin? Недостатки метода
10) Расскажите про многоуровневую очередь с обратной связью


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Алгоритмы планирования
  Источник: http://life-prog.ru/1_1057_algoritmi-planirovaniya.html
- Алгоритмы планирования
  Источник: http://baumanki.net/lectures/10-informatika-i-programmirovanie/353-operacionnye-sistemy/4804-algoritmy-planirovaniya.html


## ЛЕКЦИЯ №9

**ТЕМА: управление памятью. Функции ОС по управлению памятью. Типы адресов.**

**ЦЕЛЬ: изучить методы распределения памяти и типы адресов.**

### *1 ОСНОВНЫЕ ПОНЯТИЯ*

***Основная память*** - это устройство для хранения информации. Она состоит из оперативного запоминающего устройства (ОЗУ) и постоянного запоминающего устройства (ПЗУ).

***1.Оперативное запоминающее устройство (ОЗУ)***

**ОЗУ**-быстрая, полупроводниковая, энергозависимая память. В ОЗУ хранятся исполняемые в данный момент программы и данные, с которыми она непосредственно работает. Это значит, что когда вы запускаете какую-либо компьютерную программу, находящуюся на диске, она копируется в оперативную память, после чего процессор начинает выполнять команды, изложенные в этой программе.

ОЗУ - это память, используемая как для чтения, так и для записи информации. При отключении электропитания информация в ОЗУ исчезает (энергозависимость).

***2. Постоянное запоминающее устройство (ПЗУ)***

**ПЗУ** – это тоже быстрая память, предназначенная только для чтения. Информация заносится в нее один раз (обычно в заводских условиях) и сохраняется постоянно (при включенном и выключенном компьютере). В ПЗУ хранится информация, присутствие которой постоянно необходимо в компьютере.

В ПЗУ находятся:

- тестовые программы, проверяющие при каждом включении компьютера правильность работы его блоков;
- программы для управления основными периферийными устройствами -дисководом, монитором, клавиатурой;
- информация о том, где на диске расположена операционная система.

Основная память состоит из регистров. Регистр - это устройство для временного запоминания информации в оцифрованной (двоичной) форме.

Так вот за всю основную память отвечает МЕНЕДЖЕР ПАМЯТИ.

**Менеджер памяти** - часть операционной системы, отвечающая за управление памятью.

**Основные методы распределения памяти:**

1) **БЕЗ ИСПОЛЬЗОВАНИЯ ВНЕШНЕЙ ПАМЯТИ** (например: HDD-жесткий диск)

***Жёсткий диск*** – Это наше постоянное запоминающее устройство компьютера, то есть, его основная функция - долговременное хранение данных.

Получается, что жёсткий диск служит лучшим местом на компьютере для хранения личной информации: файлы, фотографии, документы и видеозаписи, явно будут долго храниться именно на нём, а сохранённую информацию можно будет использовать и в дальнейшем в своих нуждах.

**2) С ИСПОЛЬЗОВАНИЕМ ВНЕШНЕЙ ПАМЯТИ (**виртуальная память и свопинг**) cpc**

Так как памяти, как правило, не хватает. Для выполнения процессов часто приходится использовать диск.

Основные способы использования диска:

- **Свопинг** (подкачка) - процесс целиком загружается в память для работы
- **Виртуальная память** - процесс может быть частично загружен в память для работы

## 1. МЕТОДЫ БЕЗ ИСПОЛЬЗОВАНИЯ ВНЕШНЕЙ ПАМЯТИ

1. Однозадачная система без подкачки на диск
2. Распределение памяти с фиксированными разделами
3. Распределение памяти динамическими разделами

### 1.1 Однозадачная система без подкачки на диск

*Память разделяется только между программой и операционной системой.*

Схемы разделения памяти:


Схемы разделения памяти

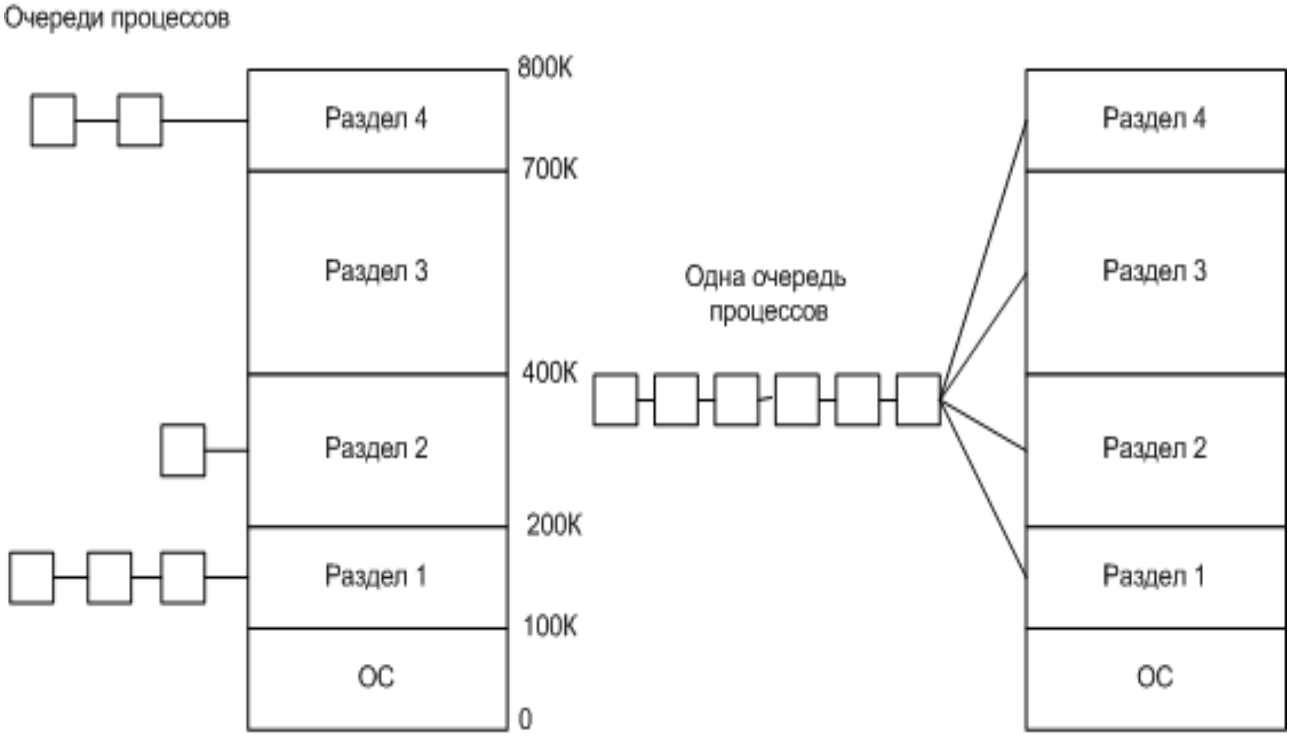
Третий вариант используется в MS-DOS. Та часть, которая находится в ПЗУ, часто называется BIOS.

### 1.2 Распределение памяти с фиксированными разделами.

Память просто разделяется на несколько разделов (возможно, не равных). Процессы могут быть разными, поэтому каждому разделу необходим разный размер памяти.

Системы могут иметь:

1. общую очередь ко всем разделам
2. к каждому разделу отдельную очередь




Распределение памяти с фиксированными разделами

**Недостаток** системы многих очередей очевиден, когда большой раздел может быть свободным, а к маленькому выстроилась очередь.

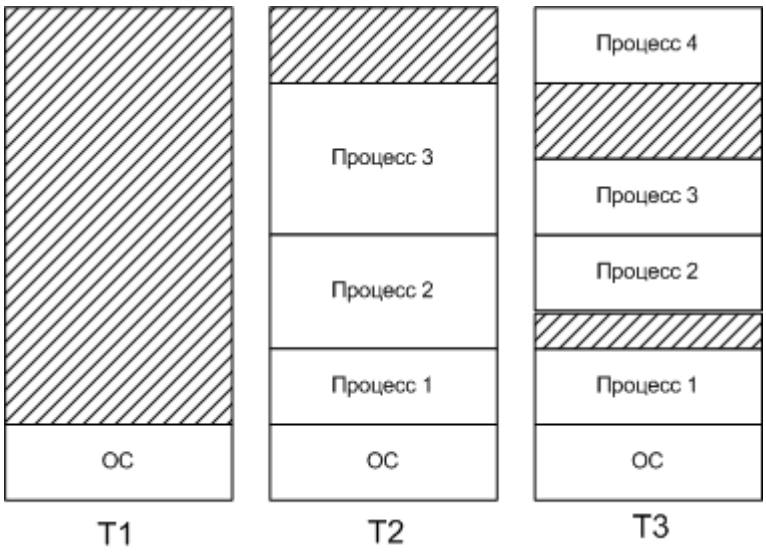
*Алгоритмы планирования в случае одной очереди:*

1. поочередный
2. выбирается задача, которая максимально займет раздел

Также может быть смешанная система.

### 1.3 Распределение памяти динамическими разделами

В такой системе сначала память свободна, потом идет динамическое распределение памяти.



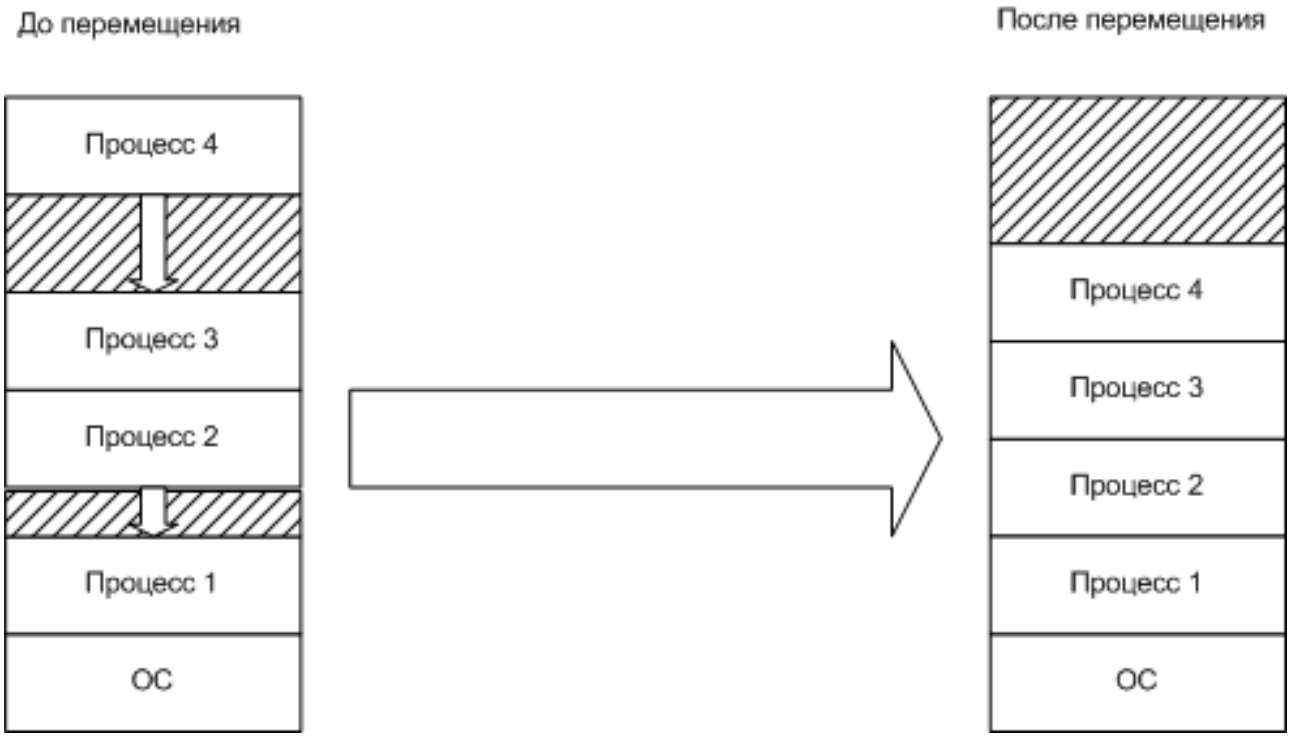

Распределение памяти динамическими разделами.

**Недостатки:**

1. Сложность
2. Память фрагментируется

**Перемещаемые разделы**

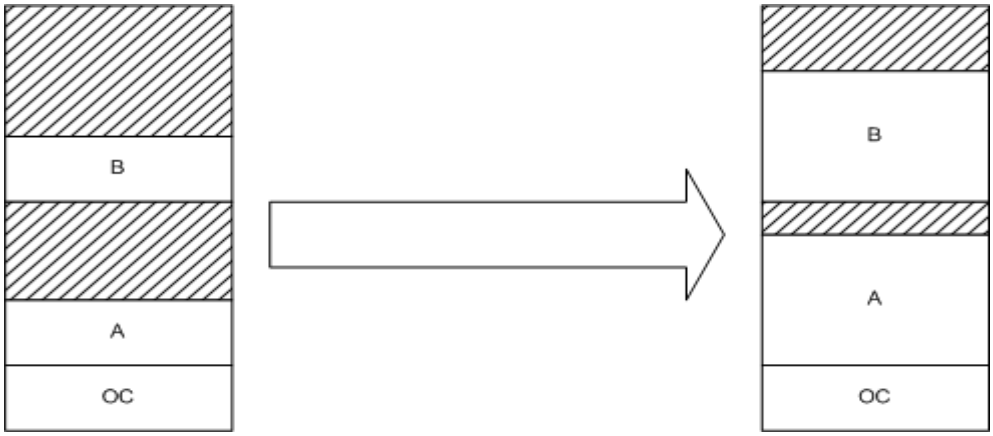
Это один из методов борьбы с фрагментацией. Но на него уходит много времени.




Перемещаемые разделы

**Рост разделов**

Иногда процессу может понадобиться больше памяти, чем предполагалось изначально, УВЕЛИЧИВАЕТСЯ РОСТ РАЗДЕЛОВ.




**Настройка адресов и защита памяти**

В предыдущих примерах мы можем увидеть две основные проблемы.

- Настройка адресов или перемещение программ в памяти
- Защита адресного пространства каждой программы

Решение обоих проблем заключается в оснащении машины специальными аппаратными регистрами.

1. Базовый (указывает начало адресного пространства программы)
2. Предельный (указывает конец адресного пространства программы)

   Благодаря этому мы будем знать где начинается и где заканчивается используемое пространство.

**2. МЕТОДЫ С ИСПОЛЬЗОВАНИЕМ ВНЕШНЕЙ ПАМЯТИ (СВОПИНГ И ВИРТУАЛЬНАЯ ПАМЯТЬ) СРС.**

## *2 ФУНКЦИИ ОС ПО УПРАВЛЕНИЮ ПАМЯТЬЮ*

Под памятью (memory) здесь подразумевается оперативная память компьютера. В отличие от памяти жесткого диска, которую называют внешней памятью (storage), оперативной памяти для сохранения информации требуется постоянное электропитание.

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы.

**Процессор может выполнять только инструкции, находящиеся в оперативной памяти. Память распределяется как между модулями прикладных программ, так и между модулями самой операционной системы.**

**Функции ОС по управлению памятью в мультипрограммной**

**системе:**

1. отслеживание свободной и занятой памяти;
2. выделение памяти процессам и ее освобождение при завершении процесса;
3. вытеснение процессов из оперативной памяти на диск при нехватке оперативной памяти и возвращение в оперативную память при освобождении места в ней (механизм ***виртуальной памяти***);
4. настройка адресов программы на конкретную область физической памяти;
5. динамическое выделение памяти процессам (выделение памяти по запросу приложения во время его выполнения); выделяются свободные участки, расположенные произвольным образом, что приводит к фрагментации памяти;
6. дефрагментация освобожденной динамической памяти;
7. выделение памяти для создания служебных структур ОС (дескрипторы процессов и потоков, таблицы распределения ресурсов, буферы, синхронизирующие объекты и т.д.;
8. защита памяти – выполняемый процесс не должен записывать или читать данные из памяти, назначенной другому процессу.

## 3 ТИПЫ АДРЕСОВ

Для идентификации переменных и команд на разных этапах жизненного *цикла* программы используются символьные имена, преобразуемые в виртуальные адреса(математические, условные, логические – все это синонимы) и в итоге – в физические адреса.

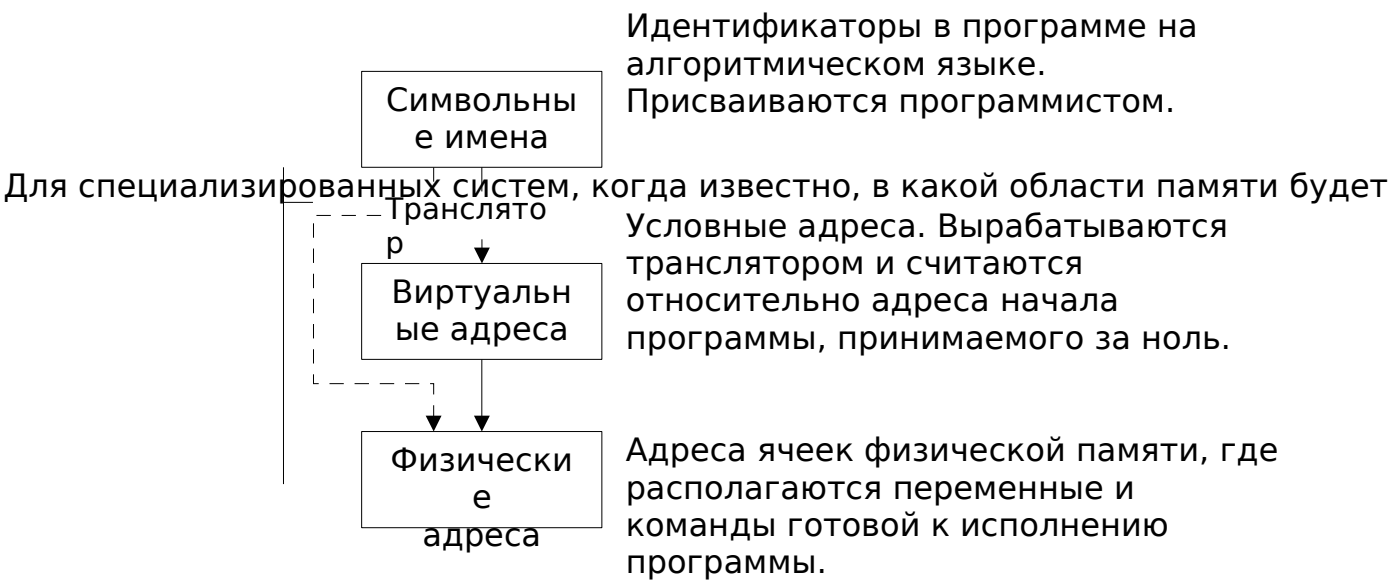



**Рис. 1** Типы адресов

**Символьные имена** присваивает *пользователь* при написании программ на алгоритмическом языке или ассемблере.

**Виртуальные** адреса вырабатывает *транслятор*, переводящий программу на *машинный язык*. Поскольку во время трансляции неизвестно , в какое *место* оперативной памяти будет загружена *программа*, *транслятор* присваивает переменным и командам виртуальные (условные) адреса, считая *по* умолчанию, что начальным адресом программы будет нулевой *адрес*.

**Физические адреса** соответствуют номерам ячеек оперативной памяти, где в действительности будут расположены переменные и команды.

Совокупность всех виртуальных адресов процесса называется **виртуальным адресным пространством.** *Диапазон* адресов виртуального пространства у всех процессов один и тот же и определяется разрядностью адреса процессора (для Pentium *адресное пространство* составляет объем, равный $2^{32}$ *байт*, с диапазоном адресов от $0000.0000_{16}$ до $FFFF.FFFF_{16}$).

***Существует два принципиально отличающихся подхода к преобразованию виртуальных адресов в физические.***

**В первом случае** *замена виртуальных адресов на физические* выполняется один раз для каждого процесса во время начальной загрузки программы в память.

Специальная системная программа **— перемещающий загрузчик** — на основании имеющихся у нее исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, а также информации, предоставленной транслятором об адресно-зависимых элементах программы, выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическим

**Второй способ** заключается в том, что *программа* загружается в *память* в виртуальных адресах. Во *время выполнения* программы при каждом обращении к памяти *операционная система* преобразует виртуальные адреса в физические.

## Контрольные вопросы:

1. Что такое «виртуальный адрес», «виртуальное адресное пространство»?
2. Функции ОС по управлению памятью ?
3. Типы виртуальных адресов?
4. Какие методы распределения памяти вы знаете, опишите их?
5. ***Подходы к преобразованию виртуальных адресов в физические?***
6. Назначение перемещающего загрузчика


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер

- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Операционные системы: Учебник для вузов. 2-е изд. /А.В. Гордеев. – СПб.: Питер, 2006. - 416 с.: ил.

## ЛЕКЦИЯ №10

**ТЕМА: кэш-память. Принципы работы кэш-памяти.**

**ЦЕЛЬ: изучить принцип работы кэш-памяти, его структуру и взаимодействие модулей. Понять для чего используется кэш-память и кэширование данных.**

### КЭШ-ПАМЯТЬ

Проще всего ответить на вопрос, зачем нужна кэш память. Как известно, процессор работает с данными, хранящимися в оперативной памяти. Однако скорость работы оперативной памяти и процессора существенно различаются: *если бы процессор напрямую общался с оперативной памятью (читал или записывал данные), то большую часть времени попросту простаивал бы.* Именно для сокращения задержек доступа к оперативной памяти и применяется кэш-память, которая значительно более скоростная в сравнении с оперативной.

Фактически если оперативная память используется для того, чтобы сгладить задержки доступа к данным на накопителе (HDD-диске, SSD-накопителе или флэшпамяти), то *кэшпамять процессора применяется для нивелирования задержек доступа к самой оперативной памяти.* В этом смысле оперативную память можно рассматривать как кэш накопителя.

Однако между оперативной памятью и кэшем процессора есть одно очень серьезное различие: кэш процессора полностью виден для программиста, то есть нельзя адресовать программным образом находящиеся в нем данные.

Понятно, что для того, чтобы кэш процессора мог выполнять свою основную задачу, то есть сглаживать доступ к оперативной памяти, он должен работать гораздо быстрее, чем оперативная память. Так, если оперативная память представляет собой динамическую память с произвольным доступом (Dynamic Random Access Memory, DRAM), то кэш процессора выполняется на базе статической оперативной памяти (Static Random Access Memory, SRAM).

**Кэш** — промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше идёт быстрее, чем выборка исходных данных из оперативной (ОЗУ) и быстрее внешней (жёсткий диск или твердотельный накопитель) памяти, за счёт чего уменьшается среднее время доступа и увеличивается общая производительность компьютерной системы.

Кэш состоит из набора записей. Каждая запись ассоциирована с элементом данных или блоком данных (небольшой части данных), которая является копией элемента данных в основной памяти. Каждая запись имеет идентификатор, определяющий соответствие между элементами данных в кэше и их копиями в основной памяти. Это идея позволяет программистам проще работать с кэшем и обращаться к ней по идентификаторам, которые им присвоены.

## ПРИНЦИП РАБОТЫ КЭША ПРОЦЕССОРА

Итак, мы разобрались с назначением кэша процессора, а теперь рассмотрим базовые принципы работы кэша, которые позволяют ему решать свою основную задачу.

Кэш состоит из контроллера и собственно кэшпамяти**. Кэш-контроллер** управляет работой кэшпамяти, то есть загружает в нее нужные данные из оперативной памяти и возвращает, когда нужно, модифицированные процессором данные в оперативную память.

Архитектурно кэшконтроллер расположен между процессором и оперативной памятью (рис. 1).

Когда процессор послал запрос в ОП, контроллер Перехватывая запросы к оперативной памяти, **кэш-контроллер** определяет, имеется ли копия затребованных данных в кэше. Если такая копия там есть, то это называется **кэш-попаданием (cache hit**) , в таком случае данные очень быстро извлекаются из кэша (существенно быстрее, чем из оперативной памяти). Если же требуемых данных в кэше нет, то говорят о **кэш--промахе (cache miss**) — тогда запрос данных переадресуется к оперативной памяти.


*Рис. 1. Структура кэш-памяти процессора*

Для достижения наивысшей производительности кэшпромахи должны происходить как можно реже (в идеале — отсутствовать).

Учитывая, что по емкости кэшпамять намного меньше оперативной памяти, добиться этого не так-то просто. **А потому основная задача кэш-контроллера заключается в том**, чтобы загружать кэшпамять действительно нужными данными и своевременно удалять из нее данные, которые больше не понадобятся.

Важно понимать, что кэш всегда «полон», так как оставлять часть кэшпамяти пустой нерационально. Новые данные попадают в кэш только путем вытеснения (замещения) какихлибо старых данных.

Загрузка кэша данными реализуется на основе так называемой стратегии кэширования, а выгрузка данных — на основе политики замещения.

**Контрольные вопросы:**

1. Принцип работы кэш-памяти?
2. Какая структура кэш-памяти процессора?
3. Что такое кэш-память? Чем она отличается от оперативной памяти?
4. Для чего применяется кэш-память?


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Операционные системы: Учебник для вузов. 2-е изд. /А.В. Гордеев. – СПб.: Питер, 2006. - 416 с.: ил.


## ЛЕКЦИЯ №11

**ТЕМА: Файлова система. Логическая организация файловой системы. Диски, разделы, сектора, кластеры.**

**ЦЕЛЬ: изучить принцип работы логической файловой системы, начать разбираться как она должна быть связана с физической организацией ФС.**

Прежде чем перейти к логической и физической организации файловой системой, ознакомимся, что вообще такое файловая система и из чего она состоит.

Одной из основных задач файловой системы является предоставление удобств пользователю при работе с данными, хранящимися на дисках. Для этого ОС подменяет физическую структуру хранящихся данных некоторой удобной для пользователя логической моделью.

*В общем случае, данные, содержащиеся в файле, имеют некоторую логическую структуру. Эта структура является базой при разработке программы, предназначенной для обработки этих данных.*

**Например**, чтобы текст мог быть правильно выведен на экран, программа должна иметь возможность выделить отдельные слова, строки, абзацы.

Признаками, отделяющими один структурный элемент от другого (абзац от отдельного слова), могут служить определенные кодовые последовательности или просто известные программе значения смещений этих структурных элементов, относительно начала файла.

***Логическая модель файловой системы материализуется в виде дерева каталогов, в символьных составных именах файлов, в командах работы с файлами.*** Базовым элементом этой модели является файл, который так же, как и файловая система в целом, может характеризоваться как логической, так и физической структурой.

**Файловая система (ФС**) - это совокупность файлов, системных структур данных, отслеживающих размещение файлов на диске и свободное дисковое пространство, а также комплекс необходимых системных программных средств.

***Файловая система (ФС)*** **как часть операционной системы, включает в себя:**

1. совокупность всех файлов на диске
2. служебные структуры, включая каталоги
3. системные программные средства

*Файл* - это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные.

**Файловые системы поддерживают несколько функционально различных типов файлов:**

1. Обычные файлы (ОС не контролирует содержимое этих файла)
2. Каталоги (содержит системную информацию о наборе файлов)

3. Специальные файлы (фиктивные файлы, соответствующие устройствам ввода-вывода)

## ЛОГИЧЕСКАЯ ОРГАНИЗАЦИЯ

Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. **Логическая запись** - это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи.


Рис. Способы логической организации файла

1) **Фиксированной длины**. Размер записи фиксирован в пределах файла, а записи в различных файлах, принадлежащих одной и той же файловой системе, могут иметь различный размер.

В таком случае доступ к n-й записи файла осуществляется либо путем последовательного чтения (n-1) предшествующих записей, либо прямо по

адресу, вычисленному по ее порядковому номеру. Например, если L — длина записи, то начальный адрес n-й записи равен Lxn.

2) **Переменной длины.** Представление данных в виде последовательности записей, размер которых изменяется в пределах одного файла. *Для поиска нужной записи система должна последовательно считать все предшествующие записи*. Вычислить адрес нужной записи по ее номеру при такой логической организации файла невозможно, а следовательно, не может быть применен более эффективный метод прямого доступа.(недостаток, нужно все считывать последовательно)

3) **Индексированные файлы**, они допускают более быстрый прямой доступ к отдельной логической записи. Записи имеют одно или более ключевых (индексных) полей и могут адресоваться путем указания значений этих полей. Для быстрого поиска данных в индексированном файле предусматривается *специальная индексная таблица,* в которой значениям ключевых полей ставится в соответствие адрес внешней памяти. Т.е. по указанию индекса поля мы быстро обращаемся к нужной нам части памяти.

Этот адрес может указывать либо непосредственно на искомую запись, либо на некоторую область внешней памяти, занимаемую несколькими записями, в число которых входит искомая запись. Ведение индексных таблиц берет на себя файловая система. Понятно, что записи в индексированных файлах могут иметь произвольную длину.

## ДИСКИ, РАЗДЕЛЫ СЕКТОРА, КЛАСТЕРЫ

Основным типом устройства, которое используется в современных вычислительных системах для хранения файлов, являются дисковые накопители.

Жесткий диск (дисковые накопители) состоит из одной или нескольких стеклянных или металлических пластин, каждая из которых покрыта с одной или двух сторон магнитным материалом.

Пользователь может образно представить себе жесткий диск как блокнот в клеточку. Одна клеточка на странице – это один кластер. Файловая система – это содержание блокнота, а файл – слово.

Таким образом, **диск** в общем случае состоит из *пакета* пластин. На каждой стороне каждой пластины размечены тонкие концентрические кольца — ***дорожки*** *(traks),* на которых хранятся данные.

Нумерация дорожек начинается с 0 от внешнего края к центру диска. Когда диск вращается, элемент, называемый головкой, считывает двоичные данные с магнитной дорожки или записывает их на магнитную дорожку. Головки перемещаются над поверхностью диска дискретными шагами, каждый шаг соответствует сдвигу на одну дорожку. Совокупность дорожек одного радиуса на всех поверхностях всех пластин пакета называется ***цилиндром*** *(cylinder) разделом.*

Каждая дорожка разбивается на фрагменты, называемые ***секторами*** *(sectors),* ТАК ЧТО ВСЕ ДОРОЖКИ ИМЕЮТ РАВНОЕ ЧИСЛО СЕКТОРОВ, в которые можно максимально записать одно и то же число байт[1]. Сектор имеет фиксированный для конкретной системы размер, выражающийся степенью двойки.

Чаще всего размер сектора составляет 512 байт. Учитывая, что дорожки разного радиуса имеют одинаковое число секторов, плотность записи становится тем выше, чем ближе дорожка к центру.

**Сектор — наименьшая адресуемая единица обмена данными дискового устройства с оперативной памятью.** Для того чтобы контроллер мог найти на диске нужный сектор, ему необходимо задать все составляющие адреса сектора: номер цилиндра, номер поверхности и номер сектора.

Операционная система при работе с диском использует, как правило, собственную единицу дискового пространства, называемую ***кластером*** *(cluster)* . При создании файла место на диске ему выделяется кластерами. Например, если файл имеет размер 2560 байт, а размер кластера в файловой системе определен в 1024 байта, то

файлу будет выделено на диске 3 кластера. Величина кластера определяется емкостью данного диска: чем больше диск, тем больше значение кластера. В РЕАЛЬНОСТИ ИМЕННО КЛАСТЕР ЯВЛЯЕТСЯ ЕДИНИЦЕЙ АДРЕСАЦИИ ВНЕШНЕЙ ДИСКОВОЙ ПАМЯТИ.

**Контрольные вопросы:**

1) Что такое файловая система из чего она состоит?
2) Что такое файл?
3) Что включает в себя файловая система как часть ОС?
4) Опишите логическую организацию файловой системы
5) Что такое диск из чего он состоит?
6) Что такое кластер?
7) Опишите взаимосвязь диска, раздела, сектора и кластера


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Операционные системы: Учебник для вузов. 2-е изд. /А.В. Гордеев. – СПб.: Питер, 2006. - 416 с.: ил.


## ЛЕКЦИЯ №12

**ТЕМА: Физическая организация файловой системы. Взаимосвязь с логической организацией ФС.**

**ЦЕЛЬ: изучить принцип работы физической организации файловой системы. Изучить взаимосвязь логической и физический организации ФС**

Напомним, что Операционная система при работе с диском использует, как правило, собственную единицу дискового

пространства, называемую **кластером** *(cluster)* . При создании файла место на диске ему выделяется кластерами.

Величина кластера определяется емкостью данного диска: чем больше диск, тем больше значение кластера. В РЕАЛЬНОСТИ ИМЕННО КЛАСТЕР ЯВЛЯЕТСЯ ЕДИНИЦЕЙ АДРЕСАЦИИ ВНЕШНЕЙ ДИСКОВОЙ ПАМЯТИ.

## ФИЗИЧЕСКАЯ ОРГАНИЗАЦИЯ

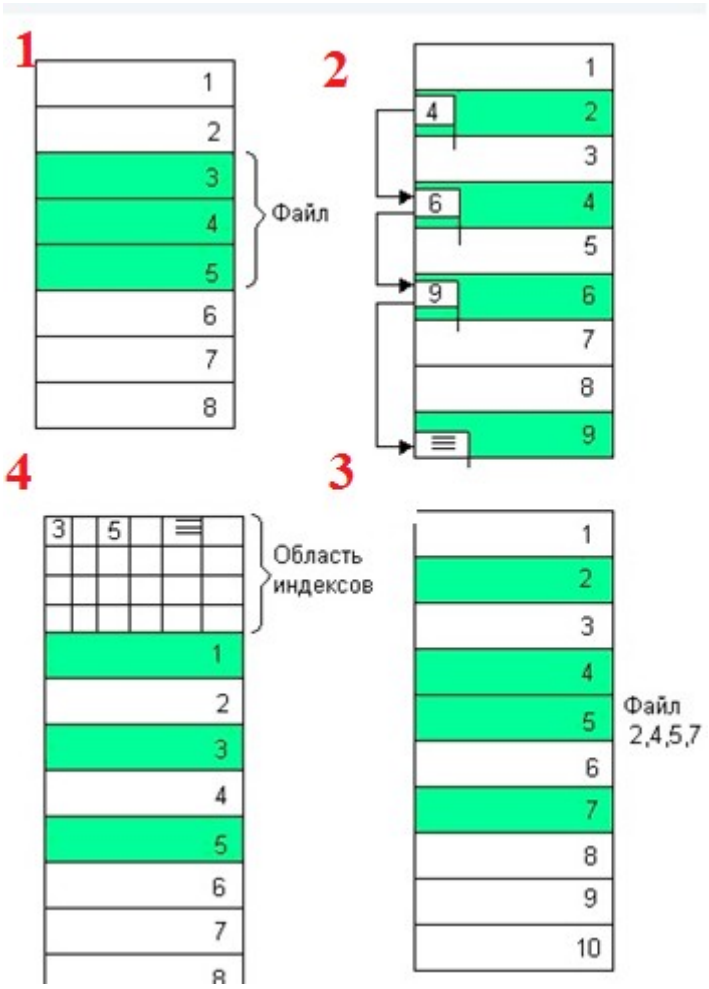
Файл, имеющий образ цельного, непрерывающегося набора байт, на самом деле очень часто разбросан «кусочками» по всему диску, причем это разбиение никак не связано с логической структурой файла,

Отдельная логическая запись файлов может быть расположена в несмежных секторах диска. Логически объединенные файлы из одного каталога совсем не обязаны соседствовать на диске. **Пример** файлы, которые у нас объедены в одну папку, не находятся в одном дисковом пространстве

Принципы размещения файлов, каталогов и системной информации на реальном устройстве описываются физической организацией файловой системы. Очевидно, что разные файловые системы имеют разную физическую организацию.

Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске. Файл состоит из физических записей - блоков. **Блок** - наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью.

**Основными критериями эффективности физической организации файлов являются:**




1. скорость доступа к данным;
2. объем адресной информации файла;
3. степень фрагментированности дискового пространства;
4. максимально возможный размер файла.

Важным компонентом физической организации файловой системы является физическая организация файла, то есть способ размещения файла на диске.

**Способы размещения файла на диске бывают:**

1. **Непрерывное размещение**
2. **Размещение файла в виде связанного списка кластеров** дисковой памяти
3. **Использование связного списка индексов**
4. **Простое перечисление номеров кластеров**

**1) Непрерывное размещение** — простейший вариант физической организации, при котором файлу предоставляется последовательность кластеров диска, образующих непрерывный участок дисковой памяти.

**2) размещение файла в виде связанного списка кластеров** дисковой памяти . При таком способе В НАЧАЛЕ КАЖДОГО КЛАСТЕРА СОДЕРЖИТСЯ УКАЗАТЕЛЬ НА СЛЕДУЮЩИЙ КЛАСТЕР. В этом случае адресная информация минимальна: расположение файла может быть задано одним числом — номером первого кластера.

Популярным способом, применяемым, например, в файловой системе FAT(3)

**3) использование связанного списка индексов** . Этот способ является некоторой модификацией предыдущего. Файлу также выделяется память в виде связанного списка кластеров. Номер первого кластера запоминается в записи каталога, где хранятся характеристики этого файла. Остальная адресная информация отделена от кластеров

файла. С каждым кластером диска связывается некоторый элемент — индекс. Индексы располагаются в отдельной области диска.

4) **простое перечислении номеров кластеров**, способ заключается в перечислении номеров кластеров. Этот перечень и служит адресом файла.

**Недостаток** данного способа очевиден: длина адреса зависит от размера файла и для большого файла может составить значительную величину.

**Достоинством** же является высокая скорость доступа к произвольному кластеру файла, так как здесь применяется прямая адресация, которая исключает просмотр цепочки указателей при поиске адреса произвольного кластера файла. Фрагментация на уровне кластеров в этом способе также отсутствует.

Последний подход с некоторыми модификациями используется в традиционных файловых системах **ОС UNIX s5 и ufs**. Для сокращения объема адресной информации прямой способ адресации сочетается с косвенным.

**Взаимосвязь логической ФС и физической**

Одной из основных задач ОС является предоставление удобств пользователю при работе с данными, хранящимися на дисках. Для этого ОС подменяет физическую структуру хранящихся данных некоторой удобной для пользователя логической моделью, которая реализуется в виде дерева каталогов,( выводимого на экран такими утилитами, как Norton Commander, Far Manager или Windows Explorer). Базовым элементом этой модели является файл, который так же, как и файловая система в целом, может характеризоваться как логической, так и физической структурой.

**Контрольные вопросы:**

**8)** Какие способы размещения файла на диске бывают при физической организации

**9)** Опишите физическую организацию файловой системы

**10)** Чем отличается логическая организация от физической организации файловой системы?


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Операционные системы: Учебник для вузов. 2-е изд. /А.В. Гордеев. – СПб.: Питер, 2006. - 416 с.: ил.


## ЛЕКЦИЯ №13

**ТЕМА: Физическая организация FAT. Основные возможности файловой системы NTFS**

**ЦЕЛЬ: изучить сановные файловые системы. Понять в чем разница между FAT и NTFS, в каких случаях какие файловые системы нужно применять.**

Лекция будет посвящается ф***айловым системам***. При установке ОС Windows предлагает выбрать файловую систему на разделе, где она будет устанавливаться, и пользователи ПК должны выбирать из двух вариантов **FAT** или **NTFS**.

В большинстве случаев пользователи довольствуются знанием, что **NTFS «лучше»**, и выбирают этот вариант.

Однако иногда им становится интересно, **а чем именно лучше?**

В данной лекции я постараюсь объяснить, **что такое файловая система, какие они бывают, чем отличаются, и какую стоит использовать.**

Для того чтобы упорядочить данные на физических носителях, обязательным условием является наличие файловой системы. Этот способ размещения данных определяет, каким образом будет предоставлен доступ операционной системы к файлам.

Обычно файловая система записана в начале жесткого диска.

С точки зрения ОС, жесткий диск – это набор кластеров.

*Кластер* – это область диска определенного размера для хранения данных.

Для жестких дисков в ПК в данный момент наиболее распространены две файловые системы: **FAT** или **NTFS**. Сначала появилась **FAT (FAT16)**, затем **FAT32**, а потом **NTFS**.

При форматировании HDD или флеш накопителя система предоставляет пользователю выбор, какой вид файловой системы (FAT16, FAT32, exFAT, NTFS) будет организован на этом носителе. Ввиду того, что FAT16 является уже историей в мире IT, а exFAT еще новая и малораспространенная система, на сегодняшний день наиболее популярными файловыми системами являются FAT32 и NTFS.

**В чем отличия FAT32 и NTFS? Начнем с азработки**

Структура FAT была разработана Биллом Гейтсом и Марком МакДональдом в 1977 году. Использовалась в качестве основной файловой системы в операционных системах DOS и Microsoft Windows (до версии Windows ME).

**FAT** (*Таблица Размещения Файлов*)— это классическая архитектура файловой системы, которая из-за своей простоты всё ещё широко используется для флеш-накопителей. Используется в дискетах, и некоторых других носителях информации. Ранее использовалась и на жестких дисках.

**Таблица размещения файлов FAT** — это файловая система, в основе которой лежит электронная таблица данных. Существуют две наиболее популярные разновидности данной системы: FAT16 и FAT32. По сути, это однотипные таблицы размещения информации с одной лишь разницей: использование 16-ти или 32-х разрядных адресаций кластеров. В современных системах FAT16 уже не используется, ввиду ее ограниченных возможностей по размеру тома (логического диска).

Существует четыре версии FAT — *FAT12*, *FAT16*, *FAT32* и *exFAT*. Они отличаются количеством бит, отведённых для хранения номера кластера.

*FAT12* применяется в основном для дискет, *FAT16* — для дисков малого объёма, а новая *exFAT* преимущественно для флэш-накопителей. Максимальный размер кластера, который поддерживается в FAT, составляет 64Кб.

*FAT16* впервые представлена в ноябре 1987 года. Индекс *16* в названии показывает, что для номера кластера используется 16 бит. Вследствие этого максимальный объем раздела диска (тома), который может поддерживать эта система, равен 4Гб.

Позже, с развитием технологий и появлением дисков объемом более 4Гб, появилась файловая система ***FAT32*.** Она использует 32-разрядную адресацию кластеров и появилась вместе с Windows 95 OSR2 в августе 1996 года. *FAT32* ограничена в размере тома в 128Гб. Также эта система может поддерживать длинные имена файлов.

**NTFS** (аббревиатура *New Technology File System — Файловая Система Новой Технологии*) — стандартная файловая система для семейства операционных систем Microsoft Windows NT.

**NTFS** – файловая система, в основе которой лежит использование сводной таблицы с информацией о файлах в начале раздела диска, а уже потом размещаются сами файлы. Данная файловая система использует специализированные структуры данных, что позволяет обеспечить высокую надежность и эффективность использования места на жестком диске.

**Основные особенности NTFS:** встроенные возможности разграничивать доступ к данным для различных пользователей и групп пользователей, а также назначать квоты (ограничения на максимальный объём дискового пространства, занимаемый теми или иными

пользователями), использование системы журналирования для повышения надёжности файловой системы.

Спецификации файловой системы являются закрытыми. Обычно размер кластера равен 4Кб. На практике не рекомендуют создавать тома более 2ТБ. Жесткие диски только достигли таких размеров, возможно в будущем нас ждет новая файловая система.

Во время установки ОС Windows предлагается отформатировать диск в системе **FAT** или **NTFS**. При этом имеется в виду **FAT32**.

Все файловые системы построены на принципе: один кластер – один файл. Т.е. один кластер хранит данные только одного файла.

Основное отличие для обычного пользователя между этими системами – размер кластера. «Давным-давно, когда диски были маленькими, а файлы – очень маленькими» это было очень заметно.

**Пример: По аналогии с блокнотом кластер – это клетка. И поставив точку в клетку, мы уже логически занимаем ее всю, а в действительности остается много свободного места.**

*Том* – это синоним раздела диска, пользователи том обычно называют «диск С», Для просмотра, что у нас установлено, ДИСК => СВОЙСТВА

В настоящий момент широко используются диски объемом 320Гб и больше. Поэтому использовать систему **NTFS** для оптимального использования дискового пространства.

**Основными достоинствами NTFS являются:**

1. **Рациональное использование места на носителе;**
2. **Высокая производительность при работе с большими файлами;**
3. **Значительная безопасность**.( Файловая система поддерживает объектную модель безопасность и рассматривает все тома, каталоги, файлы как самостоятельные объекты. NTFS обеспечивает безопасность на уровне файлов, это означает, что право доступа к файлам зависит от учетной записи польз., и тех групп к кот. он принадлежит)
4. **Поддержка сжатия;**

**5. Надежность** (Высокопроизводительные компьютеры и системы совместного пользования (серверы) должны обладать повышенной надежностью, которая является ключевым элементом структуры и поведения NTFS. Одним из способов увеличения надежности является введение механизма транзакций, при котором осуществляется ведение журнала, куда заносятся записи о всех выполненных файловых операций;)

**6. Восстановление системы при сбоях.(** Файловая система восстанавливает при отказе системы и сбоев дисков. Это достигнуто средствами используемыми механизм транзакции, при котором осуществляется журналирование файловой операции)

**7. Расширенная функциональность**. (NTFS проектировалась с учетом возможного расширения. В ней реализованы такие возможности, как эмуляция других ОС, параллельная обработка потоков данных и создание файловых атрибутов определенных пользователем.)

**8. Гибкость**. (Модель распределения дискового пространства в NTFS отличается чрезвычайной гибкостью. Размер кластера может изменяться от 512 байт до 64 Кбайт; он представляет собой число, кратное внутреннему кванту распределения дискового пространства. *NTFS также поддерживает длинные имена файлов, набор символов Unicode и альтернативные имена формата 8.3 для совместимости с FAT*.)

**Есть у этой системы и ряд недостатков:**

1. Высокая требовательность к объему оперативной памяти;
2. Отсутствие доступа NTFS-томов в MS-DOS;
3. Снижение производительности при работе с малыми объемами томов.

Индустриальные стандарты ограничивают размер таблицы разделов $2^{32}$ секторами. Другим ограничением является размер сектора, который обычно равен 512 байтам. Поскольку размер сектора может измениться в будущем, текущий размер сектора дает ограничение на размер одного тома – 2 Тбайт ( $2^{32} \times 512$ байт = 241 ). Таким образом, размер тома в 2 Тбайт является практическим пределом для физических и логических томов NTFS.

**Сравнение NTFS и FAT 32.**

## NTFS.

**Достоинства:**

1. Быстрая скорость доступа к файлам малого размера;
2. Размер дискового пространства на сегодняшний день практически не ограничен;
3. Фрагментация файлов не влияет на саму файловую систему;
4. Высокая надежность сохранения данных и собственно самой файловой структуры;
5. Высокая производительность при работе с файлами большого размера;

**Недостатки:**

1. Более высокие требования к объему оперативной памяти по сравнению с FAT 32;
2. Работа с каталогами средних размеров затруднена из-за их фрагментации;
3. Более низкая скорость работы по сравнению с FAT 32

## FAT 32

**Достоинства:**

1. Высокая скорость работы;
2. Низкое требование к объему оперативной памяти;
3. Эффективная работа с файлами средних и малых размеров;
4. Более низкий износ дисков, вследствие меньшего количества передвижений головок чтения/записи.

**Недостатки:**

1. Низкая защита от сбоев системы;
2. Не эффективная работа с файлами больших размеров;
3. Ограничение по максимальному объему раздела и файла;
4. Снижение быстродействия при фрагментации;
5. Снижение быстродействия при работе с каталогами

Каждая из представленных файловых систем обладает рядом преимуществ и недостатков. Выбирая одну из них, необходимо определиться, для каких целей будет использован компьютер и какие у него параметры. ЕСЛИ СИСТЕМА УСТАНАВЛИВАЕТСЯ НА МОЩНЫЙ СЕРВЕРНЫЙ КОМПЬЮТЕР, ТО БОЛЕЕ ПОДХОДЯЩИМ ВАРИАНТОМ ОКАЖЕТСЯ NTFS. ПРИ РАБОТЕ НА ДОМАШНЕМ КОМПЬЮТЕРЕ ДОСТАТОЧНЫМ БУДЕТ ИСПОЛЬЗОВАНИЕ FAT32.

**Контрольные вопросы:**

1. Назовите достоинства файловой системы NTFS
2. Какие версии FAT существуют, в чем их различия?
3. В каких случаях лучше использовать FAT  и в каких NTFS?
4. Какие виды файловых систем вы знаете, опишите их.
5. Назовите недостатки NTFS
6. Основные особенности NTFS
7. Назовите недостатки FAT
8. Что такое файловая система, приведите пример

**Список используемых источников:**


- Современные операционные системы, Э. Таненбаум, 2002, СПб, Питер, 1040 стр.
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер
- Сетевые операционные системы Н. А. Олифер, В. Г. Олифер, 2001, СПб, Питер, 544 стр.
- Операционные системы: Учебник для вузов. 2-е изд. /А.В. Гордеев. – СПб.: Питер, 2006. - 416 с.: ил.
- http://yura.puslapiai.lt/archiv/per/fat.html
- https://studopedia.org/3-170736.html
- http://www.starlink.ru/articles/ntfs-vs-fat/
- http://www.pc-user.ru/view_post.php?id=55

# ЛЕКЦИЯ №14

## ТЕМА: Обзор системы Linux. Основные понятия

## ЦЕЛЬ: ознакомится с операционной системой ОС Linux.

На сегодняшний день наиболее известными операционными системами для компьютеров являются семейства операционных систем Microsoft Windows и UNIX. Первые ведут свою родословную от операционной системы MS-DOS, которой оснащались первые персональные компьютеры фирмы IBM.

Операционная система UNIX была разработана группой сотрудников Bell Labs под руководством Денниса Ричи, Кена Томпсона и Брайана Кернигана (Dennis Ritchie, Ken Thompson, Brian Kernighan) в 1969 году.

Но в наши дни, когда говорят об операционной системе UNIX, чаще всего имеют в виду не конкретную ОС, а скорее целое семейство UNIX-подобных операционных систем Linux

**Основные отличия между Linux и Unix?**

*Отличия между Linux и UNIX значительны*. UNIX – широкое понятие, конкретнее говоря, некий фундамент для построения и сертификации всех UNIX-подобных систем, а Linux – одна из веток, UNIX-подобная

Linux – ОС с открытым исходным кодом, распространяется бесплатно, Unix – только ее производные находятся в свободном доступе. Linux можно назвать своеобразным клоном Unix, который не использует его код. Linux изначально разрабатывался для домашних ПК, а Unix для больших корпораций. Правда, на сегодня Linux поддерживает больше платформ чем Unix и является более популярным среди пользователей. И, конечно же, Linux поддерживает больше типов файловых систем чем Unix.

(Линукс) – это операционная система, которая на сегодняшний день является фактически единственной альтернативной заменой ОС Windows от Microsoft.

Свое начало Linux берет с 1991 года, когда молодой программист с Финляндии Линус Торвальдс взялся за работу над самой первой версией системы, которая и была названа в честь его имени.

**Linux** — общее название UNIX-подобных операционных систем на основе одноимённого ядра и собранных для него библиотек и системных программ, разработанных в рамках проекта GNU.

**Можно выделить несколько основных областей, где нередко можно встретить Linux:**

1. Серверы, требующие высокого аптайма.
2. Компьютеры нестандартной архитектуры (например, суперкомпьютеры) — из-за возможности быстрой адаптации ядра операционной системы и большого количества ПО под нестандартную архитектуру.
3. Системы военного назначения (например, МСВС РФ) — по соображениям безопасности.
4. Компьютеры, встроенные в различные устройства (банкоматы, терминалы оплаты, мобильные телефоны, маршрутизаторы, стиральные машины и даже беспилотные военные аппараты) — из-за широких возможностей по конфигурированию Linux под задачу, выполняемую устройством, а также отсутствия платы за каждое устройство.
5. Массовые специализированные рабочие места (например, тонкие клиенты, нетбуки) — также из-за отсутствия платы за каждое рабочее место и по причине их ограниченной

**Особенности и достоинства ОС Линукс**

1. **Бесплатность.** Установив Linux, вы получите набор из тысяч бесплатных программ.

Хоть они и не столь привычны как Windows- программы, но абсолютно функциональны. Все больше людей понимают, что пиратская копия Windows может принести крупные неприятности. А на платную лицензионную версию Windows раскошелится мало кто готов. Так же как и на покупку программ, работающих под данной ОС.

2. **Надежность.** Корректная работа аппаратной части вашего ПК, позволит Linux’у работать годы без перезагрузки и зависаний. А кнопка Reset вообще никогда не понадобится.
3. **Безопасность**. . Само построение операционной системы исключает работу вредоносных программ.

В Linux практически нету вирусов и по этому вы можете обойтись без антивирусных программ, тормозящих компьютер и мешающих работать.

Не нужно все время обновлять антивирусные базы и проверять жесткий диск на вирусы, теряя бесценное время.

4. **Открытый исходный код**. Это дает возможность использовать и модифицировать код по своему желанию. Можно в любой момент исправить какие-нибудь ошибки или недочёты системы, а также расширить её функциональность, путём написания дополнений или программ, работающих под ее управлением.

На данный момент вокруг Линукс сформировалось огромное сообщество программистов, которые постоянно усовершенствуют систему. Они разрабатывают новые версии и разновидности данной ОС, пишут самые разнообразные программы, работающие под Linux.

**Линукс-системы представляют собой модульные Unix-подобные операционные системы.** В большей степени дизайн Линукс-систем базируется на принципах, заложенных в Unix в течение 1970-х и 1980-х годов.

Такая система использует монолитное ядро Линукс, которое управляет процессами, сетевыми функциями, периферией и доступом к файловой системе.

Драйверы устройств либо интегрированы непосредственно в ядро, либо добавлены в виде модулей, загружаемых во время работы системы

В отличие от большинства других операционных систем, GNU/Linux не имеет единой «официальной» комплектации. Вместо этого GNU/Linux поставляется в большом количестве так называемых дистрибутивов, в которых программы соединяются с ядром Linux и другими программами.

Большинство пользователей для установки GNU/Linux используют дистрибутивы. **Дистрибутив** — это не просто набор программ, а ряд решений для разных задач пользователей, объединённых едиными системами установки, управления и обновления пакетов, настройки и поддержки. ФАКТИЧЕСКИ ЭТО ЯДРО LINUX И НАБОР РАЗЛИЧННОГО ПО

Некоторые операционные системы Linux используют ядро Linux неизменным, другие изменяют его для получения большей безопасности или реализации необходимых функций.

Преимущества той или иной операционной системы на Linux зависят от набора программного обеспечения, которое в ней используется.

**Самые распространённые в мире дистрибутивы:**

5. **Debian** GNU/Linux — один из старейших дистрибутивов,. Служит основой для создания множества других дистрибутивов. Отличается строгим подходом к включению несвободного ПО.
6. **Ubuntu** — дистрибутив, основанный на Debian. Основная сборка ориентирована на лёгкость в освоении и использовании, при этом существуют серверная и минимальная сборки.
7. **Linux Mint** — дистрибутив, основанный на Ubuntu и полностью с ним совместимый, включающий в себя по умолчанию Java, Adobe Flash и многое другое.
8. **openSUSE** — дистрибутив, отличается удобством в настройке и обслуживании благодаря использованию утилиты YaST.
9. **Fedora** — поддерживается сообществом и корпорацией RedHat.
10. **Slackware** — один из старейших дистрибутивов, отличается консервативным подходом в разработке и использовании.
11. **Gentoo** — дистрибутив, полностью собираемый из исходных кодов. Позволяет очень гибко настраивать конечную систему и оптимизировать производительность, поэтому часто называет себя мета-дистрибутивом. Ориентирован на экспертов и опытных пользователей.
12. **Arch Linux** — ориентированный на применение самых последних версий программ и постоянно обновляемый, поддерживающий одинаково как бинарную, так и установку из исходных кодов дистрибутив ориентирован на компетентных пользователей, которые хотят иметь всю силу и модифицируемость Linux, но не в ущерб времени обслуживания.

# Графические интерфейсы Linux

**Графический интерфейс (GUI)** — оконный менеджер и приложения для работы с файлами и мультимедиа.

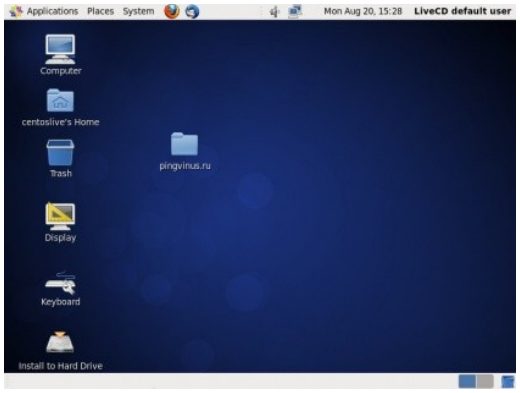
**ГРАФИЧЕСКИЕ интерфейсы разделены на две группы**:

1. среды рабочего стола
2. оконные менеджеры.

**1. Среда рабочего стола Linux** (Desktop Environment) — это комплексная готовая к работе оболочка. Как правило среда рабочего стола включает панель задач, функциональные меню, менеджер входа в систему, программы настройки, базовые программы и другие функциональные элементы, включая оконный менеджер.

**2. Оконный менеджер Linux** (Window Manager) — это программа, которая занимается отрисовкой окон, позволяет перемещать и изменять размер окна, обрабатывает действия пользователя, которые он делает в окне программы. Оконный менеджер может работать независимо или быть в составе среды рабочего стола.

СРЕДЫ РАБОЧЕГО СТОЛА:



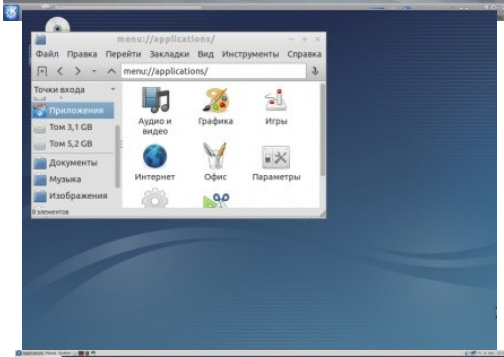

## Gnome

Gnome (GNU Network Object Model Environment) — самая популярная среда рабочего стола для Linux.

Gnome является одной из самых функциональных рабочих сред и включает в себя набор утилит для настройки среды, прикладное программное обеспечение, системные утилиты и другие модули.

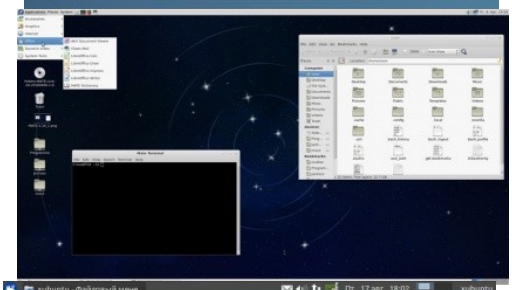
## KDE

KDE — полнофункциональная среда рабочего стола.
В рамках проекта KDE разрабатывается большое количество приложений для повседневных нужд. Рабочий стол KDE изобилует различными графическими эффектами.



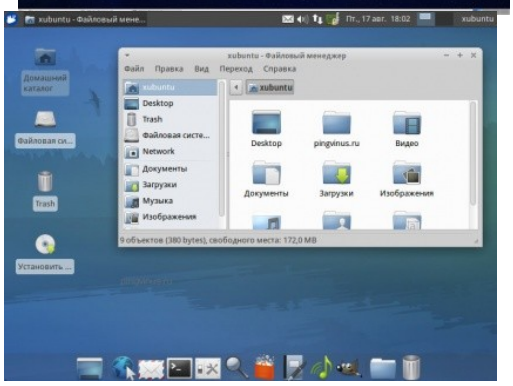

## LXDE

LXDE (Lightweight X11 Desktop Environment) — быстрая легковесная среда рабочего стола, не требовательная к ресурсам компьютера.
Окна и меню открываются без задержек, интерфейс отзывчивый и не вызывает раздражения.



## MATE

MATE — среда рабочего стола, которая является продолжением развития **Gnome 2**. MATE является сбалансированной средой с хорошим набором программ и утилит и приятным классическим интерфейсом.




## Xfce

Xfce (ЕКС Ф СИЕЙ) — легковесное рабочее окружение не требовательное к ресурсам компьютера. Имеет современный интерфейс и при этом потребляет мало оперативной памяти.

# Оконные менеджеры

## Enlightenment

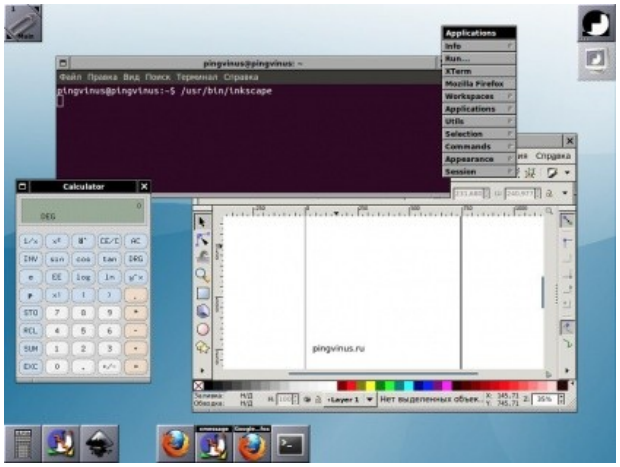
Enlightenment (или просто E) — легковесный **оконный** менеджер не требовательный к ресурсам компьютера, потребляет очень мало оперативной памяти.

Поддерживается анимация элементов интерфейса, темы, виртуальные рабочие столы.
Интерфейс нельзя назвать очень стильным и современным, он требует привыкания.

## Openbox

Openbox — легковесный оконный менеджер с простым **минималистским** интерфейсом. Данный оконный менеджер не требователен к системным ресурсам и работает очень быстро.
**При клике правой кнопкой мыши вызывается главное меню Openbox**, через которое можно вызывать любые программы. Openbox хорошо настраивается и поддерживает темы оформления.

## Window Maker



Window Maker — менеджер окон для Linux.

Главными элементами интерфеса в Window Maker являются функциональные кнопки на рабочем столе и меню, вызываемое при клике правой кнопкой мыши по рабочему столу. Работает быстро, хорошо настраивается.

### ***Пользователь***

С самого начала ОС UNIX замышлялась как интерактивная система. Другими словами, UNIX предназначен для терминальной работы(**Терминал** часто называют командной строкой или оболочкой.). Чтобы начать работать, человек должен "войти" в систему, введя со свободного терминала свое учетное имя (account name) и, возможно, пароль (password). Человек, зарегистрированный в учетных файлах системы, и, следовательно, имеющий учетное имя, называется -**зарегистрированным пользователем системы**.

Регистрацию новых пользователей обычно выполняет администратор системы. Пользователь не может изменить свое учетное имя, но может установить и/или изменить свой пароль.

Пароли хранятся в отдельном файле в закодированном виде. Не забывайте свой пароль, снова узнать его не поможет даже администратор!

Все пользователи ОС UNIX явно или неявно работают с файлами. **Файловая система ОС UNIX имеет древовидную структуру**. *Промежуточными узлами дерева являются каталоги со ссылками на другие каталоги или файлы, а листья дерева соответствуют файлам или пустым каталогам*.

**Что доступно зарегистрированному Пользователю?** Каждому зарегистрированному пользователю соответствует некоторый каталог файловой системы, который называется "домашним" (home) каталогом пользователя. При входе в систему пользователь получает неограниченный доступ к своему домашнему каталогу и всем каталогам и файлам, содержащимся в нем.

Пользователь может создавать, удалять и модифицировать каталоги и файлы, содержащиеся в домашнем каталоге. Потенциально возможен доступ и ко всем другим файлам, однако он может быть ограничен, если пользователь не имеет достаточных привилегий.

## ***Привилегированный пользователь***

Ядро ОС UNIX идентифицирует каждого пользователя по его идентификатору (**UID - *User Identifier***), уникальному целому значению, присваиваемому пользователю при регистрации в системе. Кроме того, каждый пользователь относится к некоторой группе пользователей, которая также идентифицируется некоторым целым значением (**GID - *Group IDentifier***).

Значения UID и GID для каждого зарегистрированного пользователя сохраняются в учетных файлах системы и приписываются процессу, в котором выполняется командный интерпретатор, запущенный при входе пользователя в систему. Эти значения наследуются каждым новым процессом, запущенным от имени данного пользователя, и *используются ядром системы для контроля правомощности доступа к файлам, выполнения программ* и т.д.

Понятно, что администратор системы, который, естественно, тоже является зарегистрированным пользователем, должен обладать большими возможностями, чем обычные пользователи.

В ОС UNIX эта задача решается путем выделения одного значения **UID (нулевого)**. **Пользователь** с таким UID называется ***суперпользователем*** **(`superuser`) или `root`**.

***суперпользователем*** Он имеет неограниченные права на доступ к любому файлу и на выполнение любой программы. Кроме того, такой пользователь имеет возможность полного контроля над системой. Он может остановить ее и даже разрушить.

В мире UNIX считается, что человек, получивший статус суперпользователя, должен понимать, что делает.

*Суперпользователь отвечает за безопасность системы, ее правильное конфигурирование, добавление и исключение пользователей, регулярное копирование файлов и т.д.*

Еще одним отличием суперпользователя от обычного пользователя ОС UNIX является то, что на суперпользователя не распространяются ограничения на используемые ресурсы.

Для обычных пользователей устанавливаются такие ограничения как максимальный размер файла, максимальное число сегментов разделяемой памяти, максимально допустимое пространство на диске и т.д. Суперпользователь может изменять эти ограничения для других пользователей, но на него они не действуют.

**Контрольные вопросы:**

1. **Дайте определение Linuxс ОС**
2. **Назовите и опишите самые распространенные дистрибутивы**
3. **Основные отличия между Linux и Unix?**
4. **Особенности и достоинства ОС Linux?**
5. **Какая архитектура ядра заложена в ОС Linux?**
6. **Назовите графические интерфейсы и опишите ,что они делают**
7. **Чем пользователь отличается от root пользователя?**
8. **Чтобы начать работу, что должен сделать пользователь?**
9. **Как организована файловая система?**

**Список используемых источников:**


- Эви Немет, Гарт Снайдер, Трент Хейн, Бэн Уэйли. Unix и Linux: руководство системного администратора. Как установить и настроить Unix и Linux = Unix and Linux System
- Administration Handbook. — 4-е изд. — М.: Вильямс, 2012. — 1312 с. — ISBN 978-5-8459-1740-9.
- http://help.ubuntu.ru/wiki/linux
- http://citforum.ru/operating_systems/linux1/index.shtml
- http://pingvinus.ru/gui
- https://www.sao.ru/hq/sts/linux/doc/os_unix_ru/glava_2.html