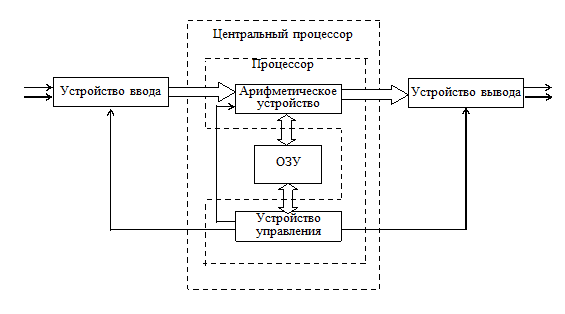
**1. Вычислительная система, ОС, функции ОС.**

**Электронно-Вычислительная Машина** – это вычислительная система, содержащая технические устройства, т.е. центральный процессор, устройства ввода-вывода, пульт управления и т.д. и программное обеспечение, т.е. операционную систему, транслятор, диагностические программы, другие программы общего пользования и т.п.



**Операционная система** — комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

**Основные функции:**

1) Исполнение запросов программ (ввод и вывод данных, запуск и остановка других программ, выделение и освобождение дополнительной памяти и др.).

2) Загрузка программ в оперативную память и их выполнение.

3) Стандартизированный доступ к периферийным устройствам (устройства ввода-вывода).

4) Управление оперативной памятью (распределение между процессами, организация виртуальной памяти).

5) Управление доступом к данным на энергонезависимых носителях (таких как жёсткий диск, оптические диски и др.), организованным в той или иной файловой системе.

6) Обеспечение пользовательского интерфейса.

7) Сохранение информации об ошибках системы.

8) Параллельное или псевдопараллельное выполнение задач (многозадачность).

9) Эффективное распределение ресурсов вычислительной системы между процессами.

10) Разграничение доступа различных процессов к ресурсам.

**2. Принципы построения ОС. Архитектура ОС.**

1. Принцип модульности. Модуль – функционально законченный элемент, выполняемый в соответствии с принятым межмодульным интерфейсом. Модуль выделяется по функциональному признаку. Модульная организация позволяет легко изменять неправильно работающие модули в ОС.

2. Принцип функциональной избирательности. Для организации эффективной работы ОС, необходимо выделить некоторые модули и хранить их в ОЗУ. Эти модули составляют ядро ОС.

3. Принцип генерируемости ОС. Подразумевает собой возможность генерации ОС в зависимости от аппаратного обеспечения. Процесс генерации обычно производится один раз, перед достаточно долгим режимом эксплуатации.

4. Принцип функциональной избыточности. В состав ОС должно входит несколько типов ПО для выполнения одинаковых функций (поддержка разных файловых систем).

5. Принцип виртуализации. Позволяет представить ресурсы ОС в виде определённого набора планировщиков и мониторов и использует единую схему распределения ресурсов.

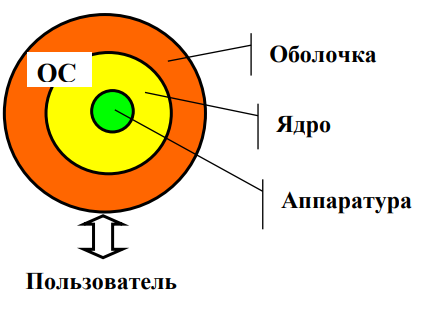
6. Принцип независимости программ от внешних устройств. Связь программ с конкретным внешним устройством производится не на этапе трансляции, а на этапе выполнения программы.

7. Принцип совместимости. Способность выполнять программы для другой ОС или даже для другой аппаратной платформы.

8. Принцип открытой и наращиваемой ОС.

9. Принцип мобильности. ОС должна легко переноситься на другую аппаратную платформу.

10. Принцип обеспечения безопасности вычислений.

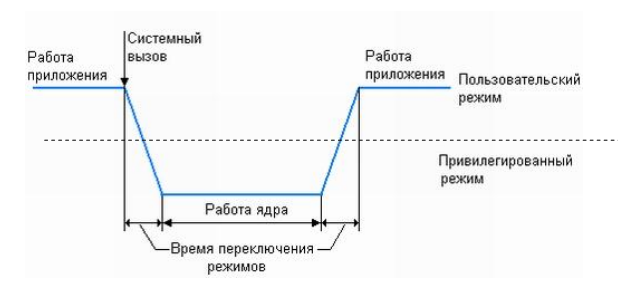


**3. Понятие ядра ОС. Привилегированный и пользовательский режимы.**

**Ядро** — центральная часть операционной системы, обеспечивающая приложениям координированный доступ к ресурсам компьютера, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации. Также обычно ядро предоставляет сервисы файловой системы и сетевых протоколов.

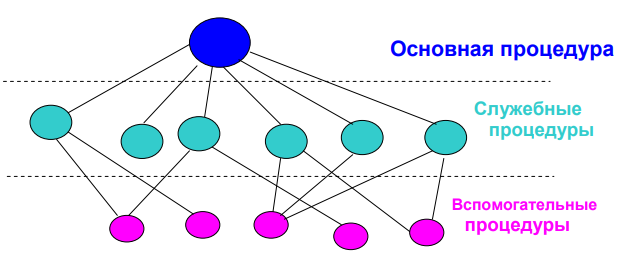


**4. Структура ядра. Механизм системных вызовов.**



**5. Основные структуры ОС: монолитные, многоуровневые, микроядерные системы. Сравнение.**

Первые ОС разрабатывались как **монолитные** **системы** без какой-либо выраженной структуры и представляли собой набор функций и самостоятельных программ, каждая из которых могла беспрепятственно вызывать любую другую. Постепенно такой подход к проектированию привел к возникновению ряда трудностей: во-первых, всё сложнее становилось обеспечить межмодульное взаимодействие, во-вторых, такие монолитные системы почти не способны развиваться, над ними крайне трудно работать большому коллективу программистов.



**Многоуровневые системы** — это системы с отдельно выделенными, иерархически расположенными, блоками, которые организованы по определенной структуре. Подсистемы, характеризующиеся рядом однородных характеристик по отношению к вышестоящим подсистемам, определяют уровень иерархической системы.



Недостатки многоуровневой архитектуры: Негибкости изменений. В случае внесения серьёзных изменений в модули уровня, влияние на смежные уровни может оказаться непредсказуемым и пагубным; Вторым существенным недостатком является трудоёмкость обеспечения безопасности при множественных межуровневых взаимодействиях.

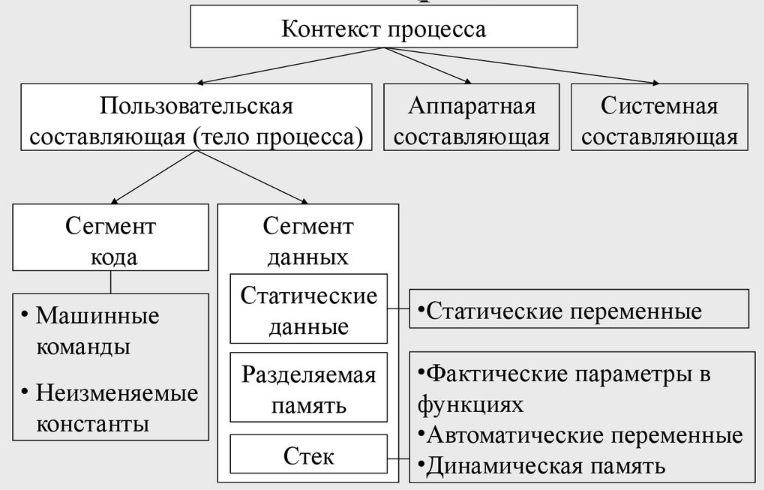
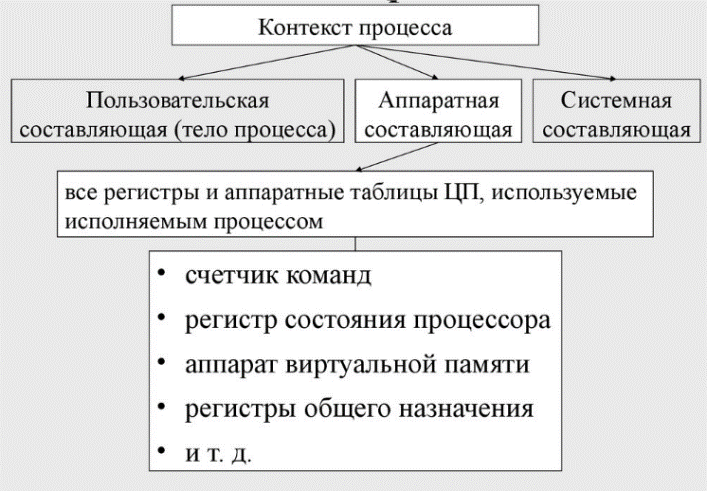
Поэтому альтернативой классическому варианту часто используют **микроядерную** **архитектуру** **ОС**.

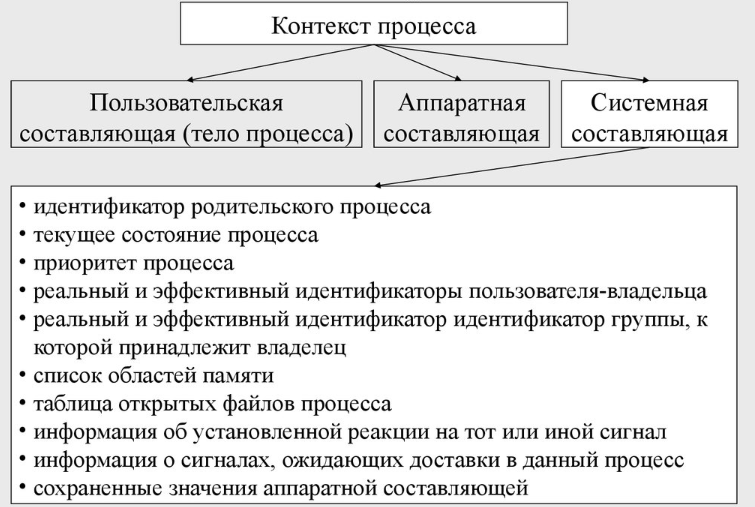
Содержательно **микроядерная** **архитектура** означает, что в привилегированном режиме остаётся работать только небольшая часть ОС — микроядро — защищённая от остальных частей и приложений. В его состав входят машинно-зависимые модули, а также модули, реализующие базовые механизмы ядра. Прочие модули функционируют в пользовательском режиме.

**6. Понятие процесса. Жизненный цикл процесса.**

**Процесс** — совокупность машинных команд и данных, которая обрабатывается в рамках вычислительной системы и обладает правами на владение некоторым набором ресурсов.

**7. Контекст процесса. Полновесные и легковесные процессы.**





**Полновесные процессы** — процессы, выполняющиеся внутри защищенных участков оперативной памяти.

**Легковесные** **процессы** — работают в мультипрограммном режиме одновременно с активировавшей их задачей и используют ее виртуальное адресное пространство.

**8. Управление процессами. Взаимодействующие процессы. Проблема взаимного исключения.**

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является **подсистема управления процессами**. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

**Взаимодействующие процессы** — это процессы, которые совместно используют некоторые переменные, и выполнение одного процесса может повлиять на выполнение другого.

**Взаимное исключение** — предотвращение одновременного обращения процессов к разделяемым переменным.

**9. Реализация взаимного исключения.**

**Семафоры Дейкстры** — формальная модель синхронизации, предложенная голландским учёным Дейкстрой, которая основывается на следующем предположении: имеется тип данных, именуемый семафором. Переменная типа семафор имеет целочисленные значения. Над семафорами определены две операции:

1) Down ( S ) (или P ( S ))‏

2) Up ( S ) (или V ( S ))‏

Операция **down** проверяет значение семафора, и если оно больше нуля, то уменьшает его на 1. Если же это не так, процесс блокируется, причем операция **down** считается незавершенной. Операция **up** увеличивает значение семафора на 1.

**Монитор** представляет собой языковую конструкцию, т.е. некоторое средство, предоставляемое языком программирования и *поддерживаемое компилятором*. **Монитор** представляет собой совокупность процедур и структур данных, объединенных в программный модуль специального типа.

Постулируются три основных свойства монитора:

1. Структуры данных, входящие в монитор, могут быть доступны только для процедур, входящих в этот монитор (таким образом, монитор представляет собой некоторый аналог объекта в объектно-ориентированных языках и реализует инкапсуляцию данных)

2. Процесс «входит» в монитор путем вызова одной из его процедур

3. В любой момент времени внутри монитора может находиться не более одного процесса. Если процесс пытается попасть в монитор, в котором уже находится другой процесс, он блокируется. Таким образом, чтобы защитить разделяемые структуры данных, их достаточно поместить внутрь монитора вместе с процедурами, представляющими критические секции для их обработки.

**10. Планирование процессов. Классификация алгоритмов планирования.**

**Планирование**— это разделение вычислительных ресурсов системы между процессами и потоками.В различных средах требуются различные алгоритмы планирования.

Это связано с тем, что различные операционные системы и различные приложения ориентированы на разные задачи. Можно выделить три среды:

1. Системы пакетной обработки данных.

2. Интерактивные системы.

3. Системы реального времени.

В системах пакетной обработки нет пользователей, сидящих за терминалами и ожидающих ответа. Такой метод уменьшает количество переключений между процессами и улучшает эффективность.

В интерактивных системах необходимы алгоритмы планирования с переключениями, чтобы предотвратить захват процессора одним процессом. Из-за ошибки в программе один процесс может заблокировать остальные. Для исключения подобных ситуаций используется планирование с переключениями.  
В системах с ограничениями реального времени приоритетность не всегда обязательна, поскольку процессы знают, что их время ограничено, и бы­стро выполняют работу, а затем блокируются. Отличие от интерактивных систем в том, что в системах реального времени работают только программы, предназна­ченные для содействия конкретным приложениям.

**11. Планирование процессов. Задачи и цели планирования.**

# Для всех типов систем=

1. Справедливость - предоставление каждому процессу справедливой доли процессорного времени.

2. Принудительное применение политики - контроль за выполнением принятой политики.

3. Баланс - поддержка занятости всех частей системы.

# Для систем пакетной обработки данных=

1. Пропускная способность - максимальное количество задач в час

2. Оборотное время - минимизация времени, затрачиваемого на ожидание, обслуживание и обработку задачи.

3. Использование процессора - поддержка постоянной занятости процессора

# Для интерактивных систем=

1. Время отклика - быстрая реакция на запросы

2. Соразмерность - выполнение пожеланий пользователя

# Для систем реального времени=

1. Окончание работы к сроку - предотвращение потери данных

2. Предсказуемость - предотвращение деградации качества в мультимедийных системах

**13. Управление памятью. Методы распределения памяти без использования внешней памяти.**

**Распределение памяти фиксированными разделами.** Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Очередная задача, поступившая на выполнение, помещается либо в общую очередь, либо в очередь к некоторому разделу.

**Подсистема управления памятью в этом случае выполняет следующие задачи:**

1) выбирает подходящий раздел, сравнивая размер программы, поступившей на выполнение, и свободных разделов;

2) осуществляет загрузку программы и настройку адресов.

Преимущество – простота реализации. Недостаток – жесткость. Даже если программа имеет небольшой объем, она будет занимать весь раздел, что приводит к неэффективному использованию памяти.

**Распределение памяти разделами переменной величины.** Память машины не делится заранее на разделы. Каждой поступающей задаче выделяется необходимая ей память. Если память отсутствует, то задача не принимается и стоит в очереди. После завершения задачи память освобождается, и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных разделов произвольного размера.

**Задачами операционной системы при реализации данного метода управления памятью является:**

1) ведение таблиц свободных и занятых разделов, в которых указываются начальные адреса и размеры участков памяти;

2) при поступлении новой задачи – анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи;

3) загрузка задачи в выделенный ей раздел;

4) корректировка таблиц свободных и занятых областей.

Выбор раздела для вновь поступившей задачи может осуществляться по разным правилам:

5) первый попавшийся раздел достаточного размера;

6) раздел, имеющий наименьший достаточный размер.

Недостаток – фрагментация памяти. Фрагментация – это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

**14. Сегментное, страничное, сегментно- страничное распределение памяти.**

Страничная организация виртуального адресного пространства не позволяет дифференцировать способы доступа к разным частям программы. При сегментном способе организации памяти виртуальное адресное пространство процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т.п. Иногда сегментация программы выполняется по умолчанию компилятором.

При загрузке процесса часть сегментов помещается в оперативную память (при этом для каждого из этих сегментов операционная система подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. **Во время загрузки система создает таблицу сегментов процесса, в которой для каждого сегмента указывается:**

- начальный физический адрес сегмента в оперативной памяти;

- размер сегмента;

- правила доступа;

- признаки модификации, присутствия, обращения к данному сегменту.

Cистема с сегментной организацией функционирует аналогично системе со страничной организацией: время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются, при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Кроме того, при обращении к памяти проверяется, разрешен ли доступ требуемого типа к данному сегменту. Виртуальный адрес при сегментной организации памяти может быть представлен парой (*g, s),* где *g* - номер сегмента, а *s* - смещение в сегменте. Физический адрес получается путем сложения начального физического адреса сегмента, найденного в таблице сегментов по номеру *g*, и смещения *s*. Недостатком данного метода распределения памяти является фрагментация на уровне сегментов и более медленное по сравнению со страничной организацией преобразование адреса.

**15. Алгоритмы замещения страниц**

**Алгоритм FIFO. Выталкивание первой пришедшей страницы**

Каждой странице присваивается временная метка. Реализуется это созданием очереди страниц, в конец которой страницы попадают, когда загружаются в физическую память, а из начала берутся, когда требуется освободить память. Для замещения выбирается старейшая страница. Эта стратегия с достаточной вероятностью будет приводить к замещению активно используемых страниц, например, страниц кода текстового процессора при редактировании файла.

**Оптимальный алгоритм (OPT)**

Замещает страницу, которая не будет использоваться в течение самого длительного периода времени. Каждая страница должна быть помечена числом инструкций, которые будут выполнены, прежде чем на эту страницу будет сделана первая ссылка. Выталкиваться должна страница, для которой это число наибольшее.

Этот алгоритм легко описать, но реализовать невозможно. ОС не знает, к какой странице будет следующее обращение. Данный алгоритм применяется для оценки качества реализуемых алгоритмов.

**Выталкивание дольше всего не использовавшейся страницы. Алгоритм LRU**

LRU - хороший, но труднореализуемый алгоритм. Необходимо иметь связанный список всех страниц в памяти, в начале которого будут хранится недавно использованные страницы. Причем этот список должен обновляться при каждом обращении к памяти. Много времени нужно и на поиск страниц в таком списке.

**Выталкивание редко используемой страницы. Алгоритм NFU**

Для него требуются программные счетчики, по одному на каждую страницу, которые сначала равны нулю. При каждом прерывании по времени операционная система сканирует все страницы в памяти и у каждой страницы с установленным флагом обращения увеличивает на единицу значение счетчика, а флаг обращения сбрасывает.

Кандидатом на освобождение оказывается страница с наименьшим значением счетчика, как страница, к которой реже всего обращались. Главный недостаток алгоритма NFU состоит в том, что он ничего не забывает. Например, страница, к которой очень часто обращались в течение некоторого времени, а потом обращаться перестали, все равно не будет удалена из памяти, потому что ее счетчик содержит большую величину.

Другим, уже более устойчивым недостатком алгоритма является длительность процесса сканирования таблиц страниц.

**16. Файловые системы. Определение. Имена и типы файлов. Логическая организация ФС.**

**Файловая система** – это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

**Файл** – именованная область внешней памяти, предназначенная для считывания и записи данных.

Задачи, решаемые ФС, зависят от способа организации вычислительного процесса в целом. Самый простой тип – это ФС в однопользовательских и однопрограммных ОС.

**Основные функции в такой ФС нацелены на решение следующих задач:**

1. Именование файлов.

2. Программный интерфейс для приложений.

3. Отображения логической модели ФС на физическую организацию хранилища данных.

4. Устойчивость ФС к сбоям питания, ошибкам аппаратных и программных средств.

**Обычные файлы**: содержат информацию произвольного характера, которую заносит в них пользователь или которая образуется в результате работы системных и пользовательских программ. Содержание обычного файла определяется приложением, которое с ним работает.

**Обычные файлы могут быть двух типов:**

1. *Программные*(исполняемые) –  представляют собой программы, написанные на командном языке ОС, и выполняют некоторые системные функции (имеют расширения .exe, .com, .bat).

2. *Файлы данных* –  все прочие типы файлов: текстовые и графические документы, электронные таблицы, базы данных и др.

**Каталоги** – это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр, или файлы, составляющие один программный пакет), а с другой стороны – это особый тип файлов, которые содержат системную справочную информацию о наборе файлов, сгруппированных пользователями по какому-либо неформальному признаку (тип файла, расположение его на диске, права доступа, дата создания и модификация).

**Специальные файлы** – это фиктивные файлы, ассоциированные с устройствами ввода/вывода, которые используются для унификации механизма доступа к файлам и внешним устройствам. Специальные файлы позволяют пользователю осуществлять операции ввода/вывода посредством обычных команд записи с файлов или чтения из файлов. Эти команды обрабатываются сначала программами ФС, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством (PRN, LPT1 – для порта принтера (символьные имена, для ОС – это файлы), CON – для клавиатуры).

**Символьные имена могут быть трех типов: простые, составные и относительные:**

1. ***Простое*** имя идентифицирует файл в пределах одного каталога, присваивается файлам с учетом номенклатуры символа и длины имени.

2. Полное имя представляет собой цепочку простых символьных имен всех каталогов, через которые проходит путь от корня до данного файла, имени диска, имени файла. Таким образом, полное имя является ***составным***, в котором простые имена отделены друг от друга принятым в ОС разделителем.

3. Файл может быть идентифицирован также ***относительным*** именем. Относительное имя файла определяется через понятие «текущий каталог». В каждый момент времени один из каталогов является текущим, причем этот каталог выбирается самим пользователем по команде ОС. Файловая система фиксирует имя текущего каталога, чтобы затем использовать его как дополнение к относительным именам для образования полного имени файла.