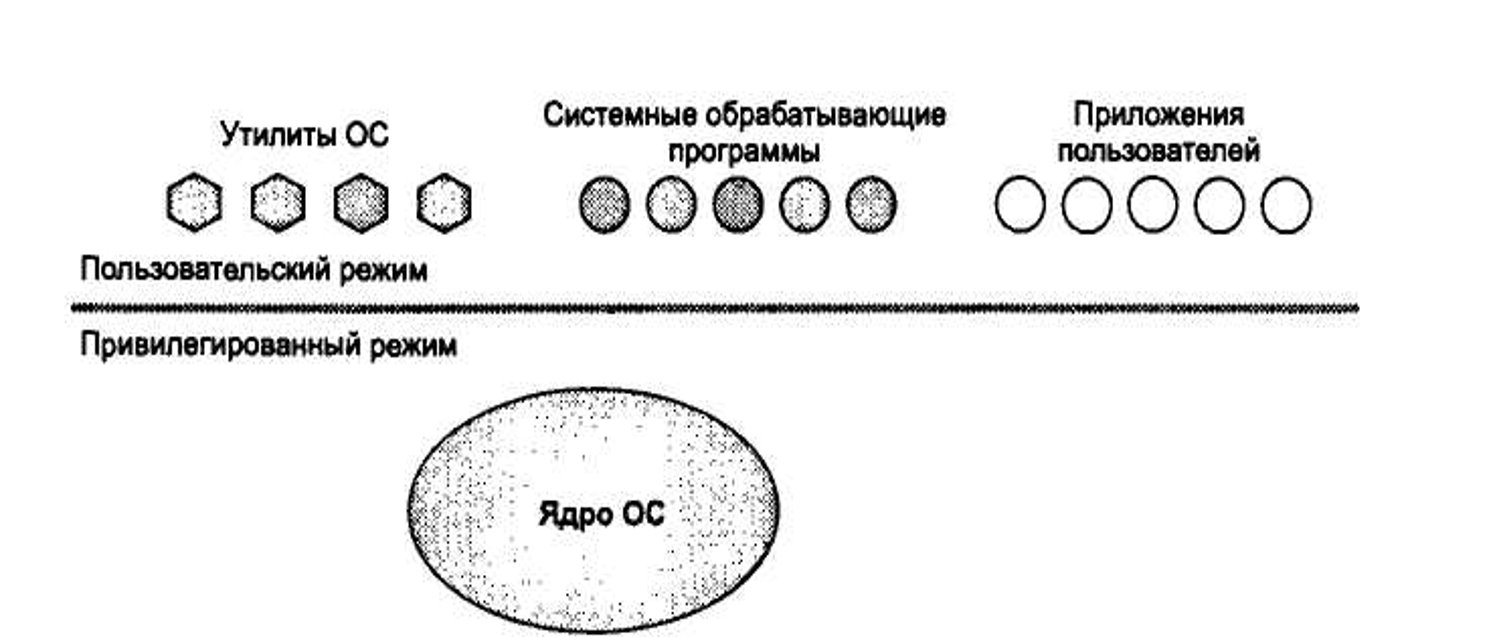
13) **Режим пользователя и режим ядра. Ядро ОС и вспомогательные функции(примеры).**

Программы, запущенные на компьютере, имеют два режима работы:

* режим ядра ;
* и режим пользователя.

В режиме ядра программа имеет полный доступ ко всему аппаратному обеспечению, ей доступны все машинные команды процессора и вся память, она может ссылаться на любую ячейку.

В режиме пользователя программе доступно часть машинных команд процессора и ограниченное поле системной памяти.Переключение занимает значительное количество времени(60-80 тактов). ).

**Ядро ОС работает в привилегированном режиме(режиме ядра). Вспомогательные модули работают в пользовательском режиме.**

Ядро - основная часть ОС, которая обеспечивает запуск всех действий и организует работу других программ.

В зависимости от архитектуры ОС ядро полностью или частично постоянно находится в памяти и все модули или их часть работают в режиме ядра

Функции, входящие в состав ядра можно разделить на два класса :

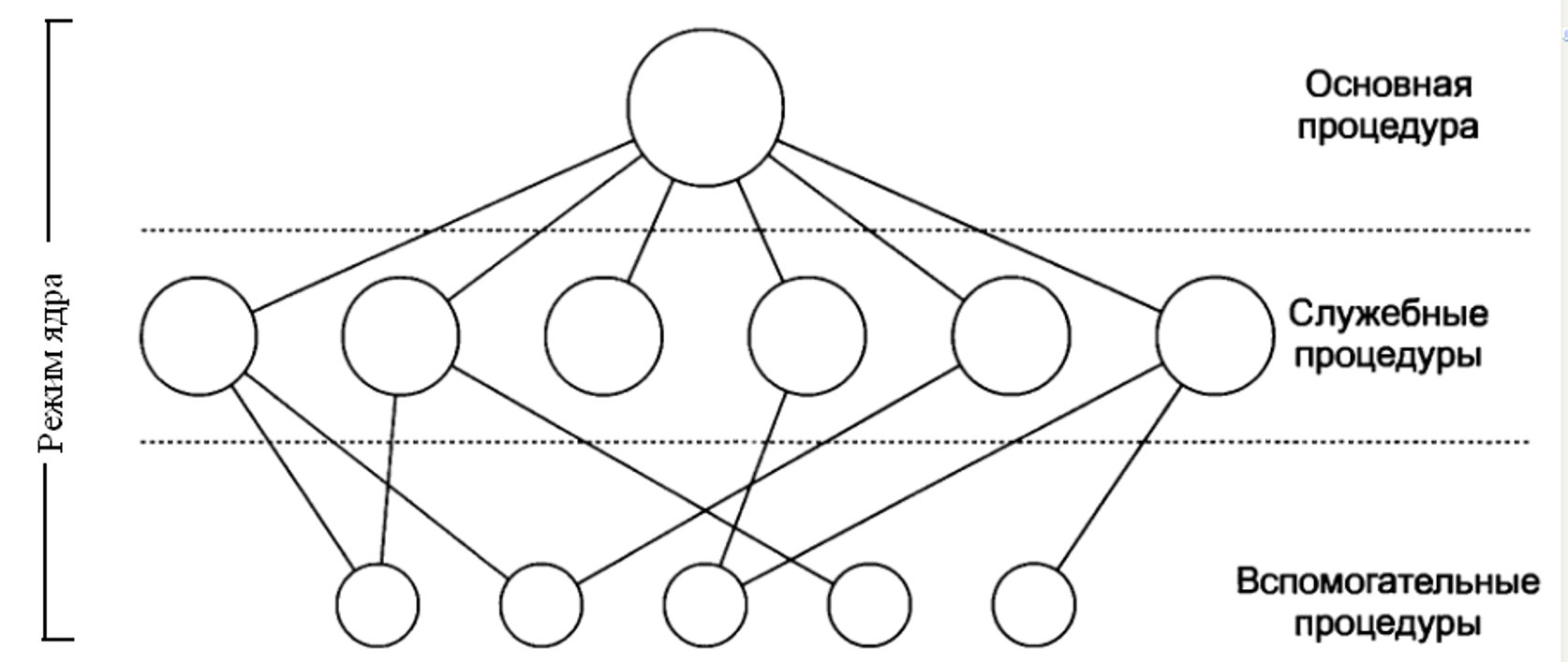
**1 класс.** Функции для управления вычислительным процессом (*эти функции недоступны для приложений)* :

* программами (процессами);
* памятью;
* прерываниями
* вводом -выводом;
* файловой системой

**2 класс**. Функции для поддержки приложений (доступны приложениям). Эти функции создают для приложений 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.
* Вспомогательные модули ОС обычно подразделяются на следующие группы:
* утилиты — программы, решающие отдельные задачи управления и сопровождения компьютерной системы (*например, программы сжатия шифрования дисков (*MS BitLocker, CHKDISK для Windows*), архивирования данных и. т. д*.);
* программы предоставления пользователю дополнительных услуг — (*калькулятор, игры, текстовый редактор, например, Internet Explorer*);
* системные обрабатывающие программы - компиляторы, компоновщики, отладчики, (например компилятор GCC, встроенный в ОС Linux. Однако *чаще они являются частью системы программирования*);
* библиотеки процедур различного назначения, упрощающие разработку приложений (*например, библиотека математических функций, библиотека ввода – вывода)*

*14)* Классификация архитектур ОС. Монолитная и монолитно-уровневая архитектуры ОС. Достоинства и недостатки

**Монолитная архитектура**(пример: MS DOS) 

Одна большая программа, все части которой работают в режиме ядра.

* Состав монолитной ОС :
  + Основная программа, которая может вызывать требуемую служебную процедуру (подпрограмму).
  + Набор служебных процедур, для выполнения какой-либо операции (управление памятью, файлами и.т.д.)
  + Набор вспомогательных процедур, содействующих работе служебных процедур.

Каждая процедура(программа) может свободно вызвать любую другую процедуру(программу), если та выполняет для её какое-нибудь полезное действие. Все части работают в одном адресном пространстве.

Достоинства монолитного ядра:

* скорость работы;
* упрощённая разработка модулей;
* богатство предоставляемых возможностей и функций;
* поддержка большого количества разнообразного оборудования.

Недостатки:

-необходимость перекомпиляции при любых изменениях в составе оборудования.

-плотность программных ошибок(примерно 10 штук на 1000 строк кода)

-повышенные требования к размеру ОП

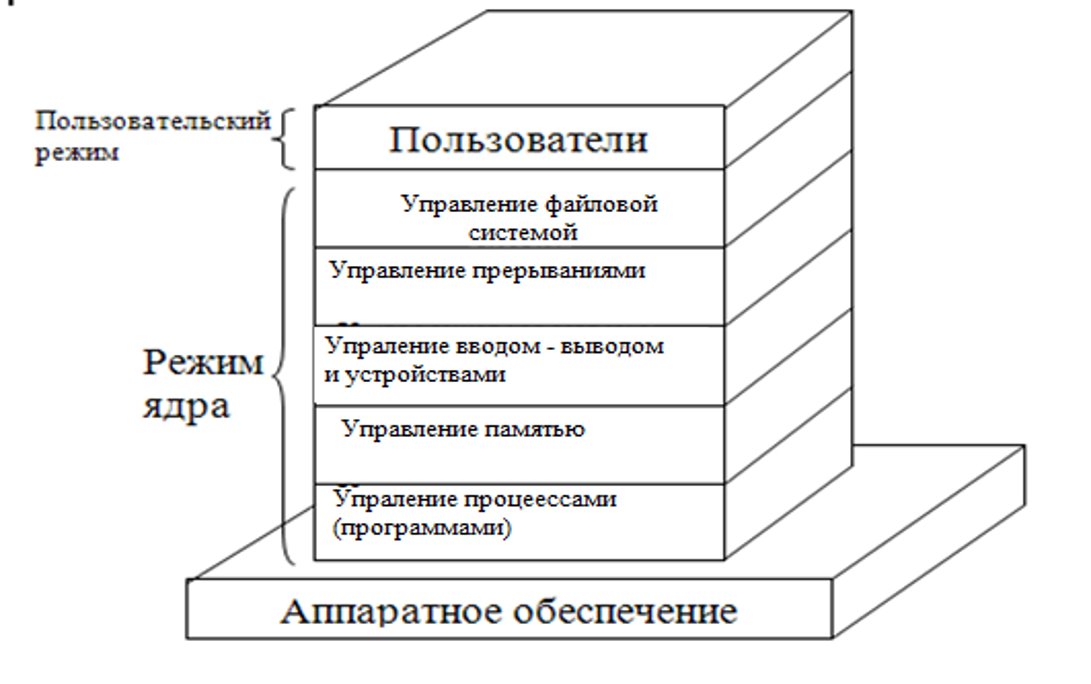
Ограничения:

-расширяемость(возможность добавления функционала)

-переносимость(возможность без полной переработки кода перенести его на другие платформы )

-совместимость с программами, ориентированными на другие ОС или более ранние версии.

**Монолитно-уровневая архитектура(UNIX,Linux,Android)**



Функции ОС реализованы в виде отдельных модулей (уровней), которые могут динамически подгружаться из внешней памяти

Идея многоуровневой архитектуры :

* Полная функциональность операционной системы разделяется на уровни(модули), например, уровень управления памятью, уровень файловой системы, уровень управления процессами и т.п.
* Для каждого уровня определяются интерфейс взаимодействия с соседним уровнем, т.е. некоторый набор правил, согласно которым следует обращаться за услугами данного уровня.
* Каждый уровень может обращаться за услугами только к соседнему нижележащему уровню через его интерфейс.
* Внутренняя структуры каждого уровня не доступна другим уровням.

Преимущества перед монолитной:

-большинство современных монолитных многоуровневых ядер, позволяют во время работы подгружать и выгружать *модули*, выполняющие часть функций ядра Что помогает сократить размер кода. Когда функциональность, предоставляемая модулем больше не требуется, он может быть выгружен. Отпадает необходимость перекомпиляции.

-при внесении нового функционала необходимо внести изменения только в зависимые модули.

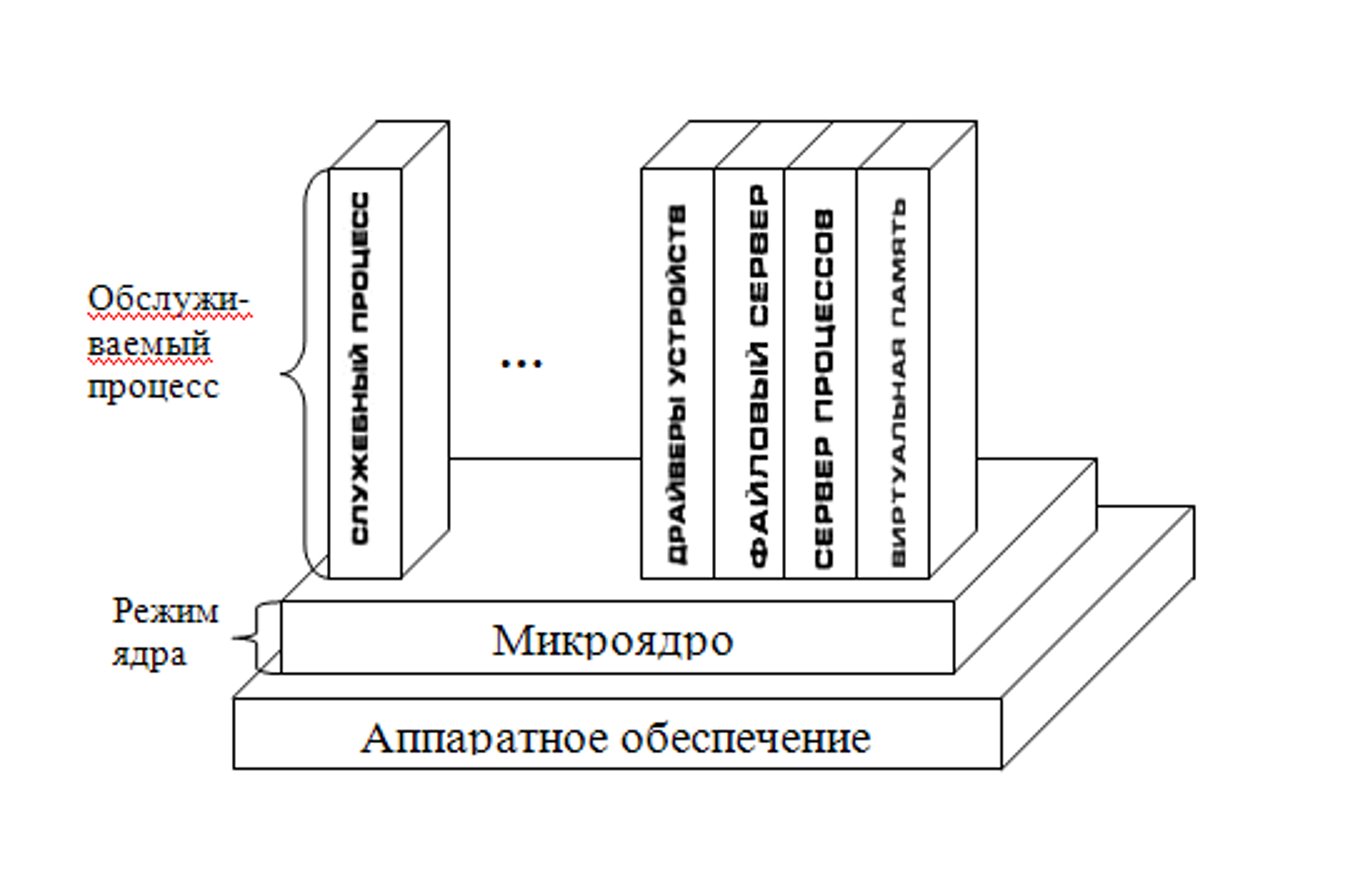
-Переносимость. При переносе операционной системы на другую аппаратную платформу, аппаратно зависимый уровень может быть просто заменен.

Недостатки:

* Интерфейс уровня с остальными уровнями становится громоздким, поэтому заменить уровень новым или нарастить его функциональность становится сложной задачей
* Если потребуется ввести новые функции в уровень, то необходимо вносить изменения и в соседние уровни для возможности обращения к этим функциям.
* Проблема безопасности, т. к. между уровнями много точек обмена.

***15)* Классификация архитектур ОС. Гибридная и микроядерная архитектуры ОС. Достоинства и недостатки**

**Структура на основе микроядра(CISCO IOS)**



Идея структуры типа клиент-сервер на основе микроядра:

* Серверы постоянно находится в состоянии ожидания клиентских запросов.
* Клиенты посылают серверам запросы ( например, запрос на чтение файла, запрос на выделение памяти)
* Получив запрос от клиента, сервер выполняет его, при этом он сам может обратиться за услугами к другим серверам.
* Cервер отсылает клиенту сообщение о завершении задания и результаты работы.
* Клиенты и серверы общаются только через микроядро.
* Одна и та же программа может быть одновременно сервером по отношению к одному виду услуг и клиентом по отношению к другому виду услуг.

Состав микроядра:

-Модули, выполняющие базовые функции ядра:

* + управление процессами (только код для переключения процессора с процесса на процесс);
  + обработка прерываний (перехват аппаратных прерываний);
  + управление виртуальной памятью (установка регистров блока управления памятью);

- Модуль передачи сообщений, преобразующий вызовы пользовательских модулей ОС в системные вызовы и возвращающий результаты.

* **Достоинства –** расширяемость, переносимость и совместимость
* **Недостаток** – производительность микроядерной операционной системы заметно ниже производительности монолитных операционных систем из – за частого переключения из режима ядра в режим пользователя.(в монолитных 2, микроядерных 4)

**Гибридное ядро(Windows)**

Гибридные ядра — это модифицированные микроядра, позволяющие для ускорения работы запускать «несущественные» части в пространстве ядра.

* Гибридное ядро ОС кроме функций микроядра выполняет следующие дополнительные функции как в монолитных ОС:
  + низкоуровневое управление памятью;
  + взаимодействие между процессами (программами);
  + управление вводом-выводом;
  + управление прерываниями;

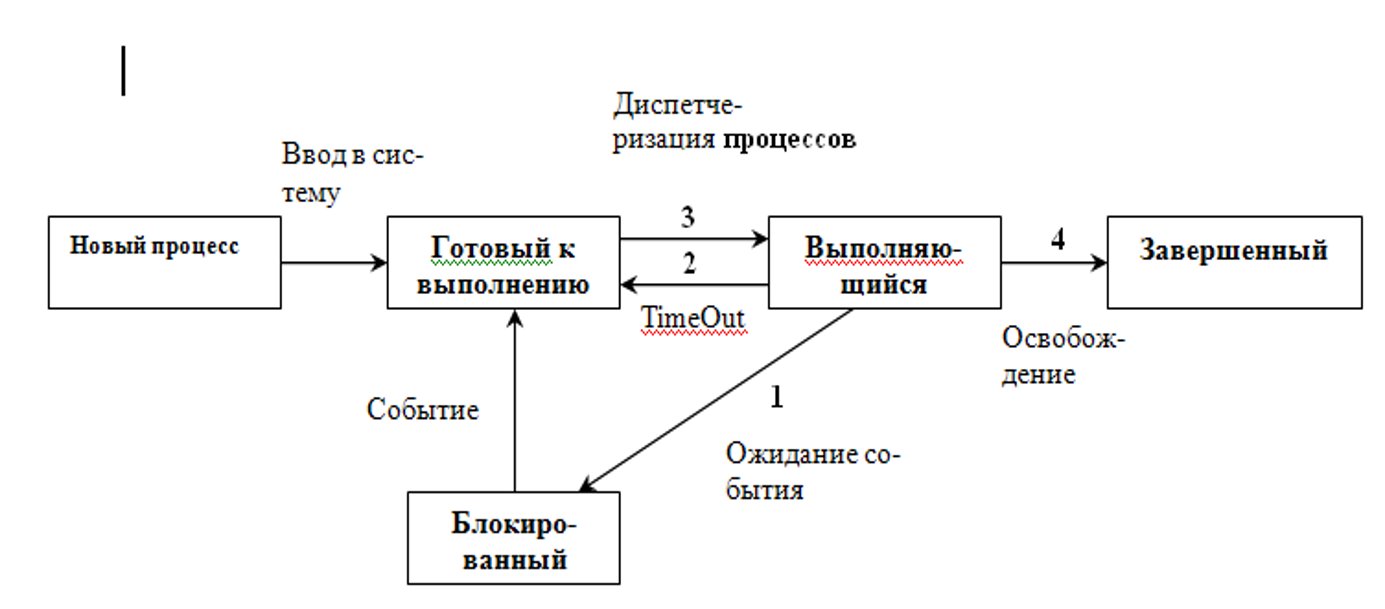
Достоинства: возможно добавлять драйвера устройств двумя способами: и внутрь ядра, и в пользовательское пространство.

Недостатки:

* производительность микроядерной операционной системы заметно ниже производительности многоуровневых и монолитных операционных систем из- за частых переключений из режима ядра в режим пользователя;
* поскольку всё ядро работает в одном адресном пространстве, сбой в одном из компонентов может нарушить работоспособность всей системы;

**16)Понятие процесса. Модель процесса. Переход процесса из одного** **состояния в другое по диаграмме процесса.**

Процесс- система действий, реализующая выполнение программы в компьютерной системе(Процесс- это домик , в котором живет программа)



1. Новый процесс. Только что созданный процесс, информация о процессе помещена ОС в таблицу процессов, но процесс не загружен в оперативную память.
2. Готовый к выполнению. Процесс загружен в память и будет запущен, как только представится возможность.
3. Выполняющийся. Процесс, выполняющийся процессором в данный момент.
4. Блокированный. Процесс, который ожидает некоторого события.
5. Завершающийся. Процесс, удаленный из множества запущенных процессов.

**Диаграмма переходов процесса**

* *Переход из «новый» в «готовый».* ОС осуществляет переход, когда будет готова к обработке дополнительных процессов. В большинстве систем существует ограничения на количество выполняющихся процессов или на объем памяти.
* *Переход из «готовый» в «выполняющийся».* Происходит, когда ОС выбирает новый процесс для его выполнения процессором. Выбор процесса происходит по определенным правилам.
* *Переход из «выполняющийся» в «готовый».* Чаще всего происходит, когда процесс отрабатывает максимальный промежуток времени, отведенный для непрерывной работы одного процесса.
* *Переход из «выполняющийся» в «блокированный».* Процесс переводится в заблокированное состояние, если для продолжения его работы требуется какое-либо событие, необходимое для продолжения работы процесса. (например, медленная операция ввода-вывода).
* Переход «заблокированный»-«готовый». Осуществляется, когда происходит ожидаемое событие.
* *Переход из «выполняющийся» в «завершающийся».* Выполняется тогда, когда процесс сигнализирует об окончании своей работы, или ОС прекращает его выполнение в силу каких-то причин.

**17) Системный, регистровый и пользовательский контексты процесса и размещение процесса в памяти**

* При управлении процессами операционная система создает три информационные структуры :
  + 1) дескриптор процесса (системный контекст) ;
  + 2) регистровый контекст;
  + 3) пользовательский контекст.

**Системный контекст** хранится в области ядра и доступен только ядру, приложение не может самостоятельно напрямую модифицировать его.

В системный контекст входят:

* + время запуска;
  + идентификатор пользователя, создавшего процесс;
  + состояние процесса;
  + расположение процесса в оперативной памяти и на диске
  + приоритет процесса;
  + используемое процессорное время;
  + информация о родственных процессах;
  + параметры планирования;
  + системный стек процесса
  + указатели на открытые процессом файлы;
  + информация об операциях ввода-вывода, используемая процессом.

***Регистровый контекст :***

* + Сохраняется текущее состояние регистров процессора каждый раз, когда ОС прерывает выполнение процесса
  + Извлекается из регистрового контекста обратно в регистры процессора, когда ОС возобновляет выполнение процесса.

Регистровый контекст хранится в области памяти ядра

**Пользовательский контекст:**

* + код программы процесса
  + данные процесса
  + пользовательский стек процесса
  + общая совместная память используемая процессами

Пользовательский контекст процесса хранится в пользовательской области памяти процесса и перемещается при необходимости вместе с ним.

**18)Размещение образа процесса в памяти**

В образ процесса в памяти входит следующая информация:

1. Код программы

2. Данные программы

3. Пользовательский стек.

4. Heap – Куча ( Область памяти куда пользовательская программа может сохранять динамические данные, т.е .данные, которые появляются во время её работы )

5. Общая разделяемая память ( Могут использовать все процессы. Туда могут загружаться например, DLL библиотеки к которым могут обращаться другие процессы.)

6. Блок управления процессом(PCB)

* + Системный контекст
  + Регистровый контекст

**19) Последовательность создания процесса. Системные функции создания процесса в Linux (Unix) и Windows .**

Для создания нового процесса операционной системе необходимо :

•1. Присвоить новому процессу уникальный идентификатор, т.е. занести новую запись в таблицу процессов;

•2. Выделить пространство для процесса, т.е. выделить адресное пространство для всех элементов образа процесса;

• 3. Загрузить часть кодов и данных процесса в оперативную память.

• 4. Инициализировать управляющий блок процесса;

• 5. Поместить процесс в список “готовых” или “готовых приостановленных процессов”;

* В UNIX/Linux существует только один системный вызов для создания нового процесса — *fork*.
* Этот вызов создает точную копию вызывающего процесса.
* После выполнения системного вызова fork два процесса, родительский и дочерний, имеют одинаковые образ памяти, одинаковые строки описания конфигурации и одни и те же открытые файлы.
* По возвращаемому функцией fork() значению процесс может узнать, дочерний он (fork() возвращает 0) или родительский (fork() возвращает число больше 0).
* Для запуска(загрузки ) программы в созданном системным вызовом fork() процессе применяется системный вызов **exec()**. Вызов  **exec()** заменяет пользовательский контекст текущего процесса (образ процесса)на содержимое некоторого исполняемого файла и устанавливает начальные значения регистров процессора.
* **функция wait() -** приостанавливает родительский процесс до тех пор, пока один из его дочерних процессов не будет остановлен или не завершится, после чего возвращает информацию о том, какой процесс завершился и что стало причиной его завершения.
* Если запущено несколько процессов, то для того чтобы дождаться выполнения конкретного процесса используется функция **waitpid().**
* void exit(int status) прекращает процесс, из которого эта функция была вызвана. Кодом возврата этого процесса становиться значение status.
* В Windows системеым вызовом функции ***CreateProces*s** создается процесс, и в него загружается нужная программа.
* У этого вызова имеется 10 параметров(выполняемую программу, параметры командной строки для этой программы, различные параметры безопасности, биты, управляющие наследованием открытых файлов, информацию о приоритетах, спецификацию окна, создаваемого для процесса (если оно используется), и указатель на структуру, в которой вызывающей программе будет возвращена информация о только что созданном процессе.)
* В дополнение к функции CreateProcess в Win32 имеется около 100 других функций для управления процессами.

Особенности процесса в Windows:

Процессы Windows NT имеют свойства:

* + - Процессы Windows NT реализованы в форме объектов - EPROCESS, которые создает менеджер объектов.
    - Объект-процесс, содержит заголовок (PCB), который создает менеджер объектов. Заголовок инициализирует (заполняет ) диспетчер процессов.
    - Менеджер процессов Windows NT не поддерживает между процессами отношений типа «родитель-потомок», т.е родительский процесс не может создать дочерний процесс.
    - Вместо этого в процессе Windows сразу создается начальный поток, в котором выполняется программа процесса.
    - Начальный поток может порождать дочерние потоки

**20) Синхронизация процессов (критические секции, блокирующие переменные).**

**Критические секции**

Та часть программы (процесса), в которой используется доступ к общей памяти (общим переменным), называется критической областью или критической секцией.

Для избежания состязаний необходимо, что бы никакие два процесса не находились одновременно в своих критических секциях.

Для задания режима взаимного исключения используется запрещение прерываний. В этом случае запрещаются прерывания по таймеру. И процессор не может переключится по таймеру на другой процесс (недостаток – запрещение прерываний может привести к краху всей системы).

**Блокирующие переменный**

Когда процессу требуется войти в свою критическую область, сначала он проверяет значение блокирующей переменной.

Если оно равно 0, процесс устанавливает его в 1 и входит в критическую область (т.е. сначала устанавливает, потом входит). Если значение уже равно 1, процесс просто ждет, пока оно не станет равно нулю.

Таким образом, 0 означает, что ни один из процессов не находится в своей критической области, а 1 означает, что какой-то процесс находится в своей критической области

**Семафоры**

Инициализация семафора (задать начальное значение счётчика):

* + Захват семафора (ждать пока счётчик станет больше 0, после этого уменьшить счётчик на единицу);
  + Освобождение семафора (увеличить счётчик на единицу);
  + Бывает бинарный семафор и семафор счетчик.

Семафор — неотрицательная целая переменная, над которой возможны два вида операций:

**P-операция**. Если перед выполнением P-операции значение семафора было больше 0, то P-операция выполняется без задержек. Если перед выполнением P-операции значение семафора было равным 0, то процесс, выполняющий P-операцию, переводится в состояние ожидания до тех пор, пока значение семафора не станет большим 0.

**V-операция**. Если при этом имеются процессы, задержанные на выполнении P-операции на данном семафоре, один из этих процессов выходит из состояния ожидания и может выполнить свою P-операцию.

**21)Основные механизмы межпроцессорного взаимодействия процессов:**

**Общие файлы**

Термином «общие файлы» обычно называют совместный доступ к файлам, преимущественно в контексте операционных систем или служб локальных компьютерных сетей.

**Общая память**

В модели программирования с общей памяти для управления доступом к общей памяти используются всевозможные механизмы синхронизации типа семафоров и блокировок процессов. Преимущество этой модели с точки зрения программирования состоит в том, что не нужно явно задавать обмен данными между производителями и потребителями.

**Разделяемая память**

Техника разделяемой памяти позволяет осуществлять обмен информацией через общий для процессов сегмент памяти без использования системных вызовов ядра. Сегмент разделяемой памяти подключается в свободную часть [виртуального адресного пространства](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D1%80%D0%B5%D1%81%D0%B0%D1%86%D0%B8%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D0%B8#.D0.B2.D0.B8.D1.80.D1.82.D1.83.D0.B0.D0.BB.D1.8C.D0.BD.D0.B0.D1.8F) процесса.

**Сигналы**

-уведомления процесса о каком-либо событии

Когда сигнал послан, процесс прерывается и управление передается функции-обработчику. По окончанию обработки процесс запускается на прерывание

Сигналы, которые можно классифицировать следующим образом:

* Посылаемые в случае завершения выполнения процесса;
* Посылаемые в случае возникновения вызываемых процессом особых ситуаций;
* Посылаемые во время выполнения системной функции при возникновении неисправимых ошибок.

**Каналы**

Каналы – это пары из входного и выходного портов, которые соединяются очередями сообщений.Процессы могут записывать и считывать данные из канала как из обычного файла.

**22) Планирование процессов. Виды планирования процессов(среднесрочное, краткосрочное; вытесняющее, невытесняющее), круговое(циклическое) краткосрочное планирование.**

Для организации многозадачности ОС осуществляет планирование процессов.

• Долгосрочное планирование – когда будет создан новый процесс. Определяет сколько процессов может работать одновременно (степень мультипрограммирования). Такое событие происходит редко (раз в несколько(десятки) минут);

• Среднесрочное планирование определяет какой из процессов будет загружен в память для выполнения вместо выгруженного на диск процесса (переход новый – готовый, миллисекунды);

• Краткосрочное планирование (диспетчер процессов) это планирование какой процесс будет выполняться на процессоре. (переход готовый – выполняющийся, единицы миллисекунд).

**Краткосрочное планирование процессов – задачи:**

• Определение момента времени для смены выполняемого процесса – когда выбирать;

• Выбор процесса на выполнение из очереди готовых процессов кого выбирать;

Алгоритмы планирования делятся на две группы:

– вытесняющие (может прерываться);

– невытесняющие (не может прерываться).

**Вытесняющие алгоритмы планирования**

* Выполняющийся в настоящий момент процесс может быть прерван и переведен операционной системой в состояние готовности к выполнению.
* Планирование возлагается на операционную систему.

**Не вытесняющие алгоритмы планирования**

* При этом способе планирования, процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс или пока не окажется в заблокированном состоянии ожидания завершения операции ввода-вывода.
* Планирование возлагается на программу (программиста) и операционную систему.
  1. Наиболее часто встречающиеся алгоритмы (когда выбирать):
* алгоритмы, основанные на квантовании;
* алгоритмы, основанные на приоритетах.

**Алгоритм квантования (Циклический алгоритм. Круговое планирование)**

Алгоритм квантования:

1.Таймер генерирует прерывания через определенные интервалы времени.

1. При каждом прерывании исполняющийся в настоящий момент процесс помещается в очередь готовых к выполнению процессов, и начинает выполняться очередной процесс.
2. Эта методика известна также как квантование времени, поскольку перед тем как оказаться вытесненным, каждый процесс получает квант времени для выполнения.

**23)Приоритетное планирование.**

* Приоритет может выражаться целыми или дробными, положительным или отрицательным значением. Чем выше привилегии процесса, тем меньше времени он будет проводить в очередях
* Приоритет может назначаться директивно администратором системы в зависимости от важности работы или внесенной платы, либо вычисляться самой ОС по определенным правилам,
* Приоритет может оставаться фиксированным на протяжении всей жизни процесса либо изменяться во времени в соответствии с некоторым законом. В последнем случае приоритеты называются динамическими.
* Приоритет - это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной машины: чем выше приоритет, тем выше привилегии.

**Алгоритмы на основе приоритетов**

**1. Первым поступил − первым обслужен**

* простейшая стратегия планирования "первым поступил — первым обслужен" или схема строгой очередности;
* как только процесс становится готовым к выполнению, он присоединяется к очереди готовых процессов. При прекращении выполнения текущего процесса для выполнения выбирается процесс, который находился в очереди дольше других.

1. **Выбор самого короткого процесса**

* для выполнения выбирается процесс с наименьшим ожидаемым временем исполнения;
* основная трудность в применении этого способа состоит в оценку времени выполнения, требующегося каждому процессу.

**24)Переключение процессов.**

Чтобы управлять выполнением процессов, ядро должно уметь приостанавливать (прерывать) процесс, работающий в данный момент, и возобновлять выполнение другого, ранее приостановленного процесса. Это называется переключение процессов (потоков).

Пример механизма переключений:

1. По истечению кванта времени выполнения процесса процессором возникает прерывание от таймера, по которому выполнение процесса приостанавливается, содержимое регистров процессора (счетчик команд, слово состояния и др.) сохраняется в регистровом контексте процесса в ядре.

2. Планировщик процессов выбирает нужный, ранее прерванный процесс (согласно алгоритма планирования). В регистры процессора из регистрового контекста данного процесса загружается сохраненное ранее содержимое регистров процессора.

3. Выполнение процесса возобновляется с команды программы следующей после последней выполненной процессором в предыдущем цикле работы программы.