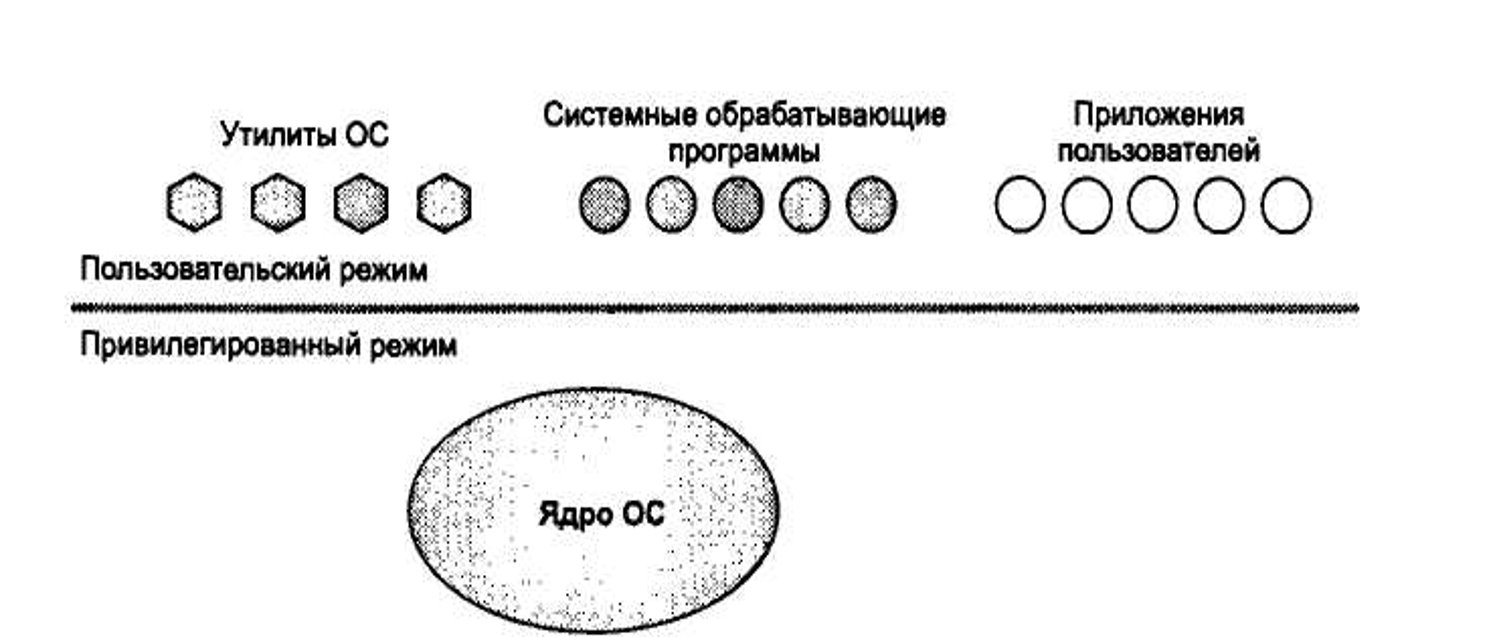
13) **Режим пользователя и режим ядра. Ядро ОС и вспомогательные функции(примеры).**

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 :

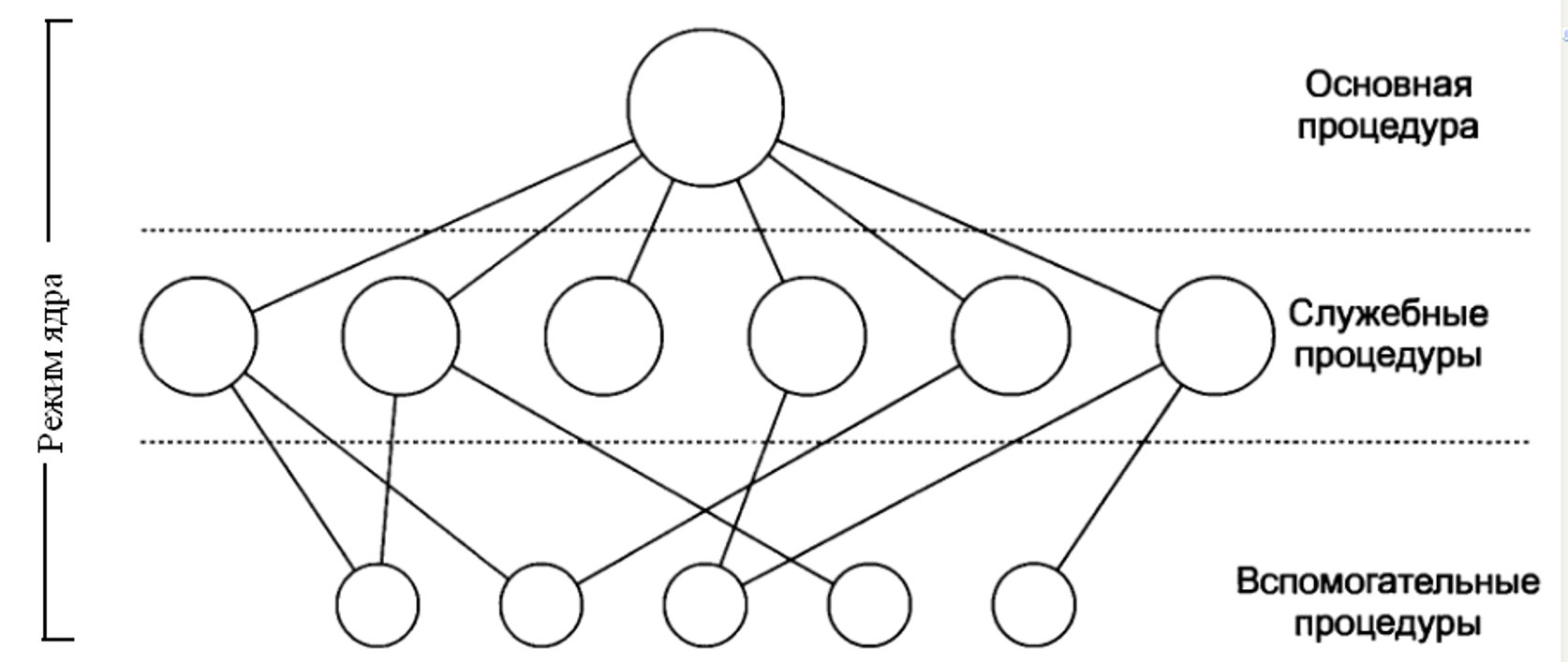
**1 класс.** Функции для управления вычислительным процессом (*эти функции недоступны для приложений)* :

* 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

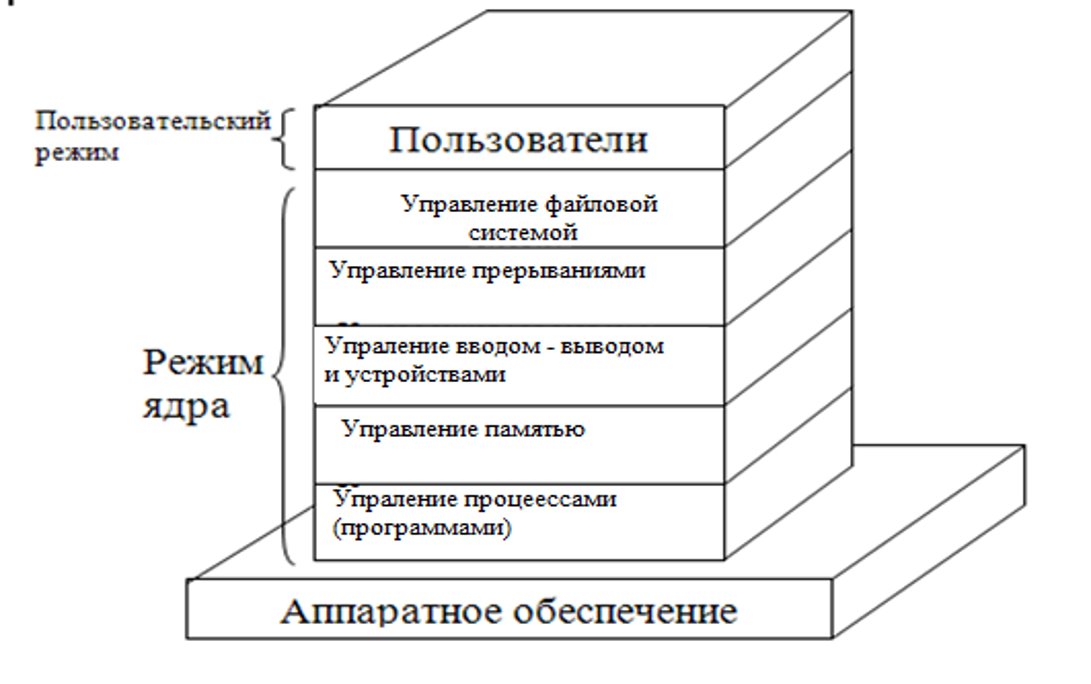
Ограничения:

-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.

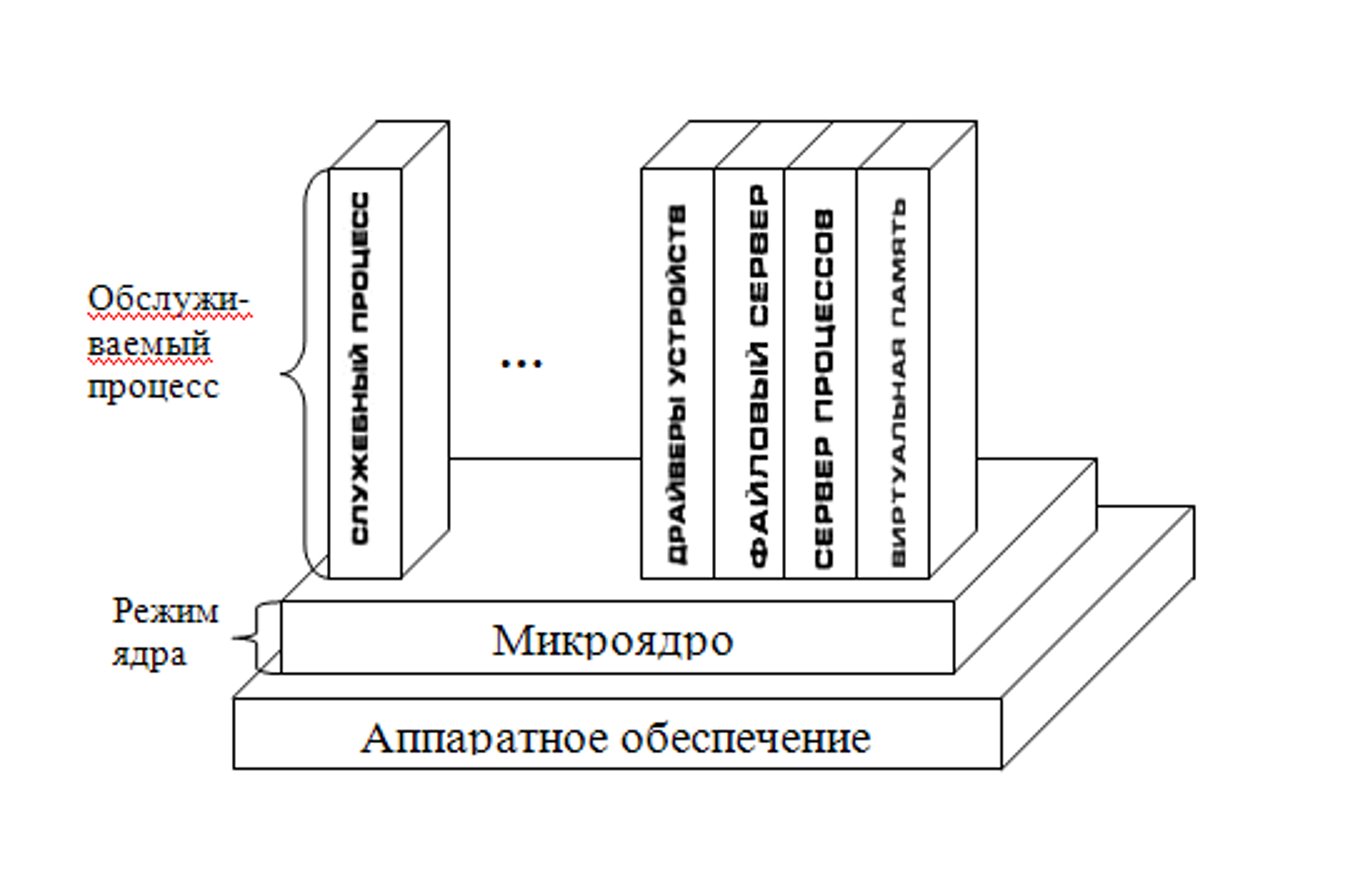
-Переносимость. При переносе операционной системы на другую аппаратную платформу, аппаратно зависимый уровень может быть просто заменен.

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;

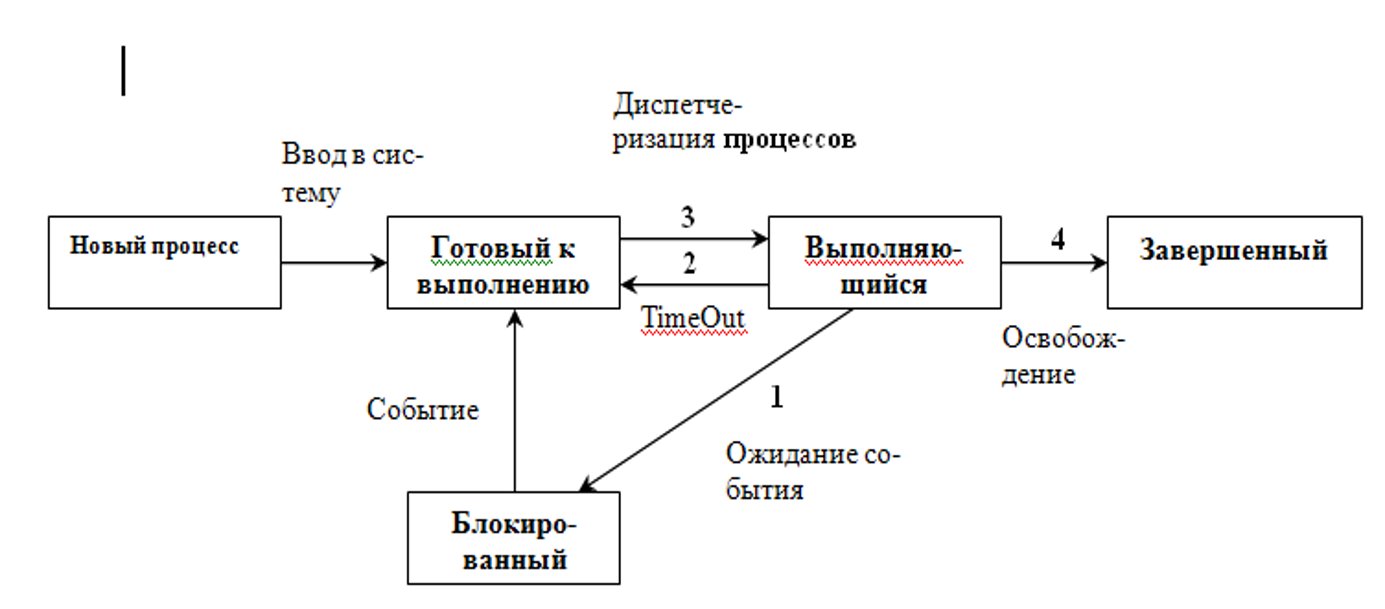
Достоинства: возможно добавлять драйвера устройств двумя способами: и внутрь ядра, и в пользовательское пространство.

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