1. **Назначение и функции ОС. Принципы построения ОС.**

ОС – комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем; ОС обеспечивает совместное функционирование всех устройств компьютера и предоставляет пользователю доступ к его ресурсам.

Функции ОС:

* Обеспечение интерфейса пользователя
* Управление вычислительными процессами
* Управление памятью
* Управление хранением данных
* Управление вводом и выводом информации
* Сервисное обслуживание

Доп. Функции ОС:

* Защита системы
* Эффективное распределение ресурсов вычислительной системы между процессами
* Параллельное или псевдопараллельное выполнение задач (многопоточность)

Принципы построения:

* Принцип модульности (Модульная организация позволяет легко (из)заменять неправильно работающие модули в ОС)
* Принцип функциональной избирательности (наиболее важные модули должны храниться в ОП(резидентное ядро))
* Принцип генерируемости (Подразумевает собой возможность генерации ОС в зависимости от аппаратного обеспечения)
* Принцип функциональной избыточности(В состав ОС должно входит несколько типов ПО для выполнения одинаковых функций
* Принцип виртуализации (структура системы представляется в виде набора планировщиков процессов; ОС скрывает от пользователя реальные механизмы управления, заменяя их абстрактным представлением)
* Принцип перемещаемости (модули ОС не должны зависеть от их расположения в памяти)
* Принцип защиты (защита данных от несанкц. Доступа, защита процессов друг от друга)
* Принцип открытой и наращиваемой ОС (наращиваемая ОС позволяет вводить в состав новые модули; открытая ОС доступна для анализа)
* Принцип мобильности (ОС должна переноситься с одной аппаратной платформы на другую)
* Принцип независимости программ от устройств (связь программы с устройствами производится на уровне исполнения)
* Принцип совместимости (способность выполнять программы, написанные для более ранних версий данной ОС и для других ОС)

1. **Понятие и состояние процесса. Свойства процесса**

Процесс – деятельность, связанная с исполнением программы на процессоре.

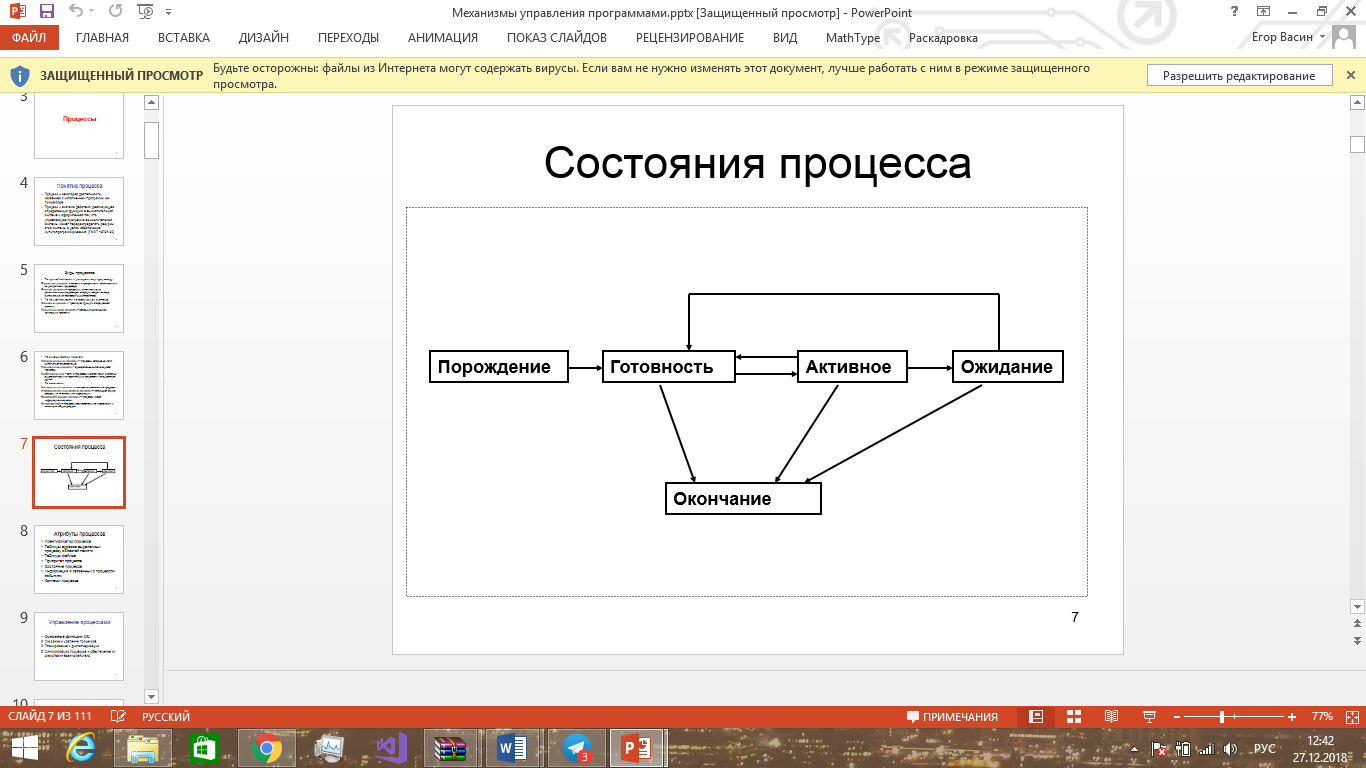
Процесс – система действий, реализующая как-либо функцию в выч. системе

По принадлежности к центральному процессору:

* Внутренние процессы (связаны с программами, исполняемыми на ЦП)
* Внешние процессы (связаны с программами, выполняемыми на переф. Устройствах)

По принадлежности к ОС:

* Системные процессы – реализуют функцию ОС
* Пользовательские процессы – связаны с выполнением прикладных программ



Также процессы можно классифицировать по динамическому признаку и по связности.

Атрибуты процессов:

* Идентификатор процесса
* Таблицы файлов
* Приоритет процесса
* Состояние процесса
* Контекст процесса

1. **Принципы управлениями процессами. Долгосрочное и краткосрочное планирование процессов.**

Управление процессами включает в себя:

* Создание и удаление процессов
* Планирование и диспетчеризация
* Синхронизация процессов

Планирование – управление очередями с целью минимизации задержек и максимизации производительности системы. Выделяют: краткосрочное, среднесрочное, долгосрочное планир-е.

**Краткосрочный** – решает, какому потоку дать следующий квант процессорного времени и какой длины.

**Среднесрочное** – решает нужно ли временно выгружать программу во вторичную память (какую и вообще нужно ли это).

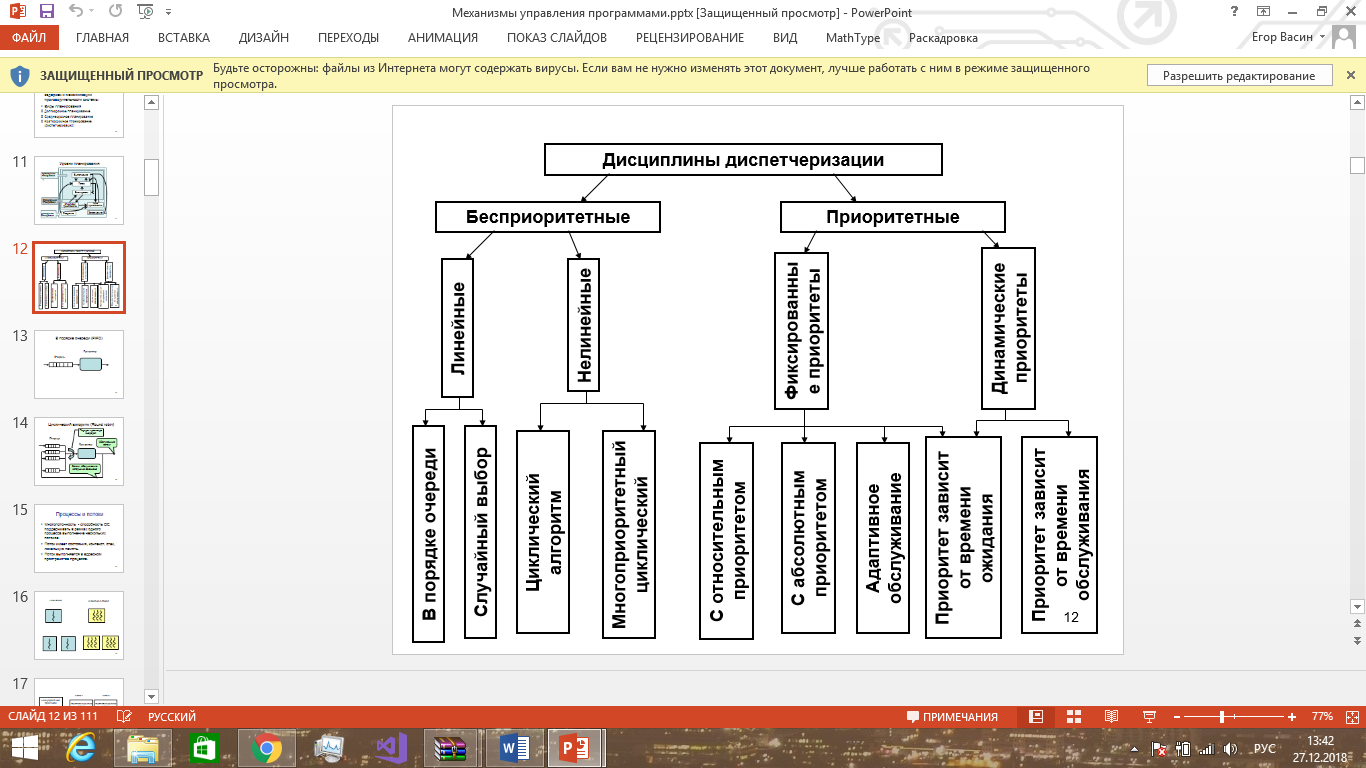
**Долгосрочное** – решает какие новые задачи будут добавлены (концептуальные вопросы).

1. **Дисциплины формирования очередей и дисциплины обслуживания. Приоритеты.**

Дисциплины формирования очередей выглядят следующим образом:



Дисциплина обслуживания очередей:





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

**5. Дескрипторы процессов. Прерывания и переключения контекстов.**

При управлении процессами операционная система использует два основных типа информационных структур: **дескриптор процесса** (структура proc) и **контекст процесса** (структура user).

**Дескриптор процесса** содержит такую информацию о процессе, которая необходима ядру в течение всего жизненного цикла процесса, независимо от того, находится ли он в активном или пассивном состоянии, находится ли образ процесса в оперативной памяти или выгружен на диск. Дескрипторы отдельных процессов объединены в список, образующий таблицу процессов. Память для таблицы процессов отводится динамически в области ядра. На основании информации, содержащейся в таблице процессов, операционная система осуществляет планирование и синхронизацию процессов. В дескрипторе прямо или косвенно (через указатели на связанные с ним структуры) содержится информация о состоянии процесса, расположении образа процесса в оперативной памяти и на диске, о значении отдельных составляющих приоритета, а также его итоговое значение - глобальный приоритет, идентификатор пользователя, создавшего процесс, информация о родственных процессах, о событиях, осуществления которых ожидает данный процесс и некоторая другая информация.



Контекст процесса содержит менее оперативную, но более объемную часть информации о процессе, необходимую для возобновления выполнения процесса с прерванного места: содержимое регистров процессора, коды ошибок выполняемых процессором системных вызовов, информацию о всех открытых данным процессом файлов и незавершенных операциях ввода-вывода (указатели на структуры file) и другие данные, характеризующие состояние вычислительной среды в момент прерывания. Контекст, так же как и дескриптор процесса, доступен только программам ядра, то есть находится в виртуальном адресном пространстве операционной системы, однако он хранится не в области ядра, а непосредственно примыкает к образу процесса и перемещается вместе с ним, если это необходимо, из оперативной памяти на диск.

Процедуру обработки прерываний, в зависимости от их типа, выполняют входящие в состав ОС соответствующие программы — обработчики прерываний (IH — interrupt handler).  
При возникновении прерывания, ОС должна обеспечивать запоминание состояния прерванного процесса и передачу управления соответствующему обработчику прерывания. Это может быть реализовано с использованием способа переключение контекста(context switching).  
При реализации этого способа используются слова состояния программы PSW(program status word), которые управляют порядком выполнения команд и содержат ряд сведений о состоянии процесса. Существует три типа PSW:  
текущее PSW, новое PSW старое PSW.  
В однопроцессорной машине существует одно текущее PSW, которое содержит адрес следующей команды, подлежащей выполнению, а также типы прерываний, разрешенных и запрещенных на данный момент. Шесть новых PSW, которые содержат адреса размещения соответствующих обработчиков прерываний, и шесть старых PSW — по одному для каждого типа прерываний.  
Когда происходит прерывание, то в первую очередь проверяется не является ли оно запрещенным (тогда оно либо задерживается либо игнорируется). Если прерывание является разрешенным, то аппаратурно производится следующее переключение PSW:  
• текущее PSW становится старым PSW для прерывания этого типа;  
• новое PSW для прерывания этого типа становится текущим PSW.  
После такого замещения слов состояния, текущее PSW содержит адрес соответствующего обработчика прерываний, который начинает обрабатывать данное прерывание. Когда обработка прерывания завершается, ЦП начинает обслуживать либо тот процесс, который выполнялся в момент прерывания, либо готовый процесс с наивысшим приоритетом.

**6. Типовые задачи синхронизации параллельных процессов. Тупики.**

1. Задача взаимного исключения

При работе нескольких параллельных процессов с общими данными возникает необходимость взаимоисключать одновременный доступ процессов к данным. При этом участки программ процессов для работы с разделяемыми данными образуют так называемые критические области (секции). В общем виде постановка задачи взаимного исключения формулируется следующим образом: необходимо согласовать работу n 2 параллельных процессов при использовании некоторого критического ресурса таким образом, чтобы удовлетворить следующим требованиям:

– одновременно внутри критической области должно находиться не более одного процесса;

– критические области не должны иметь приоритета по отношению друг к другу;

– остановка какого-либо процесса вне его критической области не должна влиять на дальнейшую работу процессов по использованию критического ресурса;

– решение о вхождении процессов в их критические области при одинаковом времени поступления запросов на такое вхождение и равноприоритетности процессов не откладывается на неопределенное время, а является конечным по времени;

– относительные скорости развития процессов неизвестны и произвольны;

– любой процесс может переходить в любое состояние, отличное от активного, вне пределов своей критической области;

– освобождение критического ресурса и выход из критической области должны быть произведены процессом, использующим критический ресурс, за конечное время.

2. Задача «производитель-потребитель»

Рассмотрим систему из двух параллельных процессов, включающую процесс производитель и процесс-потребитель. Процессы взаимодействуют через некоторую обобщенную область памяти – буфер сообщений. В эту область процесс-производитель помещает очередное сообщение, а процесс-потребитель считывает очередное сообщение. В общем случае буфер способен хранить несколько сообщений. Необходимо согласовать работу двух процессов при одностороннем обмене сообщениями по мере развития процессов таким образом, чтобы удовлетворить следующим требованиям:

– выполнять требования задачи взаимного исключения по отношению к критическому ресурсу

– учитывать состояние буфера сообщений, характеризующего возможность или невозможность посылки (принятия) очередного сообщения.

Процесс-производитель при попытке поместить очередное сообщение в полностью заполненный буфер должен быть блокирован. Попытка процесса-потребителя чтения из пустого буфера также должна быть блокирована.

3. Задача «читатели-писатели»

Процессы-читатели считывают, а процессы-писатели записывают информацию в общую область памяти. Одновременно может быть несколько активных процессов-читателей. При записи информации область памяти рассматривается как критический ресурс для всех процессов, т. е. если работает процесс-писатель, то он должен быть единственным активным процессом. Задача состоит в определении структуры управления, которая не приведет к тупику и не допустит нарушения критерия взаимного исключения.

Тупики:

Тупики и близкая к ним проблема бесконечного откладывания — важные факторы, которые должны учитывать разработчики ОС.  
Тупик (deadlock) — это такая ситуация в мультипрограммной системе, когда процесс ожидает некоторого события, которое никогда не произойдет. Системная тупиковая ситуация, или “зависание” системы — это ситуация, когда один или более процессов оказываются в состоянии тупика.

Правила предотвращения тупиков в ОС:

1. Прежде чем процесс начнет свою работу, ему должны быть предоставлены все требуемые ресурсы.
2. В том случае, если во время работы ему понадобился дополнительный ресурс, ему необходимо возвратить все ранее выделенные ресурсы ОС и затем запросить все требуемые ресурсы с этим дополнительным ресурсом.

Бесконечное откладывание процесса.   
В системе, где процессам приходится ждать пока она выделит ему требуемый ресурс может возникнуть ситуация, что будут приходить процессы с более высоким приоритетом, требующие тот же самый ресурс - **ситуация бесконечного откладывания процесса**.   
В некоторых ОС данная ситуация предотвращается благодаря увеличению приоритетности (**"старению"** процесса) для того, чтобы ему был предоставлен требуемый ресурс, после чего приоритет понижается до прежнего уровня

**7. Семафоры**

Семафоры традиционно использовались для синхронизации процессов, обращающихся к разделяемым данным. Каждый процесс должен исключать для всех других процессов возможность одновременно с ним обращаться к этим данным (взаимоисключение). Когда процесс обращается к разделяемым данным, говорят, что он находится в своем критическом участке.

Для решения задачи синхронизации необходимо, в случае если один процесс находится в критическом участке, исключить возможность вхождения для других процессов в их критические участки. Хотя бы для тех, которые обращаются к тем же самым разделяемым данным. Когда процесс выходит из своего критического участка, то одному из остальных процессов, ожидающих входа в свои критические участки, должно быть разрешено продолжить работу.

Процессы должны как можно быстрее проходить свои критические участки и не должны в этот период блокироваться. Если процесс, находящийся в своем критическом участке, завершается (возможно, аварийно), то необходимо, чтобы некоторый другой процесс мог отменить режим взаимоисключения, предоставляя другим процессам возможность продолжить выполнение и войти в свои критические участки.

Семафор - это защищенная переменная, значение которой можно опрашивать и менять только при помощи специальных операций *wait* и *signal* и операции инициализации *init*. Двоичные семафоры могут принимать только значения 0 и 1. Семафоры со счетчиками могут принимать неотрицательные целые значения.

Операция *wait(s)* над семафором *s* состоит в следующем:

если *s > 0* то *s:=s-1* иначе (ожидать на *s*)

а операция *signal(s)* заключается в том, что:

если (имеются процессы, которые ожидают на *s*)

то (разрешить одному из них продолжить работу)

иначе *s:=s+1*

Операции являются неделимыми. Критические участки процессов обрамляются операциями *wait(s)* и *signal(s)*. Если одновременно несколько процессов попытаются выполнить операцию *wait(s)*, то это будет разрешено только одному из них, а остальным придется ждать.

Семафоры со счетчиками используются, если некоторые ресурс выделяется из множества идентичных ресурсов. При инициализации такого семафора в его счетчике указывается число элементов множества. Каждая операция *wait(s)* уменьшает значения счетчика семафора *s* на 1, показывая, что некоторому процессу выделен один ресурс из множества. Каждая операция *signal(s)* увеличивает значение счетчика на 1, показывая, что процесс возвратил ресурс во множество. Если операция *wait(s)* выполняется, когда в счетчике содержится нуль (больше нет ресурсов), то соответствующий процесс ожидает, пока во множество не будет возвращен освободившийся ресурс, то есть пока не будет выполнена операция *signal*.

**8. Общие сведения об OC UNIX**

Операционная система UNIX - это набор программ, который управляет компьютером, осуществляет связь между вами и компьютером и обеспечивает вас инструментальными средствами, чтобы помочь вам выполнить вашу работу. Разработанная, чтобы обеспечить легкость, эффективность и гибкость программного обеспечения, система UNIX имеет несколько полезных функций:

* основная цель системы - это выполнять широкий спектр заданий и программ;
* интерактивное окружение, которое позволяет вам связываться напрямую с компьютером и получать немедленно ответы на ваши запросы и сообщения;
* многопользовательское окружение, которое позволяет вам разделять ресурсы компьютера с другими пользователями без уменьшения производительности. Этот метод называется разделением времени. Система UNIX взаимодействует с пользователями поочередно, но так быстро, что кажется, что взаимодействует со всеми пользователями одновременно;
* многозадачное окружение, которое позволяет вам выполнять более одного задания в одно и тоже время.

Система UNIX имеет 4 основных компонента:

ядро -

это программа, которая образует ядро операционной системы; она координирует внутренние функции компьютера ( такие как размещение системных ресурсов). Ядро работает невидимо для вас;

shell -

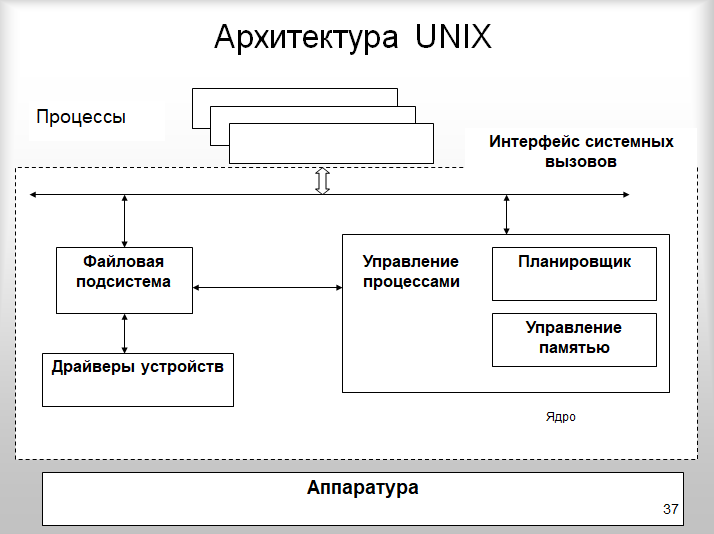
это программа, которая осуществляет связь между вами и ядром, интерпретируя и выполняя ваши команды. Так как она читает ваш ввод и посылает вам сообщения, то описывается как интерактивная;

commands -

это имена программ, которые компьютер должен выполнить. Пакеты программ называются инструментальными средствами. Система UNIX обеспечивает инструментальными средствами для таких заданий как создание и изменение текста, написание программ, развитие инструментария программного обеспечения, обмен информацией с другими посредством компьютера;

file system -

файловая система - это набор всех файлов, возможных для вашего компьютера. Она помогает вам легко сохранять и отыскивать информацию.



Ядро контролирует доступ к компьютеру, управляет памятью компьютера, обслуживает файловую систему и распределяет ресурсы компьютера среди пользователей. На рис. 1 приведено функциональное представление ядра.

Ядро

\ Распределение /

\ ресурсов /

\ системы /

\ /

Управление \ / Обслуживание

памятью \/ файловой

/ \ системы

/ \

/ Контроль \

/ доступа \

/ к компьютеру \

shell - это программа, которая позволяет вам связываться с операционной системой. shell считывает команды, которые вы вводите, и интерпретирует их как запросы на выполнение других программ, на доступ к файлу или обеспечение вывода. shell также является мощным языком программирования, не похожим на язык программировани Си, который опеспечивает условное выполнение и управление потоками данных.

**9.Процессы в ОС UNIX. Типы процессов. Атрибуты процессов.**

С процессом связана какая-либо программа, для которой выделяется место в памяти, или системные переключения между процессами. Процессы в ОС UNIX играют ключевую роль. От оптимальной настройки подсистемы управления процессами и числа одновременно выполняемых процессов зависит загрузка процессора, что оказывает непосредственное влияние на производительность системы в целом.

процесс- это набор команд или инструкций, выполняемым процессором,совместно с данными и информацией о выполняемой задаче,такой как распределение памяти, открытие файла и статус процесса.

Процесс не отождествляется с программой, т.к. программа может создать(породить) более одного процесса. Выполнение процесса заключается в точном следовании набору инструкций, при котором никогда не передаётся управление другому процессу.

Процессу также недоступны данные и стеки других процессов. Но процессы могут обмениваться друг с другом данными с помощью системы межпроцессового взаимодействия, предоставляемой ОС UNIX. К этой системе относятся:

1. сигналы (похожи напрерывания);

2. каналы (двунаправленная буферизация файла);

3. файлы;

4. разделяемая память;

5. семафоры;

6. сообщения.

В остальном процессы изолированы.

Типы процессов

Системные процессы

Системные процессы являются частью ядра ОС и всегда находятся в оперативной памяти.

Они не имеют соответствующих им программ в виде исполняемых файлов и запускаются особым образом при инициализации ядра системы, т.о. они могут вызывать функции и обращение к данным, недоступных для остальных процессов.

Системными процессами являются:

shed ¾ диспетчер свопинга

vhand ¾ диспетчер страничного замещения

bdfflush ¾ диспетчер буферного кэша

kmadaemon ¾ диспетчер памяти ядра

К системным также можно отнести процесс init (хотя он не является частью ядра и его запуск производится из специального файла в /etc) являющийся прародителем всех остальных процессов.

Демоны

Демоны ¾ это не интерактивные процессы, которые запускаются обычным образом путем загрузки в память соответствующей им программы и выполнения в фоновом режиме. Обычно демоны запускаются при инициализации системы, но после инициализации ядра, и обеспечивают работу различных подсистем (терминального доступа, печати, сетевого доступа и т.д.)

Демоны не связаны ни с одним пользовательским сеансом и не могут управляться пользователем непосредственно. Большую часть времени демоны ожидают, пока той или иной процесс не запросит определённую услугу.

Прикладные процессы

К ним относятся все остальные процессы, выполняющиеся в системе. Обычно, это процессы, порожденные в рамках пользовательского сеанса работы.

Важнейший пользовательский процесс ¾ это командный интерпретатор (shell), который запускается после регистрации пользователя в системе, а его завершение приводит к отключению пользователя от системы.

Пользовательские процессы могут выполняться как в интерактивном, так и в фоновом режиме, но время их жизни ограничено временем работы пользователя. При завершении сеанса пользователя все пользовательские процессы уничтожаются.

Интерактивные процессы монопольно владеют терминалом. И пока такой процесс не завершит работу, пользователь не может работать с другими процессами, если в функции интерактивного процесса не входит запуск на выполнение других программ (процесс shell-интерактивный, но пользователь может запускать другие процессы).

Атрибуты процесса

Атрибут обеспечивает возможность ОС по управлению работойпроцесса. К ним относятся:

\* Идентификатор процесса PID (Process ID). Каждый процесс имеет уникальный идентификатор, позволяющий ядру системы различать процессы. Когда создается новый процесс, ядро присваивает ему следующий свободный идентификатор. Присваивание идентификаторов происходит по возрастанию, т.е. идентификатор следующего процесса больше идентификатора процесса, созданного перед ним. Если идентификатор достиг максимально возможного значения, то следующий процесс получит минимальный свободный идентификатор и цикл выделяет индетификатор повторно. Когда процесс завершит свою работу, ядро освобождает занятый им идентификатор.

\* Идентификатор родительского процесса PPID(Parent Process ID) ¾ идентификатор процесса, породившего данный процесс.

\* Приоритет процесса Vice Number - относительный приоритет процесса, учитываемый планировщиком процессов при очерёдности запуска. может быть изменён пользователем или администратором,в отличие от приоритета выполнения,динамически обновляемого ядром.

\* Терминальная линия TTY ¾ это терминал (псевдотерминал), ассоциированный с процессом,если таковой существует. Демоны не имеют ассоциированного терминала.

\* Реальный (RID) и эффективный (EUID) идентификаторы пользователя. Реальный идентификатор ¾ это идентификатор пользователя, запустившего процесс, а эффективный идентификатор используется для определения прав доступа процесса к системным ресурсам, в первую очередьк рессурсам файловой системы. В большинстве случаев, реальный и эффективный идентификаторы совпадают, однако есть возможность задать процессу более широкие права, путём установки специального флага SUID.

\* Реальный (RGID) и эффективный (EGID) идентификаторы группы.

**10. Жизненный цикл процесса в ОС UNIX.**

1. Процесс только что создан системным вызовом fork и находится в переходном состоянии: он существует, но не готов к запуску и не находиться в состоянии сна.

2. Процесс не выполняется, но готов к запуску, как только планировщик выберет его (состояние runnable). Процесс находиться в очереди на выполнение и обладает всеми необходимыми ему ресурсами, кроме процессора.

3. Процесс выполняется в пользовательском режиме. При этом процессором выполняются прикладные инструкции данного процесса.

4. Процесс находиться в состоянии сна (asleep), ожидая недоступного в данный момент ресурса, например завершения операции ввода-вывода.

5. Процесс выполняется в режиме ядра. При этом процессом выполняются системные инструкции ядра от имени процесса.

6. Процесс возвращается из режима ядра в режим задачи, но ядро прерывает его и производит переключение контекста для запуска более приоритетного процесса.

7. Процесс выполнил системный вызов exit и перешел в состояние зомби (zombie, defunct). Как такового процесса не существует, но остаются записи, содержащие код возврата и временную статистику его выполнения, доступную для родительского процесса. Это состояние является конечным в жизненном цикле процесса.

**11. Порождение и завершение процессов в ОС UNIX. Запуск новой программы.**

Порождение

Новый процесс порождается системным вызовом fork, который создает дочерний процесс - копию родительского. В дочернем процессе выполняется та же программа, что и в родительском, и когда дочерний процесс начинает выполняться, он выполняется с точки возврата из системного вызова fork. Системный вызов forkвозвращает родительскому процессу PID дочернего процесса, а дочернему процессу - 0. По коду возврата

вызова fork дочерний процесс может "осознать" себя как дочерний. Свой PID процесс может получить при помощи системного вызова getpid, а PID родительского процесса - при помощи системного вызоваgetppid. Если требуется, чтобы в дочернем процессе выполнялась программа, отличная от программы родительского процесса, процесс может сменить выполняемую в нем программу при помощи одного из системных вызовов семейства exec. Все вызовы этого семейства загружают для выполнения в процессе программу из заданного в вызове файла и отличаются друг от друга способом передачи параметров этой программе.

Завершение

Процесс завершает работу при выполнении системного вызова exit. Процесс может сам завершить свою работу, в соответствии с алгоритмом, либо может быть прекращен ядром.

При завершении процесса последовательно выполняются следующие действия:

1. Отключаются все сигналы.

2. В вызвавшем процессе закрываются все дескрипторы открытых файлов.

3. Если родительский процесс находится в состоянии вызова wait, то системный вызов wait завершается, выдавая родительскому процессу в качестве результата идентификатор завершившегося процесса, и младшие 8 бит его кода завершения.

4. Если родительский процесс не находится в состоянии вызова wait, то завершающийся процесс переходит в состояние зомби.

У всех существующих потомков завершенных процессов, а также у зомби-процессов идентификатор родительского процесса устанавливается равным 1. Таким образом, они становятся потомками процесса инициализации (init).

Если идентификатор процесса, терминальная линия и идентификатор группы процессов у завершающегося процесса совпадают, то всем процессам с тем же идентификатором группы процессов посылается сигнал SIGHUP. Тем самым, завершаются и все порожденные в приоритетном режиме процессы.

Родительскому процессу посылается сигнал SIGCHLD (завершение порожденного процесса). Этот сигнал пробуждает родительский процесс, если тот ожидает завершения порожденных процессов.

Запуск новой программы

/\* порождение дочернего процесса и запоминание его PID \*/

if (!(ch\_pid=fork())

/\* загрузка другой программы в дочернем процессе \*/

exec(программа);

else

продолжение родительского процесса

**12. Планирование выполнения процессов в ОС UNIX. Таймеры. Алармы.**

Определение последовательности выполнения потоков предусматривает два алгоритма, которые работают независимо друг от друга:

1. Низкоуровневый алгоритм – выбирает следующий процесс из набора процессов в памяти, готовых к работе.

2. Высокоуровневый алгоритм (swapper) – перемещает процессы из памяти на диск и обратно, т.е. осуществляет свопинг.

В низкоуровневом алгоритме используется несколько очередей. С каждой очередью связан диапазон непересекающихся значений приоритетов.

\* Процессы, выполняющиеся в режиме «процесс» обычно имеют положительное значение приоритетов.

\* У процессов, выполняющихся в режиме система, значение приоритетов, как правило, отрицательное. Отрицательное значение приоритетов считается наивысшими.

Когда запускается низкоуровневый алгоритм, он ищет очередь, начиная с самого высокого приоритета, т.е. наименьшего отрицательного значения, пока не находит очередь, в которой есть хотя бы один процесс. После этого в этой очереди выбирается и запускается первый процесс. Если процесс использует весь свой квант времени (~100мкс), он помещается в конец очереди, а алгоритм планирования запускается снова.

Один раз в секунду приоритет каждого процесса пересчитывается следующим образом:

PRI=K\*CPU+NICE+BASE

На основе нового приоритета каждый процесс прикрепляется к соответствующей очереди.

\* CPU – Среднее значение тиков таймера, в течение которого процесс выполнялся. При каждом тике таймера счетчик использования ЦП процессом в таблице процессов увеличивается на 1.

\* K – Коэффициент для конкретной UNIX системы. Иногда это число K, иногда функция K(t).

\* NICE - Пользовательская составляющая приоритета процесса, принимает диапазон значений от -20 до +20. С помощью системного вызова nice можно менять приоритет процесса от 0 до 20. Системный администратор может запросить обслуживание с более высоким приоритетом от -20 до 20. Это однозначно увеличивает частоту эксплуатации ЦП этим процессом.

\* BASE

Обработчик прерываний ядра вызывается аппаратным прерыванием таймера, приоритет которого самый высокий. Обраб прерывания должна заним минимальное количество времени. Обработчики:

1. Обновление статистики использования CPU для текущ процесса.

2. Планировщик процессов (пересчет приоритета , проверка истчения временного кванта для процесса) Если превышена процессорная квота, то посыл сигнал SIGEXCPU

3. Обновление системного времени, обработка отложенных вызовов, обработка будильников.

Часть задач раегирует только на главный тик (кажд 4 тика). Отложенный вызов – определяет функцию, вызов котрой будет произведен системой через некот время.

Будильники:

Процесс может попросить ядро отправить сигнал SIGALRM себе через некотрое время.

Для установки таймера реального времени:

Int alarm( unsigned int seconds)

- Вернет 0 если не было заказа.

- Если был, то вернет количество секунд до посылки сигнала. Новый заказ отменит старый и установит свой.

1. **Организация взаимодействия процессов в ОС UNIX. Сигналы.**

Взаимодействие процессов в ОС UNIX (IPC).

* + 1. В Сигналы:
       1. Терминальные прерывания (Del, Ctrl+C…)
       2. Другие процессы (вызов kill)
       3. Управление заданиями
       4. Квоты
       5. Уведомления
       6. Алармы

Доставка и обработка сигнала

* Ядро от имени процесса проверяет наличие сигнала
* Если сигнал есть, ядро обрабатывает его по умолчанию, либо запускает специальную функцию, которая вызывает специальную функцию обработки сигнала.
  + 1. Каналы (pipe)
    2. Именованные каналы (FIFO)
    3. Сообщения (messages)
    4. Семафоры
    5. Разделяемая память
    6. Сокеты (socket)

1. **Каналы и именованные каналы в ОС UNIX.**

Различаются два вида программных каналов – именованные(FIFO) и неименованные(pipe). Именованный программный канал может служить для общения и синхронизации произвольных процессов, знающих имя данного программного канала и имеющих соответствующие права доступа. Неименованным программным каналом могут пользоваться только создавший его процесс и его потомки (необязательно прямые). Для создания именованного программного канала (или получения к нему доступа) используется обычный файловый системный вызов open. Для создания же неименованного программного канала существует специальный системный вызов pipe. Однако после получения соответствующих дескрипторов оба вида программных каналов используются единообразно с помощью стандартных файловых системных вызовов read, write и close.

1. **Сообщения в ОС UNIX.**

Очереди сообщений ( message queues) - средства, обеспечивающие возможность посылки процессом сообщений другому произвольному процессу

Для обеспечения возможности обмена сообщениями между процессами этот механизм поддерживается следующими системными вызовами:

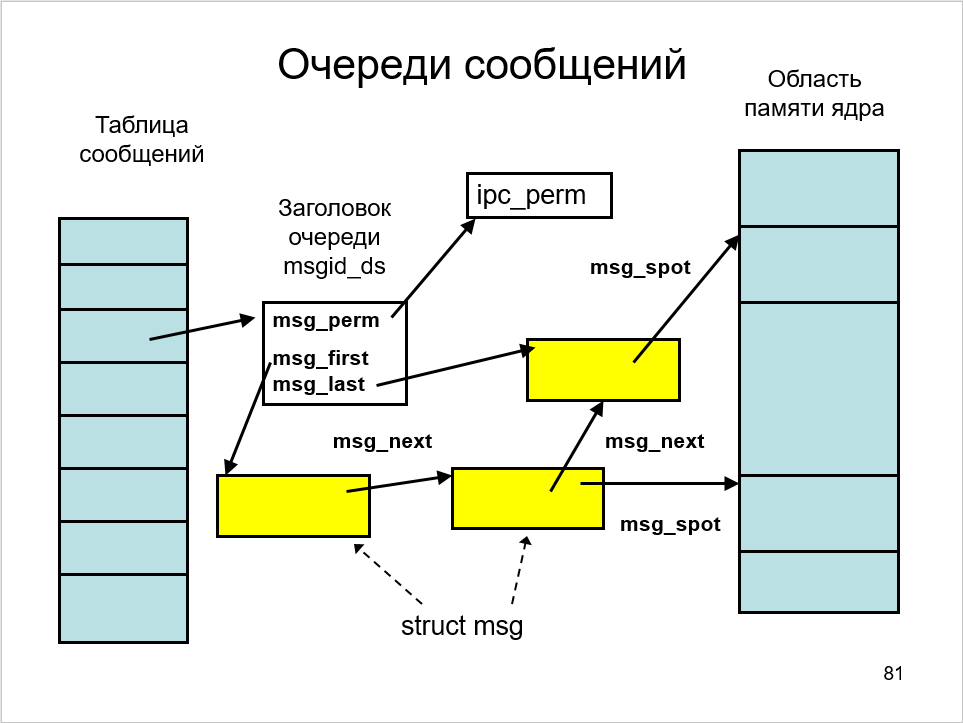
* msgget для образования новой очереди сообщений или получения дескриптора существующей очереди;
* msgsnd для посылки сообщения (вернее, для его постановки в указанную очередь сообщений);
* msgrcv для приема сообщения (вернее, для выборки сообщения из очереди сообщений);
* msgctl для выполнения ряда управляющих действий.

Системный вызов msgget обладает стандартным для семейства "get" системных вызовов синтаксисом:

msgqid = msgget(key, flag);

Ядро хранит сообщения в виде связного списка (очереди), а дескриптор очереди сообщений является индексом в массиве заголовков очередей сообщений. В дополнение к информации, общей для всех механизмов IPC в UNIX System V, в заголовке очереди хранятся также:

* указатели на первое и последнее сообщение в данной очереди;
* число сообщений и общее количество байтов данных во всех них вместе взятых;
* идентификаторы процессов, которые последними послали или приняли сообщение через данную очередь;
* временные метки последних выполненных операций msgsnd, msgrsv и msgctl.



1. **Разделяемая память в ОС UNIX**

Сегменты разделяемой памяти (shared memory segments) – средства, обеспечивающие возможность наличия общей для процессов памяти

Для работы с разделяемой памятью используются четыре системных вызова:

* shmget создает новый сегмент разделяемой памяти или находит существующий сегмент с тем же ключом;
* shmat подключает сегмент с указанным дескриптором к виртуальной памяти обращающегося процесса;
* shmdt отключает от виртуальной памяти ранее подключенный к ней сегмент с указанным виртуальным адресом начала;
* наконец, системный вызов shmctl служит для управления разнообразными параметрами, связанными с существующим сегментом.

После того, как сегмент разделяемой памяти подключен к виртуальной памяти процесса, этот процесс может обращаться к соответствующим элементам памяти с использованием обычных машинных команд чтения и записи.

Синтаксис системного вызова shmget выглядит следующим образом:

shmid = shmget(key, size, flag);

Параметр size определяет желаемый размер сегмента в байтах. Если в таблице разделяемой памяти находится элемент, содержащий заданный ключ (key), и права доступа не противоречат текущим характеристикам обращающегося процесса, то значением системного вызова является дескриптор существующего сегмента. В противном случае создается новый сегмент с размером не меньше установленного.

Подключение сегмента к виртуальной памяти выполняется путем обращения к системному вызову shmat:

virtaddr = shmat(id, addr, flags);

Здесь id - это ранее полученный дескриптор сегмента, а addr - желаемый процессом виртуальный адрес, который должен соответствовать началу сегмента в виртуальной памяти. Значением системного вызова является реальный виртуальный адрес начала сегмента (его значение не обязательно совпадает со значением прямого параметра addr).

Для отключения сегмента от виртуальной памяти используется системный вызов shmdt:

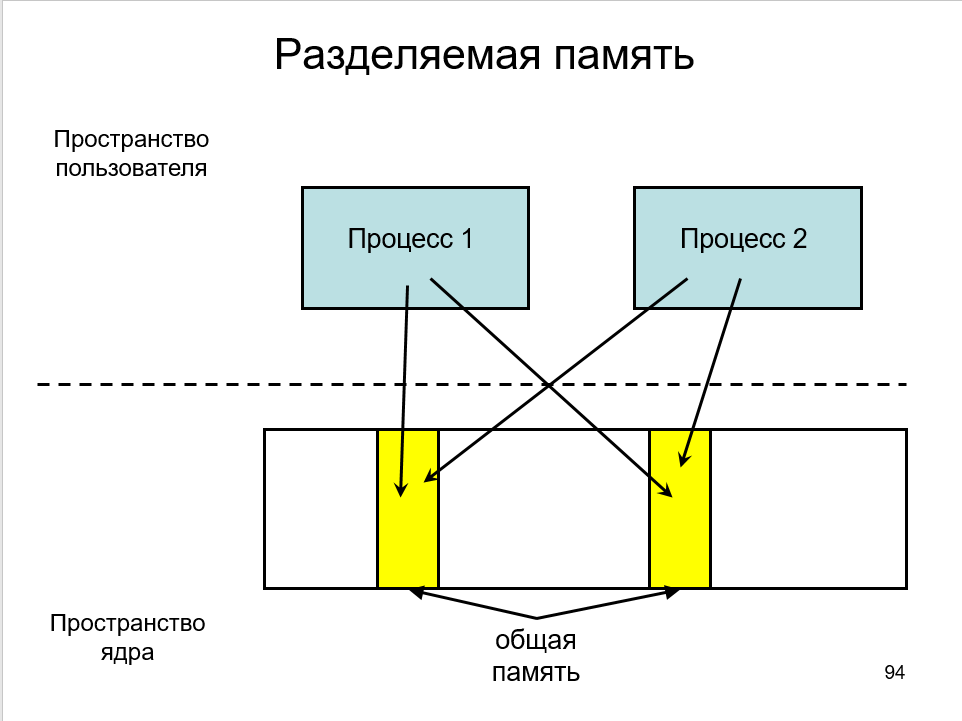
shmdt(addr);

где addr - это виртуальный адрес начала сегмента в виртуальной памяти, ранее полученный от системного вызова shmat. Естественно, система гарантирует (на основе использования таблицы сегментов процесса), что указанный виртуальный адрес действительно является адресом начала (разделяемого) сегмента в виртуальной памяти данного процесса.

Системный вызов shmctl:

shmctl(id, cmd, shsstatbuf);

содержит прямой параметр cmd, идентифицирующий требуемое конкретное действие, и предназначен для выполнения различных функций.



**21. Дескрипторы и атрибуты файлов в ОС Unix.**

Дескриптор файла:

* Имя файла (в операционной системе UNIX имя файла не входит в дескриптор)
* Данные, необходимые для указания на размещение файла
* Способ организации файла
* Тип устройства
* Данные для управления доступом к файлу
* Тип файла (текст, объединенный модуль и др.)
* Время создания, последнего изменения и последнего доступа к файлу

В различных ОС в дескриптор файла может входить и другая информация.

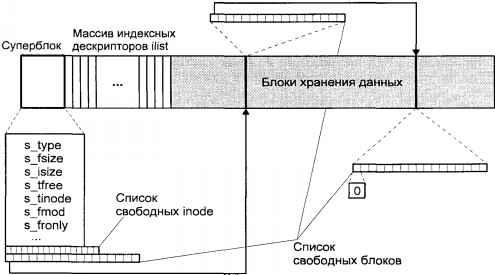
Существует две команды для управления атрибутами файла: lsattr(1) и chattr(1). Команда lsattr выводит список (LiSt) атрибутов, а команда chattr изменяет (CHange) их. Эти атрибуты могут быть установлены только для каталогов и обычных файлов. Доступны следующие атрибуты:

* A («no Access time»): если для файла или каталога установлен этот атрибут, то, всякий раз при обращении к нему для чтения или записи, у него не будет обновляться время последнего доступа. Это может быть полезно, например, для файлов и каталогов, к которым очень часто обращаются для чтения, особенно из-за того, что это единственный параметр в inode, который изменяется при открытии файла только для чтения.
* a («append only»): если для файла установлен этот атрибут, и этот файл открыт для записи, то единственной доступной операцией будет добавление данных к его предыдущему содержимому. Для каталога это означает, что вы сможете только добавить файлы, но не сможете переименовать или удалить ни одного из существующих файлов. Только root может установить или снять этот атрибут.
* d («no dump»): dump (8) - это стандартная утилита UNIX® для резервного копирования. Она делает дамп любой файловой системы, для которой счетчик дампов в файле /etc/fstab (5-е поле) равен 1 (см. главу Глава 8, Файловые системы и точки монтирования). Но если этот атрибут установлен для файла или каталога, то он, в отличие от других, будет будет пропущен при снятии дампа. Обратите внимание, что при установке его для каталогов, это также распространяется на все их подкаталоги и файлы.
* i («immutable»): файл или каталог с установленным этим атрибутом вообще не может быть изменен: он не может быть переименован, на него не может быть создана ссылка[24] и он не может быть удален. Только root может установить или снять этот атрибут. Обратите внимание, что это также предотвращает изменение времени последнего доступа, поэтому вам нет необходимости устанавливать атрибут A, если установлен i.
* s («secure deletion»): когда удаляется файл или каталог с этим атрибутом, блоки, которые он занимал на диске перезаписываются нулями.
* S («Synchronous mode»): если для файла или каталога установлен этот атрибут, все его изменения синхронизируются и немедленно записываются на диск.

**22. Базовая файловая система ОС Unix System V.**

ФС System V состоит из:

* Блока загрузки
* Суперблока
* Списка индексов
* Информационных блоков



**Суперблок:**

* Тип файловой системы (s\_type)
* Размер файловой системы в логических блоках, включая сам суперблок, ilist и блоки хранения данных (s\_fsize)
* Размер массива индексных дескрипторов (s\_isize)
* Число свободных блоков, доступных для размещения (s\_ftree)
* Число свободных inode, доступных для размещения (s\_tinode)
* Размер физического блока (512, 1024, 2048)
* Список номеров свободных inode
* Список адресов свободных блоков

**Структура дискового inode**

di\_mode - Тип файла, дополнительные атрибуты выполнения и права доступа

di\_nlinks - Число ссылок на файл, т. е. количество имен, которые имеет файл в файловой системе

di\_uid, di\_gid - Идентификаторы владельца-пользователя и владельца-группы.

di\_size - Размер файла в байтах. Для специальных файлов это поле содержит старший и младший номер устройства.

di\_atime - Время последнего доступа к файлу.

di\_mtime - Время последней модификации.

di\_ctime - Время последней модификации inode (кроме модификации полей di\_atime, di\_mtime).

di\_addr[13] - Массив адресов дисковых блоков хранения данных.

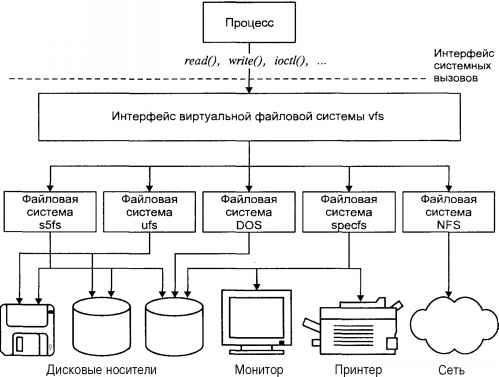
Поле di\_mode хранит несколько атрибутов файла: тип файла (IFREG - для обычных файлов, IFDIR - для каталогов, IFBLK или IFCHR для специальных файлов блочных и символьных устройств соответственно); права доступа к файлу для трех классов пользователей и дополнительные атрибуты выполнения (SUID, SGID и sticky bit). Заметим, что в индексном дескрипторе отсутствует информация о времени создания файла. Вместо этого inode хранит три значения времени: последнего доступа (di\_atime), время последней модификации содержимого файла (di\_mtime) и время последней модификации метаданных файла (di\_ctime). В последнем случае не учитываются модификации полей di\_atime и di\_mtinie. Таким образом, di\_ctime изменяется, когда изменяется размер файла, владелец, группа или число связей.

Блоки хранения данных. Данные обычных файлов и каталогов хранятся в блоках. Обработка файла осуществляется через inode, содержащего ссылки на блоки данных. Блоки хранения данных занимают большую часть дискового раздела, и их число определяет максимальный суммарный объем файлов данной файловой системы. Размер блока кратен 512 байтам, например, файловая система S51K SCO UNIX использует размер блока в 1 Кбайт (отсюда и название).

**23. Архитектура виртуальной файловой системы**

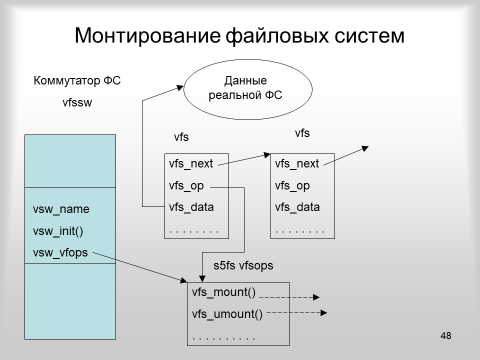
Виртуальная файловая система — уровень абстракции поверх конкретной реализации файловой системы. Целью VFS является обеспечение единообразного доступа клиентских приложений к различным типам файловых систем. VFS может быть использована, например, для прозрачного доступа к локальным и сетевым устройствам хранения данных без использования специального клиентского приложения (независимо от типа файловой системы). VFS определяет интерфейс между ядром и конкретной файловой системой, таким образом, можно легко добавлять поддержку новых типов файловых систем, внося изменения только в ядро операционной системы.

Виртуальная файловая система определяет интерфейс взаимодействия между ядром ОС и конкретной файловой системой, что упрощает поддержку новых типов файловых систем, просто дополняя интерфейс взаимодействия. Определения в интерфейсе могут приводить к изменению совместимости от релиза к релизу, что требует поддержки файловой системой перекомпиляции и возможной модификации перед компиляцией; или же, поставщик ОС может вносить изменения только с обратной совместимостью, так что поддержка конкретной файловой системы, построенной для данного релиза ОС, будет работать с будущими версиями. Иногда виртуальными файловыми системами называют псевдо-файловые системы, которые не предназначены для хранения данных. Примером такой системы является procfs(виртуальная файловая система, используемая в UNIX-подобных операционных системах).



Операция VFS\_MOUNT выполняет традиционное для UNIX монтирование файловой системы на указанный каталог уже смонтированной файловой системы для образования общего дерева, а операция VFS\_UNMOUNT отменяет монтирование. Операция VFS\_ROOT используется при разборе полного имени файла, когда встречается дескриптор vnode, который связан со смонтированной на него файловой системой. Операция VFS\_ROOT помогает найти vnode, который является корнем смонтированной файловой системы. Операция VFS\_STATVFS позволяет получить независимую от типа файловой системы информацию о размере блока файловой системы, о количестве блоков и количестве свободных блоков в единицах этого размера, о максимальной длине имени файла и т.п. Операция VFS\_SYNC выталкивает содержимое буферов диска из оперативной памяти на диск. Операция VFS\_MOUNTROOT позволяет смонтировать корневую файловую систему, то есть систему, содержащую корневой каталог / общего дерева. Для указания того, какая файловая система будет монтироваться как корневая, в UNIX System V Release 4 используется переменная rootfstype, содержащая символьное имя корневой файловой системы, например "ufs".

**24. Монтирование и доступ к файловым системам.**



**Структура виртуальной файловой системы (VFS)**

* Struct vfs \*vfs\_next – Следующая файловая система в списке
* Struct vfsops \*vfs\_op – Операции файловой системы
* Struct vnode \*vfs\_vnjdecovered – vnode, перекрываемой файловой системы
* Int vfs\_flag – флаги доступа
* Int vfs\_bsize – Размер блока файловой системы
* Caddr\_t vfs\_data – Указатель на специфические данные, относящиеся к реальной файловой системы.

**Операции файловой системы**

* VFS\_MOUNT монтирование файловой системы
* VFS\_UNMOUNT размонтирование файловой системы
* VFS\_ROOT получение vnode для корня файловой системы
* VFS\_STATVFS получение статистики файловой системы
* VFS\_SYNC выталкивание буферов файловой системы на диск
* VFS\_VGET получение vnode по номеру дескриптора файла

VFS\_MOUNTROOT монтирование корневой файловой системы

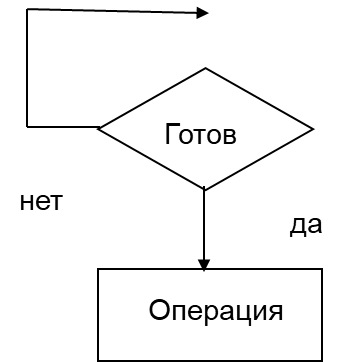
**25 Основные функции подсистемы управления вводом-выводом. Способы организации обмена данными. Управление обменом. Буферизация**

Функции подсистемы ввода-вывода

* Организация параллельной работы устройств ввода-вывода и процессора
* Согласование скоростей обмена и кэширование данных
* Разделение устройств между процессами
* Обеспечение удобного логического интерфейса между устройствами и остальной частью системы
* Поддержка широкого спектра драйверов с возможностью простого включения в систему нового драйвера
* Динамическая загрузка и выгрузка драйверов
* Поддержка нескольких файловых систем
* Поддержка синхронных и асинхронных операций ввода-вывода

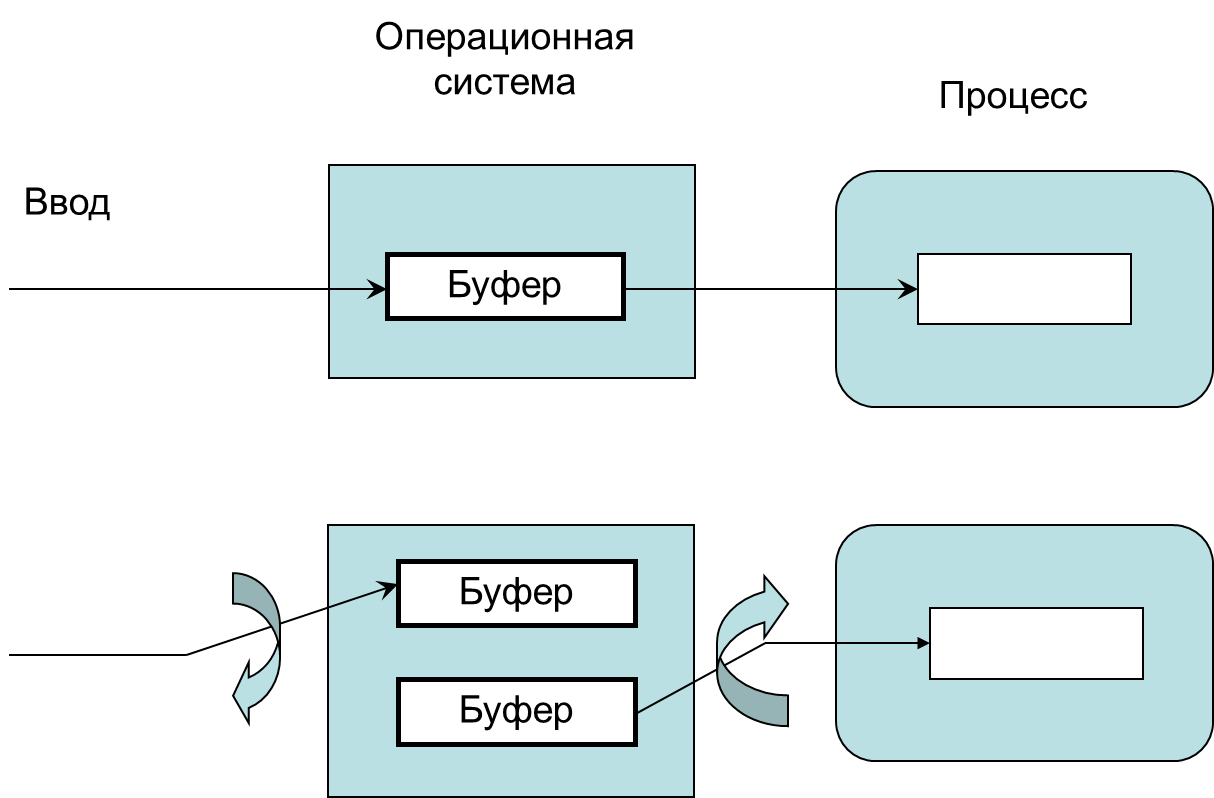
Принципы организации ввода-вывода

* Синхронный обмен
  + Обмен по запросу
  + Обмен с ожиданием готовности
* Асинхронный обмен
* Прямой доступ к памяти



Буферизация

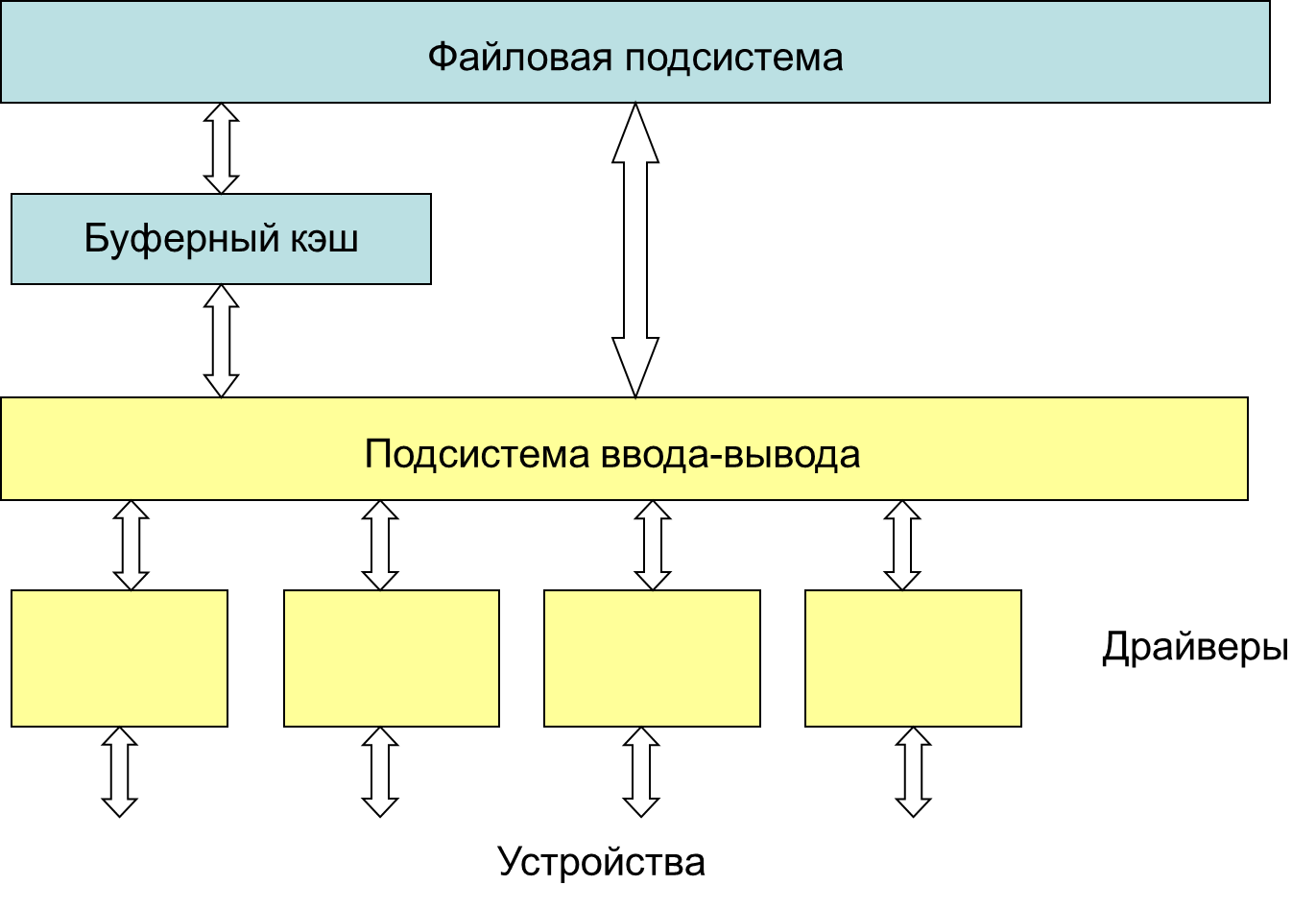
Буферизация — способ организации обмена, в частности, ввода и вывода данных в компьютерах и других вычислительных устройствах, который подразумевает использование буфера для временного хранения данных. При вводе данных одни устройства или процессы производят запись данных в буфер, а другие — чтение из него, при выводе — наоборот. Процесс, выполнивший запись в буфер, может немедленно продолжать работу, не ожидая, пока данные будут обработаны другим процессом, которому они предназначены.

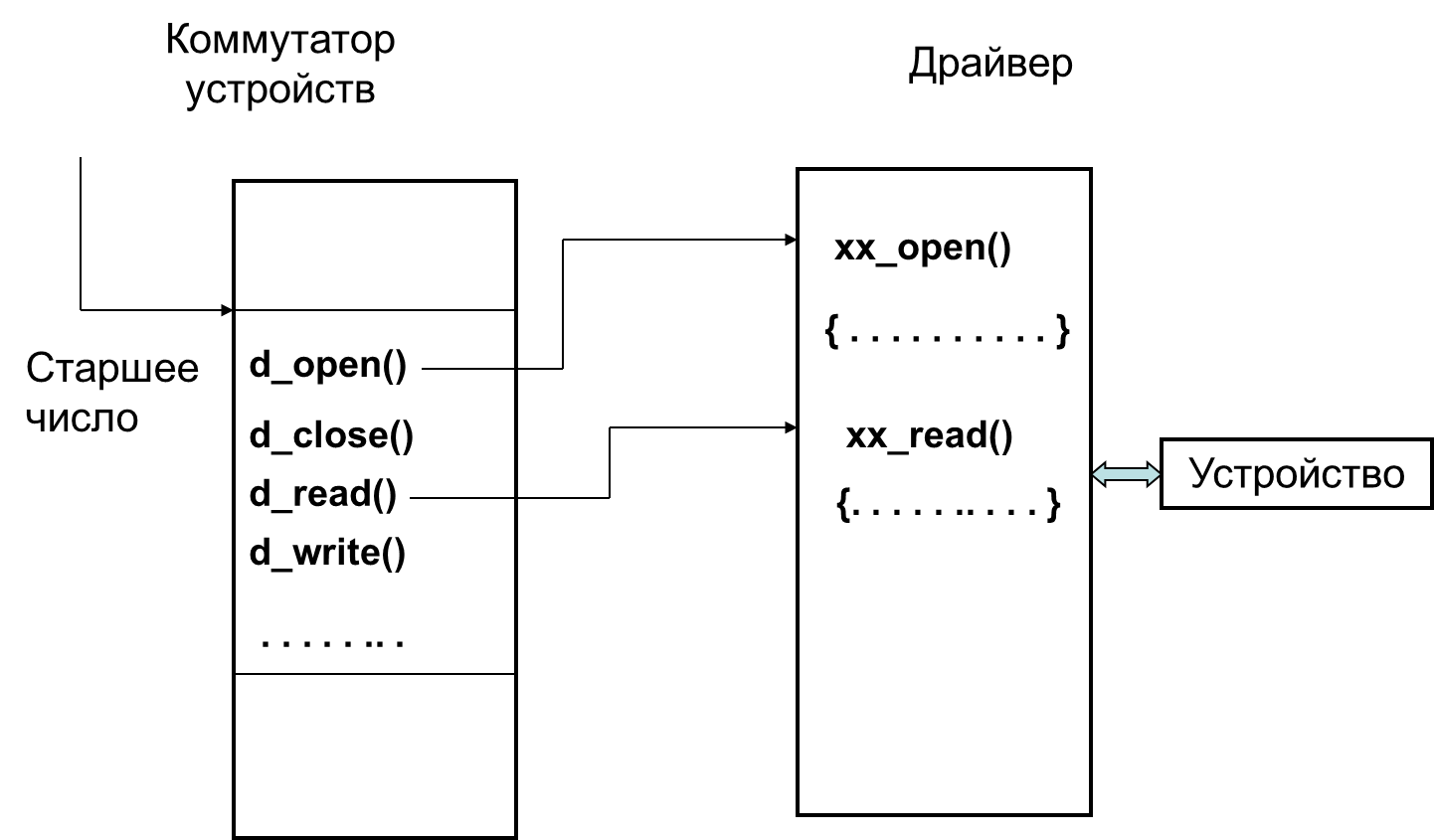


**26 Организация ввода и вывода в ОС UNIX. Драйверы устройств.**

Ввод и вывод в ОС UNIX

* Типы драйверов
  + Символьные драйверы
  + Блочные драйверы
  + Драйверы низкого уровня
  + Драйверы псевдоустройств (/dev/kmem, /dev/ksyms, /dev/mem, /dev/null, /dev/zero)
* Адресация драйвера
  + Старший номер – major number
  + Младший номер – minor number
* Коммутаторы устройств – bdevsw и cdevsw



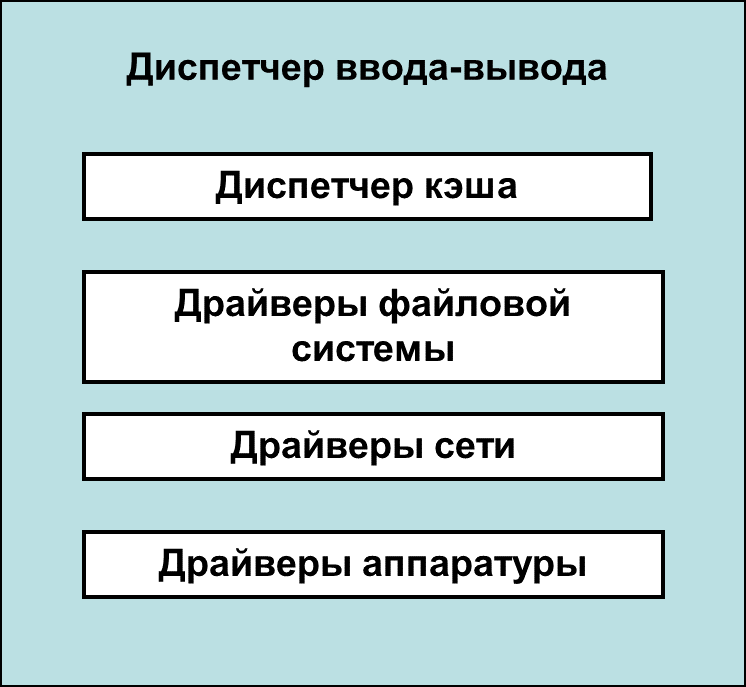


Точки входа

* xxopen() – открытие устройства
* xxclose() – закрытие устройства
* xxread() – чтение данных для символьного устройства
* xxwrite() – запись данных для символьного устройства
* xxioctl() – управление символьным устройством
* xxintr() – обработка прерываний
* xxsrategy() – общая точка входа для операций блочного ввода-вывода, ввод-вывод инициируются прерываниями

**27 Организация вывода в ОС Windows NT**

Ввод и вывод Windows 2000



Основные модули

* Диспетчер кэша. Управляет кэшированием всей подсистемы ввода-вывода.
  + Отложенная запись. Записи обновляются только в кэше. На диск записывается только последняя версия обновления.
  + Отложенное подтверждение. Работа с транзакциями.
* Драйверы файловой системы. Работа с томами.
* Драйверы сети. Интегрированные сетевые возможности и поддержка распределенных приложений.
* Драйверы аппаратуры. Работа с регистрами периферийных устройств.

Режимы ввода-вывода

* Асинхронный режим – приложение инициирует операцию и продолжает работу. Способы оповещения о завершении.
  + Сигнал объекту устройства ядра.
  + Сигнал объекту события ядра.
  + Оповещение о вводе-выводе.
  + Порты завершения ввода-вывода
* Синхронный режим – приложение блокируется до завершения операции.

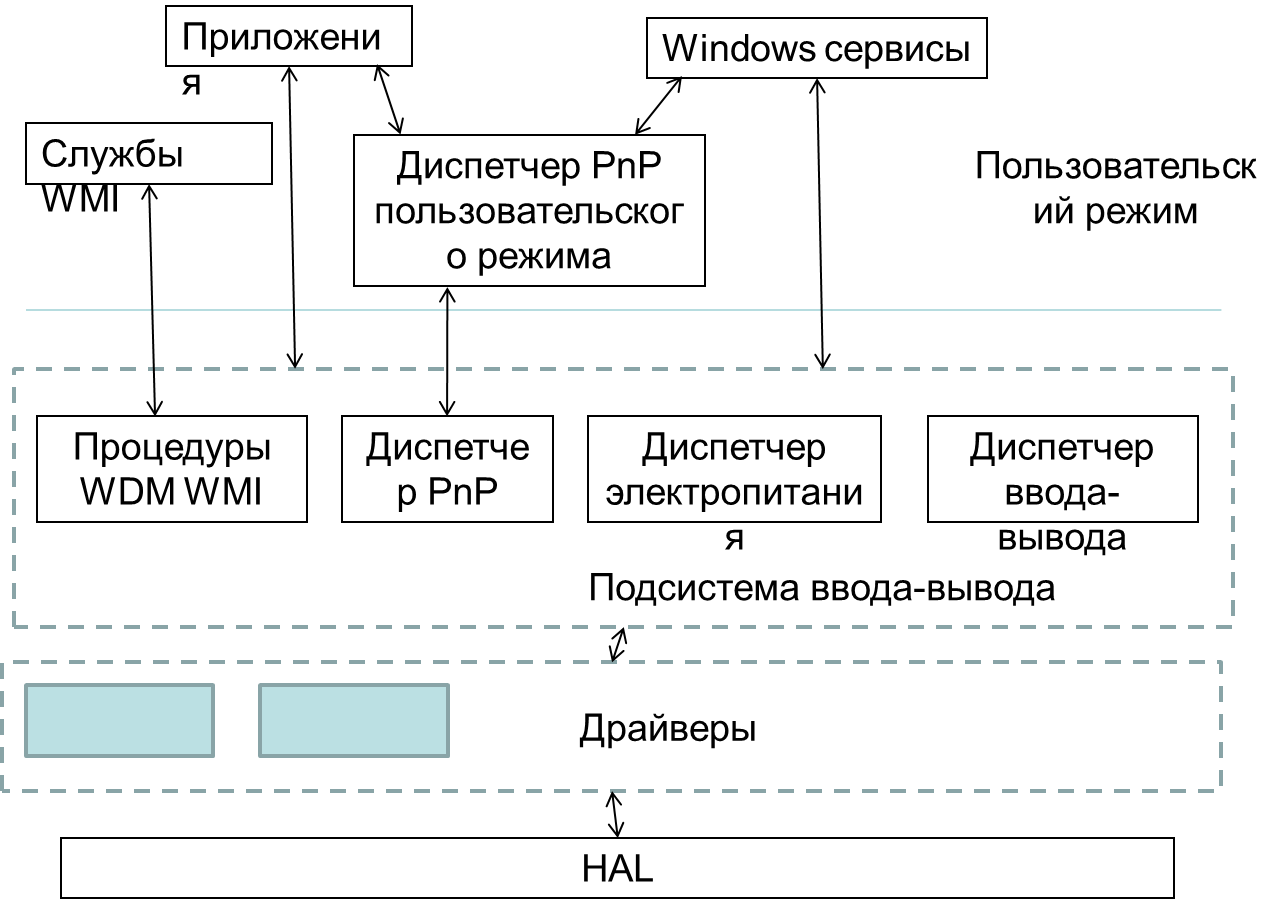
Типы драйверов (WDM)

* Драйверы пользовательского режима (UMD)
  + Драйверы виртуальных устройств (VDD)
  + Драйверы принтеров
* Драйверы режима ядра (KMD)
  + Драйверы файловой системы
  + Унаследованные драйверы
  + Драйверы видеоадаптеров
  + Драйверы потоковых устройств
  + WDM - драйверы

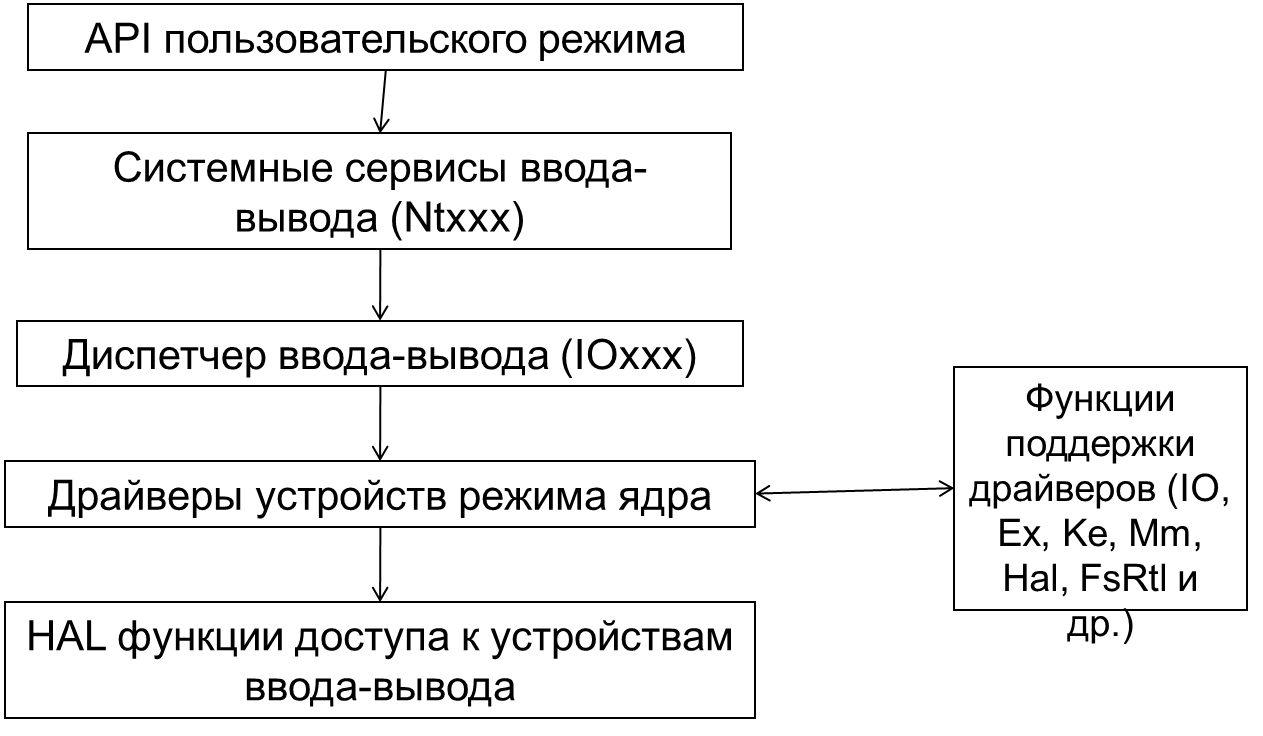
Уровни драйверов

* Уровни типа драйверов
  + Шинные драйверы
  + Фильтр – драйверы
  + Функциональные драйверы
* Фильтр – драйверы
  + Фильтр-драйверы шины
  + Фильтр-драйверы устройства и классовый фильтр-драйвер
  + Функциональный драйвер
  + Вышестоящие фильтр-драйверы устройства и классовый фильтр-драйвер

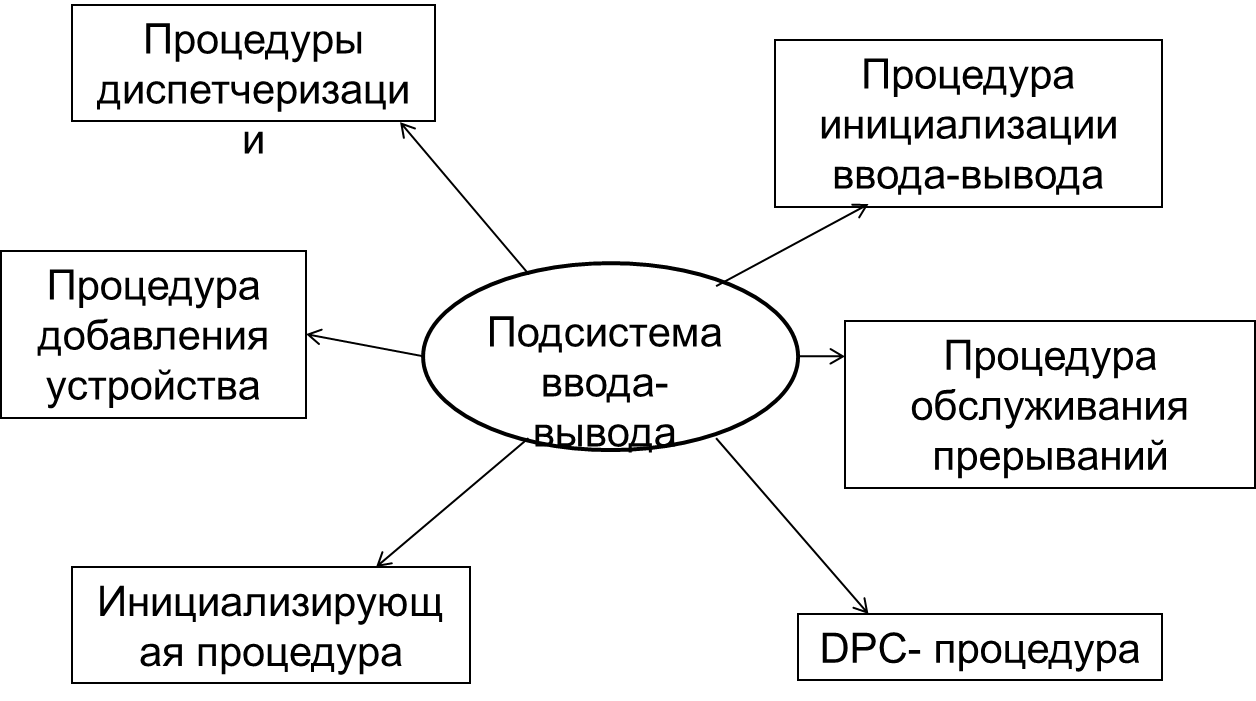
Компоненты подсистемы ввода-вывода



Базовая схема обработки запроса ввода-вывода



Основные процедуры драйвера



**28) Команды интерпретатора bash. Переназначение ввода и вывода в командном интерпретаторе**

bash — усовершенствованная и модернизированная вариация командной оболочки Bourne shell. Одна из наиболее популярных современных разновидностей командной оболочки UNIX. Особенно популярна в среде Linux, где она часто используется в качестве предустановленной командной оболочки.

Интерпретатор bash имеет множество встроенных команд, часть из которых имеют аналогичные исполняемые файлы в операционной системе. Однако следует обратить внимание, что чаще всего для встроенных команд отсутствуют man-страницы, а при попытке просмотра справки по встроенной команде на самом деле будет выдаваться справка по исполняемому файлу. Исполняемый файл и встроенная команда могут отличаться параметрами. Информация по встроенным командам расписана в справочной странице bash

|  |  |
| --- | --- |
| Echo | выводит выражение или содержимое переменной (stdout), но имеет ограничения в использовании |
| Sourse | Эта команда позволяет прочитать и выполнить все команды из указанного файла в текущей командной оболочке. |
| Printf | команда форматированного вывода, расширенный вариант команды echo |
| Read | «читает» значение переменной со стандартного ввода (stdin), в интерактивном режиме это клавиатура |
| Cd | изменяет текущий каталог |
| Pwd | выводит название текущего рабочего каталога |
| Dirs | Команда dirs отображает историю смены текущей папки. Каждая предыдущая папка отображается с новой строки. |
| Exit | Универсальная команда, которая позволяет закрыть сеанс оболочки, выйти из терминала или завершить текущую сессию. |

Перенаправление ввода-вывода — возможность командной оболочки ряда операционных систем перенаправлять стандартные потоки в определённое пользователем место, например, в файл. Характерна для Unix-подобных операционных систем, но в разной степени реализована и в операционных системах других семейств.

Стандартный ввод при работе пользователя в терминале передается через клавиатуру. Стандартный вывод и стандартная ошибка отображаются на дисплее терминала пользователя в виде текста. Ввод и вывод распределяется между тремя стандартными потоками, так же они пронумерованы:

stdin - стандартный ввод (клавиатура) - 0

stdout - стандартный вывод (экран) - 1

stderr - стандартная ошибка (вывод ошибок на экран) - 2

Из стандартного ввода команда может только считывать данные, а два других потока могут использоваться только для записи. Данные выводятся на экран и считываются с клавиатуры, так как стандартные потоки по умолчанию ассоциированы с терминалом пользователя. Потоки можно подключать к чему угодно: к файлам, программам и даже устройствам. В командном интерпретаторе bash такая операция называется перенаправлением:

< file - Использовать файл как источник данных для стандартного потока ввода

> file - Направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху

2> file - Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху

>>file - Направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец

2>>file - Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец

&>file или >&file - Направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: >file 2>&1

**29) Общие сведения об ОС Windows NT.**

Windows NT — линейка операционных систем (ОС) производства корпорации Microsoft и название первых версий ОС. Windows NT была разработана после прекращения сотрудничества Microsoft и IBM над OS/2, развивалась отдельно от других ОС семейства Windows (Windows 3.x и Windows 9x).

В отличие от Windows 3.x и Windows 9x, Windows NT позиционировалась как надёжное решение для рабочих станций (Windows NT Workstation) и серверов (Windows NT Server). Windows NT дала начало семейству операционных систем, в которое входят: собственно Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8

Архитектура Windows NT имеет модульную структуру и состоит из двух основных уровней — компоненты, работающие в режиме пользователя, и компоненты режима ядра. Программы и подсистемы, работающие в режиме пользователя, имеют ограничения на доступ к системным ресурсам. Режим ядра имеет неограниченный доступ к системной памяти и внешним устройствам. Ядро системы NT называют гибридным ядром или макроядром. Архитектура включает в себя само ядро, уровень аппаратных абстракций, драйверы и ряд служб, которые работают в режиме ядра или в пользовательском режиме Пользовательский режим Windows NT состоит из подсистем, передающих запросы ввода-вывода соответствующему драйверу режима ядра посредством менеджера ввода-вывода. Есть две подсистемы на уровне пользователя: подсистема окружения (запускает приложения, написанные для разных операционных систем) и интегрированная подсистема (управляет особыми системными функциями от имени подсистемы окружения). Режим ядра имеет полный доступ к аппаратной части и системным ресурсам компьютера. И также предотвращает доступ к критическим зонам системы со стороны пользовательских служб и приложений.

История ОС Windows делится на 2 части - до выхода Windows Vista и после. Ничего принципиально нового (кроме визуальных эффектов) в интерфейс этой версии своей ОС разработчики Microsoft не привнесли, однако полностью было переписано ядро системы.

Из-за нового ядра была потеряна совместимость со старыми программами (разработка которых остановилась до выхода Vista) и драйверами старых устройств. Этим в большей мере объясняется дурная слава и непопулярность этой версии Windows. В Windows7 разработчики учли ошибки, производители софта и железа уже имели возможность выпускать ПО и оборудование с учётом новых требований ОС, однако большинство старых программ и старого оборудования этой операционной системой уже не поддерживается.