**9. Дайте определения следующим терминам: «Виртуальный адрес», «Виртуальное адресное пространство». Объясните, чем определяется максимально возможный объем виртуального адресного пространства программы.**

Виртуальный адрес - это указатель или маркер для области памяти, по которому процесс обращается к системе виртуальной памяти. Виртуальный адрес указывает на местоположение в первичном хранилище, которое процесс может использовать независимо от других процессов.

В устройствах с управлением памятью виртуальный адрес отличается от адреса физической памяти. В таких устройствах блок управления памятью (MMU) отвечает за управление памятью, включая преобразование виртуальных адресов в физические адреса. Благодаря виртуальным адресам система управления памятью может выделять огромные объемы памяти отдельным процессам. Система позволяет каждому процессу предполагать, что он имеет всю доступную память для себя, когда фактически операционная система перемещает память между процессами по мере необходимости.  
Виртуальный адрес также используется в контексте виртуальных систем. Как и в случае адресов виртуальной памяти, более новые системы заменяют места назначения дисков физической памяти системами виртуальной памяти, где оборудование разделено на различные и более сложные типы хранилищ.

Виртуальное адресное пространство — это максимально доступное приложению адресное пространство. Объём виртуального адресного пространства зависит от архитектуры компьютера и операционной системы. Он зависит от архитектуры компьютера, так как именно архитектура определяет, сколько бит используется для адресации. Он также зависит от операционной системы, так как в зависимости от реализации операционная система может накладывать дополнительные ограничения, помимо ограничений архитектуры.

Максимальный размер виртуального адресного пространства ограничивается только разрядностью адреса, присущей данной архитектуре компьютера, и, как правило, не совпадает с объемом физической памяти компьютера. Часто бывает, что объем виртуального адресного пространства превышает доступный объем оперативной памяти. В таком случае операционная система для хранения данных виртуального адресного пространства, не помещающихся в оперативную память, использует дисковую память. Выполняется виртуализация оперативной памяти – виртуальная память.

**47. Поясните, что используется в Linux для взаимодействия пользователя с системой.**

Чтобы обеспечить взаимодействие пользователя с операционной системой и с прикладными программами необходим интерфейс: система передачи команд пользователя операционной системе и ответов системы обратно пользователю. Такое взаимодействие представляет собой «диалог» пользователя с компьютером на специальном языке, будь то язык, использующий знаки, похожие на слова и высказывания естественного языка, или язык изображений. На сегодня известны две принципиальные возможности организации интерфейса: графический интерфейс и командная строка.

Командная строка — приглашение оболочки, обозначающее готовность системы принимать команду пользователя, — в наиболее явной форме демонстрирует идею диалога. На каждую введенную команду пользователь получает ответ от системы: либо очередное приглашение, обозначающее, что команда выполнена и можно вводить следующую, либо сообщение об ошибке, представляющее собой высказывание системы о произошедших в ней событиях, адресованное пользователю. При работе в операционной среде с графическим интерфейсом происходящий диалог пользователя с системой не столь очевиден, хотя с точки зрения системы клик мышью в определенной области на экране аналогичен команде, введенной с клавиатуры, а ответ системы пользователю может быть представлен в виде диалогового окна.

Традиционный способ взаимодействия пользователя с системой LINUX основывается на использовании командных языков. После входа пользователя в систему для него запускается один из командных интерпретаторов (в зависимости от параметров, сохраняемых в файле /etc/passwd). Обычно в системе поддерживается несколько командных интерпретаторов с похожими, но различающимися своими возможностями командными языками. Общее название для любого командного интерпретатора ОС LINUX - shell (оболочка), поскольку любой интерпретатор представляет внешнее окружение ядра системы.

Вызванный командный интерпретатор выдает приглашение на ввод пользователем командной строки, которая может содержать простую команду, конвейер команд или последовательность команд. После выполнения очередной командной строки и выдачи на экран терминала или в файл соответствующих результатов, shell снова выдает приглашение на ввод командной строки, и так до тех пор, пока пользователь не завершит свой сеанс работы путем ввода команды logout или нажатием комбинации клавиш Ctrl-d.

Командные языки, используемые в ОС LINUX, достаточно просты, чтобы новые пользователи могли быстро начать работать, и достаточно мощны, чтобы можно было использовать их для написания сложных программ. Последняя возможность опирается на механизм командных файлов (shell scripts), которые могут содержать произвольные последовательности командных строк. При указании имени командного файла вместо очередной команды интерпретатор читает файл строка за строкой и последовательно интерпретирует команды.

Графический интерфейс в Linux строится на основе стандарта X Window System, разработка которого была начата в 1984 году. Первые 10 версий X Window System были разработаны всего тремя людьми - Робертом Шейфлером (Robert Sheifler), Джимом Геттисом (Jim Gettys) и Роном Ньюменом (Ron Newman).   Начиная с 1988 г. этот стандарт поддерживался консорциумом X, созданным с целью унификации графического интерфейса для ОС UNIX а также ОС Linux. В 1997 году консорциум X был преобразован в X Open Group.

Свободно распространяемая реализация стандарта X11R6 была создана группой программистов, которую вначале возглавлял Дэвид Вексельблат (David Wexelblat). Эта реализация известна как XFree86, и может использоваться не только в Linux, но и в других версиях UNIX для систем на базе процессоров Intel 80386/80486/Pentium (например, FreeBSD).

Операционная система UNIX с самого начала была многопользовательской, многозадачной системой, работавшей в режиме разделения времени. При этом она позволяла пользователям работать в удаленном режиме, либо через терминалы, либо с использованием сетевых технологий. Эти основные концепции были учтены при создании графического интерфейса для UNIX и поэтому система X Window построена на основе модели "клиент/сервер". Правда, модель эта в данном случае используется как бы в "перевернутом" виде. Дело в том, что X-сервер запускается на компьютере пользователя (а не на каком-то удаленном "сервере") и обеспечивает вывод изображения на экран монитора. Эта программа работает непосредственно с "железом" и обеспечивает управление как устройствами ввода (клавиатура, мышь и так далее), так и устройствами вывода (дисплей, монитор, динамик). X-сервер "захватывает" оборудование и предоставляет его возможности другим программам (клиентам сервера) как ресурсы (собственно, именно поэтому он и считается сервером) по особому протоколу, который называется X-протокол, или протокол сетевой связи (X Network Protocol). Кстати, специализированный компьютер, на котором исполняется исключительно X-сервер, называется (аппаратным) X-терминалом.

Если запустить только X-сервер, виден просто серый экран с характерным крестиком курсора посредине. Дело в том, что сам X-сервер изображение не формирует, он только "доставляет" графику видеоадаптеру и передает сообщения о событиях от аппаратной части (в частности, от клавиатуры и мыши, то есть сообщения о действиях пользователя) своим клиентам, а клиенты пока не запущены. Хотя на самом деле некоторые комбинации клавиш X перехватывает и обрабатывает.

Чтобы получить на экране изображения, одного X-сервера недостаточно, надо запустить менеджер окон и хотя бы одну программу-клиент, которая будет формировать изображение и обрабатывать сообщения о действиях пользователя (например, щелчок кнопкой мыши и т.п.).

Между клиентами и сервером стоят еще два очень важных компонента графического интерфейса: библиотека графических функций X-Lib и менеджер окон. Менеджер окон – это особое приложение, которое тоже является клиентом X-сервера, но выполняет особую роль – оно управляет другими клиентами. Одна из основных функций, которые выполняет любой менеджер окон – это обеспечение возможности запуска других приложений. Кроме того, менеджер окон обеспечивает выполнение всех операций с окнами: прорисовку рамок, меню, иконок, полос прокрутки и других элементов окна, предоставляет возможность изменять вид и положение окна в процессе работы в соответствии с потребностями пользователя, а также обеспечивает вывод информации от приложения в соответствующее окно. Можно запустить X-сервер без оконного менеджера, но не наоборот. Именно с менеджером окон взаимодействует пользователь при работе в графическом режиме. Менеджер окон вызывает соответствующие функции для программ-клиентов в тех случаях, когда пользователь работает с приложением с помощью клавиатуры и мыши.

Взаимодействие между менеджером окон и X-сервером осуществляется в асинхронном режиме путем обмена сообщениями. Клиентские программы открывают соединение с сервером, и затем просто посылают ему запросы примерно такого типа: “нарисуй прямую линию от точки такой-то до точки такой-то” или “выведи эту строку текста таким-то шрифтом начиная с такой-то позиции экрана”. Эти запросы, в основном, обрабатываются путем вызова соответствующих процедур из библиотеки X-Lib, которая содержит набор стандартных функций, которые обеспечивают выполнение низкоуровневых операций с графическими образами. Менеджер окон вызывает функции из X-Lib для управления дисплеем и выполнения любых преобразований изображений в окнах. Надо сказать, что хотя элементы графического интерфейса (иконки, кнопки, диалоговые окна, линейки прокрутки, различные рамки и оконные меню) и прорисовываются на экране с помощью низкоуровневых функций из библиотеки X-Lib, но приложения не вызывают эти функции непосредственно, а обращаются к функциям более высокого уровня, которые в англоязычной документации называют виджетами – "widgets”. Ведь чтобы нарисовать, например, с помощью X-Lib простую кнопку с надписью «Ввод», необходимо прорисовать два прямоугольника, скруглить углы, изобразить тень, вывести надпись, обеспечить изменение вида кнопки при наведении мыши и так далее. Чтобы не повторять каждый раз работу по программированию часто используемых элементов графического интерфейса, были разработаны несколько отдельных библиотек таких элементов (виджетов). Эти библиотеки иногда называют "тулкитами" ("toolkit"). Наиболее известными из них являются библиотеки Motif, Qt и GTk. Библиотека Motif широко использовалась в 1980-х и начале 1990-х годов. Наиболее известным приложением, построенным на ее основе, является Netscape Communicator. В наше время эта библиотека уже не так популярна, поскольку появились более совершенные разработки, причем бесплатные (Motif распространялась на коммерческой основе). Библиотека GTk была разработана как замена Motif для проекта GIMP (GTk иногда расшифровывают как GIMP Toolkit или GNU Toolkit). Библиотека Qt получила широкое распространение с появлением проекта KDE, который использует ее для создания всех элементов графического интерфейса. Еще одной библиотекой, заслуживающей упоминания, является LessTif – бесплатный аналог Motif.

Если теперь все сказанное выше изобразить графически, получим рис.1.

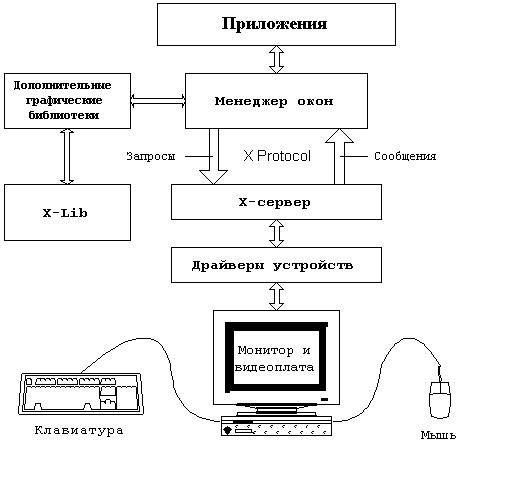


Рис.1. Архитектура системы X Window

Конечно, рисунок этот очень схематичен. Рассмотрим, к примеру, обмен сообщениями между X-сервером и менеджером окон. Существует 4 типа сообщений, передаваемых между клиентом и сервером:

Запрос – клиент требует нарисовать что-либо в окне или запрашивает у сервера информацию; Ответ – сервер отвечает на запрос; Событие – сервер сообщает клиенту о событии (например, о нажатии клавиши пользователем);  
Ошибка – сервер сообщает об ошибке.

Когда X-сервер и X-клиент работают на одной машине, обмен запросами и сообщениями между севером и клиентом осуществляется через локальный сокет. Но существует возможность передавать эти сообщения по сети, используя стек протоколов TCP/IP. Это свойство X Window является очень большим преимуществом этой системы в сравнении с другими типами графических оболочек, так как приложения могут быть запущены как на той же машине, на которой работает X-сервер, так и на другом компьютере, который может даже не иметь собственной клавиатуры и мыши. Все, что нужно программе для работы, - это знать, где искать X-сервер (для этого используется либо переменная окружения DISPLAY, либо опция в командной строке). Более того, существуют даже программы (например, emacs), которые умеют работать и с X-сервером и с обычным текстовым терминалом и сами разбираются при старте, как именно им работать в данном случае.

Поскольку взаимодействие менеджера окон с сервером в общем случае осуществляется по протоколам TCP/IP, на приведенном рисунке следовало бы еще отобразить программное обеспечение, реализующее эти протоколы. Еще одним важным ресурсом графической подсистемы являются шрифты. Оперировать со шрифтами может как непосредственно X-сервер, так и специальная программа, которая называется сервер шрифтов, и которую также следовало бы включить в рисунок.

Можно еще отметить, что X-сервер может обеспечивать работу с несколькими "дисплеями" и "экранами" одновременно. Правда, термины “экран” и “дисплей” в системе X Window имеют специальное значение. Когда X-сервер запускается, он инициализирует один или несколько “дисплеев”. Каждый “дисплей” включает в себя не только видео-компонент, но и клавиатуру, мышь и другие устройства ввода. Конечно, отдельный пользователь в любой момент времени посредством одного комплекта устройств (монитора, клавиатуры и мыши) имеет доступ только к одному “дисплею”. Но имеется возможность запустить несколько “дисплеев”, которые могут быть как локальными (виртуальными), так и расположенными на других хостах, соединенных с данным компьютером по сети. Каждый “дисплей” может иметь собственную конфигурацию (например, другое разрешение). Пользователь может выбирать, какой дисплей он хочет использовать при входе в систему. Однако в самой типичной ситуации используется только один дисплей (с номером 0).

“Экран” ("screen") по отношению к X-серверу означает то изображение, которое вы видите, когда запускаете X Window. “Экранов” тоже может быть несколько, так же, как и “дисплеев”. Дополнительные “экраны" используются в случае “дисплеев с многими экранами”. Фактически имеется возможность использовать один X-сервер на несколько компьютеров.

Приложения можно запустить, например, на мейнфрейме, а картинка будет выводиться на экран персонального компьютера. Отметим еще, что в стандартном режиме запросы и сообщения буферизуются и обрабатываются сервером в асинхронном режиме, что позволяет повысить скорость работы и снизить нагрузку на сеть. При желании клиент может перейти на синхронный режим работы, хотя этот режим работает в 30 раз медленнее (так что обычно используется только для отладки программ).

Один и тот же X-сервер может одновременно использоваться многими приложениями. Для того, чтобы обеспечить одновременную работу различных приложений и уменьшить объем информации, передаваемой по сети, сервер для каждого из приложений хранит данные об используемых приложением шрифтах, информацию о цвете и многое другое. Эти данные в совокупности называются ресурсами сервера. Клиенты имеют возможность ссылаться на эти ресурсы с помощью кодов ресурсов, сокращая тем самым сетевой трафик и облегчая доступ к данным для различных процедур.

Графический интерфейс в Linux строится по модульному принципу, причем существует достаточно много вариантов выбора для каждой из основных компонент. Различных менеджеров окон, например, существует очень много, а набор приложений вообще необозрим. Так что можно построить собственную графическую среду, использовав любой набор ее составных частей по выбору. Однако тут возникают некоторые проблемы.

Первая такая проблема связана с разнообразием тех подходов к взаимодействию с пользователем, которые применяются в разных программах. Некоторые приложения, работающие в графическом режиме, используют широко распространенные библиотеки графических элементов типа Qt или GTk, а другие строятся на основе собственных наборов графических элементов. Наборы графических элементов обычно не являются взаимозаменяемыми, поскольку используют различные программные интерфейсы. С другой стороны, не существует никаких ограничений или требований для разработчиков приложений как в части употребления тех или иных наборов графических элементов, так и в части, например, способов запуска программ (через меню, командную строку или набором определенной комбинации клавиш). Это, естественно, вызывает трудности в работе пользователей. Но более существенно то, что это вызывает нерациональные траты ресурсов компьютера. Если 6 разных приложений используют 6 разных библиотек графических элементов, то нужно хранить эти 6 библиотек на диске, так и загружать их в память, хотя Linux (и UNIX вообще) поддерживают возможность использования динамически разделяемых библиотек.

Эти недостатки отсутствуют в так называемых интегрированных графических средах или оболочках. В таких средах все основные компоненты графического интерфейса, а также множество специально разработанных приложений, строятся на основе единой графической библиотеки и единого подхода к организации интерфейса. При этом графические среды типа CDE, KDE, GNOME, GNUStep и т.д. и не замещают перечисленные выше компоненты системы X Window, а расширяют и дополняют их. KDE, например, использует библиотеку графических функций Qt и собственный менеджер окон kwm, который управляет поведением всех окон. Кроме того, KDE дополняет Qt своей особой библиотекой (kdelibs) для выполнения таких задач, как создание меню, диалоговых окон или окон сообщений, организации взаимодействия программ, печати, и других задач.

Существует два основных варианта запуска графического интерфейса пользователя в системе Linux. В первом варианте X-сессия запускается менеджером дисплея, после чего пользователь получает возможность войти в систему (логироваться) непосредственно в графическом режиме. Во втором варианте пользователь вначале входит в систему в текстовом режиме, а потом запускает X-сессию с помощью программы xinit (чаще всего для этого используется скрипт startx, который является просто оболочкой для программы запуска графического режима xinit). В любом случае система X Window запускается с правами суперпользователя, поскольку ей требуется доступ к аппаратным устройствам.

Недостатком первого варианта (через менеджер дисплея) является то, что если возникнут какие-то проблемы с переходом в графику, вы оказываетесь в затруднительном положении – ведь вы еще не вошли в систему и поправить что-либо нет возможности. Эта ситуация, конечно, не является безвыходной, однако начинающему пользователю все же лучше при инсталляции системы отказаться от автоматической загрузки графического режима и запускать его «вручную» из текстового.

Таким образом, для взаимодействия пользователя с системой можно использовать либо графический интерфейс на основе стандарта X Window System, либо командную строку.

**88. Перечислите основные функции драйвера.**

Для разработки драйверов производителями внешних устройств необходимо наличие четкого, удобного, открытого и хорошо документированного интерфейса между драйверами и другими компонентами ОС. Драйвер взаимодействует, с одной стороны, с модулями ядра ОС (модулями подсистемы ввода-вывода, модулями системных вызовов, модулями подсистем управления процессами и памятью), а с другой стороны – с контроллерами внешних устройств. Поэтому существует два вида интерфейсов: интерфейс «драйвер-ядро» (Driver Kernel Interface, DKI) и интерфейс «драйвер-устройство» (Driver Device Interface). Интерфейс «драйвер-ядро» должен быть стандартизован в любом случае. Подсистема ввода-вывода может поддерживать несколько различных интерфейсов DKI/DDI, предоставляя специфический интерфейс для устройств определенного класса. К наиболее общим классам относятся блочные устройства, например, диски, и символьные устройства, такие как клавиатура и принтеры. Может существовать класс сетевых адаптеров и др. В большинстве современных ОС определен стандартный интерфейс, который должен поддерживать все блочные драйверы, и второй стандартный интерфейс, поддерживаемый всеми символьными адаптерами. Эти интерфейсы включают наборы процедур, которые могут вызываться остальной операционной системой для обращения к драйверу. К этим процедурам относятся, например, процедуры чтения блока или записи символьной строки.

Кроме того, подсистема ввода-вывода поддерживает большое количество системных функций, которые драйвер может вызывать для выполнения некоторых типовых действий. Например, это операции обмена с регистрами контроллера, ведения буферов промежуточного хранения данных ввода-вывода, взаимодействия с DMA-контроллером и контроллером прерываний и др.

У драйверов устройств есть множество функций:

1. Обработка запросов записи-чтения от программного обеспечения управления устройствами. Постановка запросов в очередь.

2. Проверка входных параметров запросов и обработка ошибок.

3. Инициализация устройства и проверка статуса устройства.

4. Управление энергопотреблением устройства.

5. Регистрация событий в устройстве.

6. Выдача команд устройству и ожидание их выполнения, возможно, в блокированном состоянии, до поступления прерывания от устройства.

7. Проверка правильности завершения операции.

8. Передача запрошенных данных и статуса завершенной операции.

9. Обработка нового запроса при незавершенном предыдущем запросе (для реентерабельных драйверов).

Наиболее очевидная функция состоит в обработке абстрактных запросов чтения и записи независимого от устройств программного обеспечения, расположенного над ними. Но, кроме этого, они должны выполнять еще несколько функций. Например, драйвер должен при необходимости инициализировать устройство. Ему может понадобиться управлять энергопотреблением устройства и регистрацией событий.

Многие драйверы обладают сходной общей структурой. Типичный драйвер начинает работу с проверки входных параметров. Если они не удовлетворяют определенным критериям, драйвер возвращает ошибку. В противном случае драйвер преобразует абстрактные термины в конкретные. Например, дисковый драйвер преобразует линейный номер кластера в номер головки, дорожки и сектора.

Затем драйвер может проверить, не используется ли это устройство в данный момент. Если устройство занято, запрос может быть поставлен в очередь. Если устройство свободно, проверяется статус устройства, чтобы понять, может ли запрос быть обслужен прямо сейчас. Может оказаться необходимым включить устройство или запустить двигатель, прежде чем начнется перенос данных. Как только устройство включено и готово, начинается собственно управление устройством.

Управление устройством подразумевает выдачу ему серии команд. Именно в драйвере определяется последовательность команд в зависимости от того, что должно быть сделано. Определившись с командой, драйвер начинает записывать их в регистры контроллера устройства. После записи каждой команды в контроллер, возможно, будет нужно проверить, принял ли контроллер команду и готов ли принять следующую. Такая последовательность действий продолжается до тех пор, пока контроллеру не будут переданы все команды. Некоторые контроллеры способны принимать связные списки команд, находящихся в памяти. Они сами считывают и выполняют их без дальнейшей помощи ОС.

После того как драйвер передал все команды контроллеру, ситуация может развиваться по двум сценариям. Во многих случаях драйвер устройства должен ждать, пока контроллер не выполнит для него определенную работу, поэтому он блокируется до тех пор, пока прерывание от устройства не разблокирует его. В других случаях операция завершается без задержек и драйверу не нужно блокироваться. В любом случае по завершении выполнения операции драйвер должен проверить, завершилась ли операция без ошибок. Если все в порядке, драйверу, возможно, придется передать данные (например, только что прочитанный блок) независимому от устройств программному обеспечению. Наконец, драйвер возвращает информацию о состоянии для информирования вызывающей программы о статусе завершения операции. Если в очереди находились другие запросы, один из них теперь может быть выбран и запущен. В противном случае драйвер блокируется в ожидании следующего запроса.

Для поддержки процесса разработки драйверов операционной системы выпускается так называемый пакет DDK (Driver Development Kit), представляющий собой набор инструментальных средств-библиотек, компиляторов и отладчиков.

Так как набор потенциально поддерживаемых данных ОС периферийных устройств всегда шире набора устройств, которыми ОС должна управлять при установке на конкретной машине, то ценным свойством ОС является возможность динамически загружать в оперативную память требуемый драйвер (без остановки ОС) и выгружать его, если надобность в драйвере отпала. Такое свойство ОС может существенно сэкономить системную область памяти.

Альтернативой динамической загрузке драйверов при изменении текущей конфигурации внешних устройств компьютера является повторная компиляция кода ядра с требуемым набором драйверов, что создает между всеми компонентами ядра статические связи вместо динамических. Например, таким образом решалась данная проблема в ранних версиях ОС UNIX и Linux. При статистических вызовах между ядром и драйверами структура ОС упрощается, но этот подход требует наличия исходных кодов модулей ОС, доступность которых скорее является исключением (для некоммерческих версий UNIX и Linux). Кроме того, в этом варианте работающую версию ОС надо остановить и заменить новой, что не всегда допустимо в некоторых применениях.

Поэтому поддержка динамической загрузки драйверов является практически обязательным требованием для современных универсальных ОС.

Первоначально термин «драйвер» применялся в достаточно узком смысле – под драйвером понимается программный модуль, который:

- входит в состав ядра ОС, работая в привилегированном режиме;

- непосредственно управляет внешним устройством, взаимодействуя с его контроллером с помощью команд ввода-вывода компьютера;

- обрабатывает прерывания от контроллера устройства;

- предоставляет прикладному программисту удобный логический интерфейс работы с устройством, экранируя от него низкоуровневые детали управления устройством и организации его данных;

- взаимодействует с другими модулями ядра ОС с помощью строго оговоренного интерфейса, описывающего формат передаваемых данных, структуру буферов, способы включения драйвера в состав ОС, способы вызова драйвера, набор общих процедур подсистемы ввода-вывода, которыми драйвер может пользоваться и т.п.

Согласно этому определению, драйвер вместе с контроллером устройства и прикладной программой воплощали идею многослойного подхода к организации программного обеспечения. Контроллер представлял низкий слой управления устройством, выполняющий операции в терминах блоков и агрегатов устройства (например, передвижение головки дисковода, побитную передачу байта по двухпроводному кабелю). Драйвер выполнял более сложные операции, преобразуя данные, адресуемые в терминах номеров цилиндров, головок и секторов диска, в линейную последовательность блоков. В результате прикладная программа работала с данными, преобразованными в достаточно понятную форму, – файлами, таблицами баз данных, текстовыми окнами на мониторе и т.п., не вдаваясь в детали представления этих данных в устройствах ввода-вывода.

В описанной схеме драйверы не делились на слои. Постепенно, по мере развития операционных систем и усложнения структуры подсистемы ввода-вывода, наряду с традиционными драйверами в ОС появились так называемые высокоуровневые драйверы, которые располагаются в общей модели подсистемы ввода-вывода над традиционными драйверами. Появление таких драйверов можно считать развитием идеи многоуровневой организации подсистемы ввода-вывода, когда ее функции декомпозируются между несколькими модулями в соседних слоях иерархии (таких примеров много, например семиуровневая модель сетевых протоколов).

Традиционные драйверы, которые стали называть аппаратными, низкоуровневыми или драйверами устройств, освобождаются от высокоуровневых функций и занимаются только низкоуровневыми операциями. Эти низкоуровневые операции составляют фундамент, на котором можно построить тот или иной набор операций в драйверах более высоких уровней.

При таком подходе повышается гибкость и расширяемость функции по управлению устройством. Например, если различным приложениям необходимо работать с различными логическими модулями одного и того же физического устройства, то для этого в системе достаточно установить несколько драйверов на одном уровне, работающих над одним аппаратным драйвером. Несколько драйверов, управляющих одним устройством, но на разных уровнях, можно рассматривать как один многоуровневый драйвер.

На практике используют от двух до пяти уровней драйверов, поскольку с увеличением числа уровней снижается скорость выполнения операций ввода-вывода.

Высокоуровневые драйверы оформляются по тем же правилам и придерживаются тех же внутренних интерфейсов, что и аппаратные драйверы. Как правило, высокоуровневые драйверы не вызываются по прерываниям, так как взаимодействуют с устройством через посредничество аппаратных драйверов.

В модулях подсистемы ввода-вывода, кроме драйверов, могут присутствовать и другие модули, например, дисковый кэш. Достаточно специфичные функции кэша делают нецелесообразным оформление его в виде драйвера, взаимодействующего с другими модулями ОС только с помощью услуг менеджера ввода-вывода. Другим примером модуля, который чаще всего не оформляется в виде драйвера, является диспетчер окон графического интерфейса. Иногда этот модуль вообще выносится из ядра ОС и реализуется в виде пользовательского интерфейса. Таким образом, был реализован диспетчер окон в Windows NT 3.5 и 3.51, но этот микроядерный подход заметно замедляет графические операции, поэтому в Windows NT 4.0 диспетчер окон и высокоуровневые графические драйверы, а также графическая библиотека GDI были перенесены в пространство ядра.

Аппаратные драйверы после запуска операции ввода-вывода должны своевременно реагировать на завершение контроллером заданного действия путем взаимодействия с системой прерывания. Драйверы более высоких уровней вызываются не по прерываниям, а по инициативе аппаратных драйверов или драйверов вышележащего уровня. Не все процедуры аппаратного драйвера нужно вызывать по прерываниям, поэтому драйвер обычно имеет определенную структуру, в которой выделяется секция обработки прерываний (Interrupt Service Routine, ISR), которая и вызывается от соответствующего устройства диспетчером прерываний.

В унификацию драйверов большой вклад внесла ОС UNIX, в которой все драйверы были разделены на два класса: блок-ориентированные (Block-oriented) и байт-ориентированные (Character-oriented) драйверы. Это более общее деление, чем деление на вертикальные подсистемы. Например, драйверы графических устройств и сетевых устройств относятся к классу байт-ориентированных.

Блок-ориентированные драйверы управляют устройствами прямого доступа, которые хранят информацию в блоках фиксированного размера, каждый из которых имеет свой адрес. Адресуемость блоков приводит к тому, что для дисков, являющихся устройствами прямого доступа, появляется возможность кэширования данных в оперативной памяти. Это обстоятельство значительно влияет на общую организацию ввода-вывода для блок-ориентированных драйверов.

Устройства, с которыми работают байт-ориентированные драйверы, не адресуют данные и не позволяют производить операции поиска данных, они генерируют или потребляют последовательность байта (терминалы, принтеры, сетевые адаптеры и т.п.).

Однако не все устройства, управляемые подсистемой ввода-вывода, можно разделить на блок и байт-ориентированные. Для таких устройств (например, таймер) нужен специфический драйвер.

В свое время ОС UNIX сделала очень важный шаг по унификации операций и структуризации программного обеспечения ввода-вывода. В ОС UNIX все устройства рассматриваются как виртуальные (специальные) файлы, что дает возможность использовать общий набор базовых операций ввода-вывода для любых устройств независимо от их специфики. Подобная идея реализована позже в MS-DOS, где последовательные устройства – монитор, принтер и клавиатура – считаются файлами со специальными именами: con, prn, con.

**112. Опишите подсистему учетных записей.**

Основой системы разграничения доступа в ОС является понятие **учетной записи.** Для каждого зарегистрированного пользователя система создает свою учетную запись.

Учетная запись – это запись в специальной базе данных системы, содержащая информацию о пользователе, а также данные для аутентификации пользователя. Каждый раз при аутентификации пользователя, происходит сравнение введенных им данных с данными из базы, и при совпадении пользователь получает соответствующий доступ к ОС.

Для этой цели используются вопросы идентификации и аутентификации пользователей.

Для каждого пользователя система должна быть в состоянии уникальным образом идентифицировать каждого клиента сети. Существует много способов идентификации, но наиболее распространенный среди них заключается в использовании символьной строки с идентификатором пользователя (User Identification Code – UID).

Процесс аутентификации пользователей – проверки подлинности, заключается в проверке системой того факта, что пользователь действительно является тем, за кого себя выдает. Чаще всего для проведения этой проверки используются пароли. Если соблюдены все правила присвоения паролей, и пользователи тщательно им следуют, пароли становятся довольно эффективным средством аутентификации пользователей.

Создание учетных записей и групп занимает важное место в обеспечении безопасности Windows, поскольку, назначая им права доступа, администратор получает возможность ограничить пользователей в доступе к конфиденциальной информации компьютерной сети, разрешить или запретить им выполнение в сети определенного действия, например архивацию данных или завершение работы компьютера. Обычно право доступа ассоциируется с объектом – файлом или папкой. Оно определяет возможность данного пользователя получить доступ к объекту.

Каждому пользователю, работающему в домене или на одном из его компьютеров, администратор заводит учетную запись. Учетные записи необходимы для осуществления контроля доступа пользователей к ресурсам домена или локальным ресурсам компьютера.

При создании учетной записи ей присваивается уникальный защитный код – Security Identifier или SID, который предоставляет собой число, идентифицирующее учетную запись.

Виды учетных записей.

В MS Windows предусмотрено три типа учетных записей:

1. Локальная учетная запись – позволяет пользователю зарегистрироваться на конкретном компьютере, чтобы получить доступ к его ресурсам. Локальная учетная запись позволяет пользователю войти в систему и получить доступ к ресурсам только того компьютера, на котором создана эта запись. При создании локальной учетной записи Windows создает запись только в базе данных системы защиты этого компьютера, которая называется локальной базой данных безопасности.

То есть локальная учетная запись:

* + предоставляет доступ к ресурсам локального компьютера;
  + создается только на компьютерах, не включенных в домен;
  + содержится в локальной БД безопасности.

1. Учетная запись домена – позволяет пользователям получить доступ к домену и его ресурсам из любого места сети.

В процессе входа в систему пользователь вводит свой пароль и регистрационное имя.

На основе этих сведений Windows опознает пользователя и выделяет ему маркер доступа, который содержит информацию о пользователе и параметрах защиты.

Маркер доступа идентифицирует пользователя для компьютеров, работающих под управлением Windows, к ресурсам которых пользователь хочет получить доступ. Windows создает маркер доступа на время данной сессии.

Доменная учетная запись создается в копии БД каталога Active Directory на контроллере домена.

Этот контроллер реплицирует информацию о новой учетной записи на другие контроллеры домена. Затем все контроллеры в дереве домена могут опознать пользователя в процессе входа в систему.

Учетная запись домена:

-  предоставляет доступ к сетевым ресурсам в домене;

-  предоставляет маркер доступа для идентификации;

-  создается в Active Directory на контроллере домена.

3. Встроенная учетная запись – позволяет выполнять функции администрирования или получать доступ к локальным или сетевым ресурсам.

Windows 2000 и выше автоматически создает встроенные учетные записи. Наиболее часто применяется встроенная учетная запись Administrator

Ранее в версиях Windows до Windows Vista (Windows Server 2008) существовала учетная запись Guest (Гость).

Встроенная учетная запись Administrator (Администратор).

Применяется для управления компьютером в целом. Если компьютер является частью домена, можно использовать ее для конфигурирования домена. Задачи, выполняемые с помощью учетной записи Administrator, включают создание и модификацию учетных записей и групп, управление политикой безопасности, установку принтеров, назначение разрешений учетным записям для доступа к ресурсам.

Удалить учетную запись Administrator нельзя.

Учетная запись Guest (Гость)

Использование встроенной учетной записи Guest (Гость) необходимо для предоставления возможности входа в систему временным пользователям.

Учетная запись Guest по умолчанию отключена в версиях Windows, в которых она присутствовала. Активировать ее стоило только в сетях, не требующих высокой степени защиты, и следовало всегда назначить пароль. Вы могли переименовать эту учетную запись, но не удалить ее.

**Учетные записи в Linux**

Управление учетными записями пользователей в Linux может осуществляться тремя равноценными способами. Во-первых, вы можете использовать инструменты с графическим интерфейсом, предоставляемые вашим дистрибутивом. Внешний вид и принцип работы этих инструментов зависит от используемого вами дистрибутива. В том случае, если вы являетесь неопытным пользователем своей домашней системы Linux, используйте инструмент с графическим интерфейсом, предоставляемый вашим дистрибутивом. Такой подход гарантированно позволит избежать проблем.

Другим вариантом является использование таких инструментов с интерфейсом командной строки, как useradd, usermod, gpasswd, passwd и других. Администраторы серверов с большой вероятностью используют именно эти инструменты, так как они им знакомы, а также поставляются в неизменном виде в составе различных дистрибутивов.

Третий довольно радикальный способ управления учетными записями пользователей заключается в непосредственном редактировании локальных файлов конфигурации с помощью текстового редактора. Не нужно пытаться делать это при работе в системах, находящихся в промышленной эксплуатации, в том случае, если вы не обладаете соответствующими знаниями!

*Файл /etc/passwd*

Локальная база данных учетных записей пользователей в Linux (и в большинстве систем Unix) расположена в файле /etc/passwd.

Данный файл содержит данные в форме таблицы с семью столбцами, разделенными символом двоеточия. В столбцах содержатся имя пользователя, символ x, идентификатор пользователя, идентификатор основной группы пользователя, описание учетной записи пользователя, путь к домашней директории пользователя, а также путь к исполняемому файлу командной оболочки, используемой для входа пользователя в систему.

*Пользователь root*

Учетная запись пользователя root, также называемого суперпользователем, является наиболее привилегированной учетной записью системы Linux. Данный пользователь может делать практически все, включая создание учетных записей других пользователей. Пользователь root всегда имеет идентификатор, равный 0 (вне зависимости от имени учетной записи).

*Утилита useradd*

Добавлять учетные записи пользователей в базу данных с помощью утилиты useradd.

*Файл /etc/default/useradd*

Как в дистрибутиве Red Hat Enterprise Linux, так и в дистрибутивах Debian/Ubuntu имеется файл /etc/default/useradd, который содержит некоторые стандартные параметры пользовательского окружения. Помимо команды cat, вы можете использовать команду useradd -D для ознакомления с содержимым данного файла.

*Утилита userdel*

Удалить учетные записи с помощью утилиты userdel. Параметр -r утилиты userdel позволяет также удалить домашнюю директорию пользователя.

*Утилита usermod*

Модифицировать параметры учетной записи пользователя можно с помощью утилиты usermod.

Создание домашних директорий пользователей.

Простейший способ создания домашней директории пользователя заключается в передаче параметра -m утилите useradd (вполне вероятно, что данный параметр передается по умолчанию в вашем дистрибутиве Linux).

Менее простой способ заключается в самостоятельном создании домашней директории с помощью команды mkdir, что также подразумевает необходимость установки владельца и прав доступа к данной директории с помощью утилит chmod и chown.

Директория /etc/skel/

В случае использовании параметра -m утилиты useradd содержимое директории /etc/skel/ копируется в создаваемую домашнюю директорию пользователя. В директории /etc/skel/ находятся некоторые (обычно скрытые) файлы, которые содержат стандартные параметры профиля пользователя и значения параметров приложений. Таким образом, директория /etc/skel/ выступает в роли шаблона домашней директории и стандартного профиля пользователя.

Удаление домашних директорий пользователей.

В случае использования параметра -r утилиты userdel вы можете быть уверены в том, что домашняя директория пользователя будет удалена вместе с его учетной записью.

Командная оболочка, используемая для входа в систему.

В файле /etc/passwd содержится информация о командной оболочке, используемой для входа пользователя в систему.

Вы можете использовать команду usermod для изменения командной оболочки пользователя.

Утилита chsh.

Пользователи могут изменять используемую для входа в систему командную оболочку с помощью утилиты chsh.