Правительство Санкт-Петербурга
Комитет по науке и высшей школе
Санкт-Петербургское Государственное бюджетное профессиональное образовательное учреждение «Санкт-Петербургский политехнический колледж»

Е.Л. Улыбина

Курс лекций
по дисциплине

Операционные системы

Санкт-Петербург
2018

Курс лекций разработан на основе рабочей программы по дисциплине Операционные системы, предназначенной для реализации среднего профессионального образования по специальности 09.02.03 Программирование в компьютерных системах.
Составитель Улыбина Е.Л.
Издательский отдел СПб ГБПОУ СПбПК, 2018– 82 с.

РАССМОТРЕНО И ОДОБРЕНО
на заседании учебной цикловой
комиссии «Программирование
в компьютерных системах»
СПб ГБПОУ «СПбПК»
09.01.2018 Протокол № 5

Рецензенты:
Еропкин И.В. - преподаватель СПб ГБПОУ «СППбК»

Фомина Н.А. – преподаватель СПб ГБПОУ «ПКГХ»

Курс лекций предназначен для студентов колледжа, обучающихся по специальности 09.02.03 «Программирование в компьютерных системах», а также для студентов других специальностей, желающих более углубленно познакомиться с основными теоретическими вопросами, касающимися общих положений и принципов построения ОС.

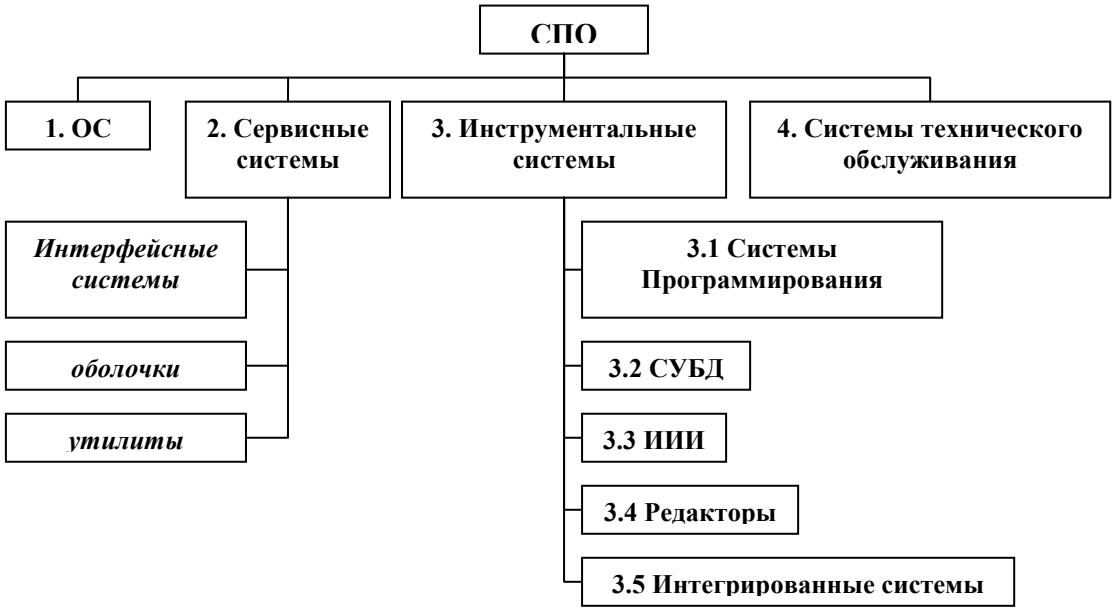
Предложенный курс лекций является кратким изложением теоретической части по дисциплине «Операционные системы» и может быть полезным для самостоятельного изучения, для подготовки к опросам, тестированию по предмету, к практическим занятиям, а также для подготовки к итоговому экзамену по дисциплине ОС.

Изучение теоретического материала темы дисциплины «Операционные системы» сопровождается выполнением практических работ, выполнение которых обязательно. Материал предложенного курса лекций поможет в подготовке к практическим работам. В конце лекций предложен глоссарий основных понятий ОС для лучшего понимания терминов, используемых в вычислительной технике.

Содержание

1. Раздел Общие положения

1.1 Структура системного программного обеспечения (СПО)



1. ***ОС*** – Операционная система – это комплекс программ, обеспечивающий автоматическое выполнение заданий пользователя, управление аппаратурой ***ПЭВМ***, а также планирование и эффективное использование ее ресурсов (необходимый объем *ОЗУ* и *время CPU (МП)*).

Разрядность ***МП*** определяет ***ОС***, которая может быть установлена на данном ПК.

2. Сервисные системы

Интерфейсные системы загружаются поверх ***ОС*** для улучшения пользовательского интерфейса и расширения ее возможностей (графического типа).

Оболочки – упрощают пользовательский интерфейс за счет организации системы меню с использованием функциональных клавиш.

Утилиты – служебные программы, расширяющие программный интерфейс ***ОС,*** предоставляя дополнительные функции.

3. Инструментальные системы – это совокупность программного продукта, обеспечивающего разработку информационно-программного обеспечения ПК.

3.1 Системы программирования – это язык программирования и комплекс программ для создания, отладки и выполнения нового программного продукта, а также реализация диалоговых возможностей на этапе создания.

В систему программирования входит:

1. Компилятор – программа, позволяющая проверять на наличие ошибок исходный модуль программы и переводить его в машинный код.
   Транслятор – это наиболее общее название программы-переводчика в машинные коды.
   Существует 2 типа трансляторов:
   - Наиболее распространенный тип – компилятор, после обработки получается объектный модуль программы, либо выдается сообщение об ошибке по тексту программы.

- Интерпретатор – обрабатывает программный текст построчно либо покомандно, при этом проверяет на ошибки, переводит в двоичный код и сразу выполняет команду.

2. Библиотека стандартных процедур и функций.
3. Компоновщик (linker) – редактор связи объединяет объектный модуль программы с объектными модулями процедур и функций из библиотеки.
4. Отладчик (debugger) – программа, которая выявляет логические ошибки в решении, позволяя пошагово просматривать выполнение программы, с помощью точек останова (breakpoint), а также отслеживать содержимое переменных и регистров процессора.
5. Текстовый редактор, а также сервисные программы для организации интерфейса.

Два подхода создания систем программирования:

1. автономное существование всех программных компонентов.
2. создание среды программирования.

3.2 Система управления базами данных (***СУБД***) – обеспечивает централизованное управление данными, хранящимися в базе данных – это данные, организованные специальным образом, разбитые на поля, с указанием связи этих полей между собой.

Базы данных: FoxPro, Paradox, Access, Visual Basic, Clipper, Delphi.

3.3 Инструментарий искусственного интеллекта. Направление развития: моделирование, поведение роботов, экспертные системы, решение комбинаторных задач, распознавание образов, обработка естественного языка и моделирование диалога, интеллектуальный вопрос на ответные системы.

3.4 Редакторы – это программный продукт, который служит для создания и изменения целевого документа (текстовый, графический)

3.5 Интегрированные системы – это совокупность функционально различных компонентов, способных взаимодействовать между собой путем обмена данными и объединенных единым унифицированным интерфейсом.

4. Системы технического обслуживания – предназначены для тестирования оборудования и исправления неисправностей, используется специалистами аппаратного обеспечения.

1.2 Классификация ОС. Разновидности ОС.

Классификация ОС.

1) по типу ***МП***:

8-ми разрядные ***МП*** (CPU) – CP/M, Intel 8080
16-ти разрядные ***МП*** (286 – AT) – MS-DOS, DR-DOS, Apple-DOS
32-х разрядные ***МП*** (386, 486, Pentium-I, II, III, IV) – OS/2, OS-386, MS Windows 3.11, 95, 98, NT 4.0, Windows-2000, XP, Vista, Unix, Linux.
64-х разрядные ***МП*** - Windows-7, 8,10, Linux-подобные.

2) по способу загрузки:

а) твердые – прошитые в ***ПЗУ*** (8-ми разрядные ***МП***)
б) гибридные (комбинированные) – ядро ***ОС*** прошито в ***ПЗУ***, а драйверы загружаются с магнитного диска (МД).*Драйвер* – программа, управляющая потоком информации от ПУ к центральной части и наоборот.

в) загружаемые (мягкие) – в ***ПЗУ*** находится только программа начальной загрузки ***ОС***, которая позволяет обратиться к стартовому сектору дискеты, винчестера, CD-ROM и считать программу системного загрузчика, которая «знает» все особенности загрузки программ ***ОС***, расположенных на данном диске.

3) по способу обработки заданий (по временному признаку):

- ***ОС*** реального времени (MS-DOS)
- Система с разделением времени – используется для многопользовательских систем, в которых организован мультипрограммный режим.
- Пакетный режим – все задания вначале формируются в очередь запросов, причем обработка заданий происходит без вмешательства пользователя, управлением выполнения заданий занимается оператор или администратор

4) по режиму работы:

однопрограммные *ОС* – в каждый момент времени выполняется только одна задача, второе задание входит в решение только после окончания решения первой (DOS)
мультипрограммные ОС – в процессоре все время разделяется на кванты, в решение могут быть запущены несколько задач, причем каждой из них предоставляется квант времени ***МП*** в соответствии с приоритетом в системе. Выигрыш получается за счет совмещения работы ***МП*** и обращений к внешним устройствам, т.к. ПУ могут работать без вмешательства ***МП***, которых лишь передает управляющий сигнал и данные и освобождается.

Разновидности ОС.

ОС мэйнфреймов (mainframe).

Мэйнфреймы отличаются от ПК своими возможностями ввода-вывода (терабайты данных, тысячи дисков). Представляют собой серверы для крупномасштабных электронно-коммерческих сайтов, а также серверы для транзакций в бизнесе. ***ОС*** ориентирована на множественную обработку заданий ввода-вывода – в основном пакетная обработка. Запросы, небольшие по объему операций, формируются в очередь и система должна отвечать на сотни и тысячи запросов в секунду (***ОС*** 360, ***ОС*** 390)

Серверные ОС.

Работают на серверах и одновременно обслуживают множество пользователей (юзер, user), разделяя между ними программные и аппаратные ресурсы (Unix, Windows NT-family, Linux).

Многопроцессорные ОС.

Для увеличения производительности ПК в одной системе соединяются несколько центральных процессоров. В зависимости от вида соединения ***МП*** и разделения работы системы называются параллельными компьютерами, мультикомпьютерами и многопроцессорными системами. Для них требуется специальная ***ОС***, предоставляющая собой варианты серверных ***ОС*** с расширенными возможностями связи.

<u>ОС для ПК</u> (Windows 95, 98, XP, 7,8, 10, MacOS, Linux)

I. ОС реального времени используется в системах управления производством, т.к. ПК собирают данные о промышленном процессе и используют их для управления роботами, конвейерами. Лимитировано время реакции ***ОС*** на запросы. К этой категории ***ОС*** попадают и ***ОС*** мультимедийной системы: разновидности QNX, VxWorks.

II. *Встроенные ОС* используются для Карманных ПК (КПК) и встроенных систем, имеют те же характеристики, что и системы реального времени, но ограничены мощностью (Windows CE, PalmOS).

III. *ОС для смарт-карт* (Java ориентированные) зашиты в ***ПЗУ***, имеют ограниченный набор операций и содержат интерпретатор виртуальной машины JAVA (Java Virtual Machine).

1.3 Критерии оценки ОС

При сравнительном рассмотрении различных ОС в целом или их отдельных подсистем возникает вечный вопрос – какая из них лучше и почему, какая архитектура системы предпочтительнее, какой из алгоритмов эффективнее, какая структура данных удобнее и т.п.

Очень редко можно дать однозначный ответ на подобные вопросы, если речь идет о практически используемых системах. Система или ее часть, которая хуже других систем во всех отношениях, просто не имела бы права на существование. На самом деле имеет место типичная многокритериальная задача: имеется несколько важных критериев качества, и система, опережающая прочие по одному критерию, обычно уступает по другому. Сравнительная важность критериев зависит от назначения системы и условий ее работы.

Надежность

Этот критерий вообще принято считать самым важным при оценке программного обеспечения, и в отношении ОС его действительно принимают во внимание в первую очередь.

Что понимается под надежностью ОС?

Прежде всего, ее ***живучесть***, т.е. способность сохранять хотя бы минимальную работоспособность в условиях аппаратных сбоев и программных ошибок. Высокая живучесть особенно важна для ОС компьютеров, встроенных в аппаратуру, когда вмешательство человека затруднено, а отказ компьютерной системы может иметь тяжелые последствия.

Во-вторых, способность, как минимум, диагностировать, а как максимум, компенсировать хотя бы некоторые типы аппаратных сбоев. Для этого обычно вводится избыточность хранения наиболее важных данных системы.

В-третьих, ОС не должна содержать собственных (внутренних) ошибок. Это требование редко бывает выполнимо в полном объеме (программисты давно сумели доказать своим заказчикам, что в любой большой программе всегда есть ошибки, и это в порядке вещей), однако следует хотя бы добиться, чтобы основные, часто используемые или наиболее ответственные части ОС были свободны от ошибок.

Наконец, к надежности системы следует отнести ее способность противодействовать явно неразумным действиям пользователя. Обычный пользователь должен иметь доступ только к тем возможностям системы, которые необходимы для его работы. Если же пользователь, даже действуя в рамках своих полномочий, пытается сделать что-то очень странное (например, отформатировать системный диск), то самое малое, что должна сделать ОС, это переспросить пользователя, уверен ли он в правильности своих действий.

Эффективность

Как известно, эффективность любой программы определяется двумя группами показателей, которые можно обобщенно назвать «время» и «память». При разработке системы приходится принимать много непростых решений, связанных с оптимальным балансом этих показателей.

Важнейшим показателем временно́й эффективности является ***производительность*** системы, т.е. усредненное количество полезной вычислительной работы, выполняемой в

единицу времени. С другой стороны, для диалоговых ОС не менее важно ***время реакции*** системы на действия пользователя. Эти показатели могут в некоторой степени противоречить друг другу. Например, в системах разделения времени увеличение кванта времени повышает производительность (за счет сокращения числа переключений процессов), но ухудшает время реакции.

В программировании известна аксиома: выигрыш во времени достигается за счет проигрыша в памяти, и наоборот. Это в полной мере относится к ОС, разработчикам которых постоянно приходится искать баланс между затратами времени и памяти.

Забота от эффективности долгое время стояла не первом месте при разработке программного обеспечения, и особенно ОС. К сожалению, оборотной стороной стремительного увеличения мощности компьютеров стало ослабление интереса к эффективности программ. В настоящее время эффективность является первостепенным требованием разве что в отношении систем реального времени.

Удобство

Этот критерий наиболее субъективен. Можно предложить, например, такой подход: система или ее часть удобна, если она позволяет легко и просто решать те задачи, которые встречаются наиболее часто, но в то же время содержит средства для решения широкого круга менее стандартных задач (пусть даже эти средства не столь просты). Пример: такое частое действие, как копирование файла, должно выполняться при помощи одной простой команды или легкого движения мыши; в то же время для изменения разделов диска не грех почитать руководство, поскольку это может понадобиться даже не каждый год.

Разработчики каждой ОС имеют собственные представления об удобстве, и каждая ОС имеет своих приверженцев, считающих именно ее идеалом удобства.

Масштабируемость

Довольно странный термин «масштабируемость» (scalability) означает возможность настройки системы для использования в разных вариантах, в зависимости от мощности вычислительной системы, от набора конкретных периферийных устройств, от роли, которую играет конкретный компьютер (сервер, рабочая станция или изолированный компьютер) от назначения компьютера (домашний, офисный, исследовательский и т.п.).

Гарантией масштабируемости служит продуманная модульная структура системы, позволяющая в ходе установки системы собирать и настраивать нужную конфигурацию.

Возможен и другой подход, когда под общим названием объединяются, по сути, разные системы, обеспечивающие в разумных пределах программную совместимость. Примером могут служить версии Windows NT/2000/XP, Windows 95/98 и Windows CE.

В некоторых случаях фирмы, производящие программное обеспечение, искусственно отключают в более дешевых версиях системы те возможности, которые на самом деле реализованы, но становятся доступны, только если пользователь покупает лицензию на более дорогую версию. Но это уже вопрос, связанный не с технической стороной дела, а с маркетинговой политикой.

Способность к развитию

Чтобы сложная программа имела шансы просуществовать долго, в нее изначально должны быть заложены возможности для будущего развития.

Одним из главных условий способности системы к развитию является хорошо продуманная модульная структура, в которой четко определены функции каждого модуля и его взаимосвязи с другими модулями. При этом создается возможность совершенствования отдельных модулей с минимальным риском вызвать нежелательные последствия для других частей системы.

Важным требованием к развитию ОС является ***совместимость версий снизу вверх,*** означающая возможность безболезненного перехода от старой версии к новой, без потери ранее наработанных прикладных программ и без необходимости резкой смены всех навыков пользователя. Обратная совместимость – сверху вниз – как правило, не гарантируется, поскольку в ходе развития система приобретает новые возможности, не

реализованные в старых версиях. Программа из Windows 3.1 будет нормально работать и в Windows XP; наоборот – вряд ли.

Фирмы-производители ОС прилагают максимум усилий для обеспечения совместимости снизу вверх, чтобы не отпугнуть пользователей. Но при этом фирмы стараются в каждую новую версию заложить какую-нибудь новую конфетку, которая побудила бы пользователей как можно скорее купить ее.

Совместимость версий – благо для пользователя, однако на практике она часто приводит к консервации давно отживших свой век особенностей или же просто неудачных решений, принятых в ранней версии системы. В документации подобные архаизмы помечаются как «устаревшие» (obsolete), но полного отказа от них, как правило, не происходит (а вдруг где-то еще работает прикладная программа, написанная двадцать лет назад с использованием именно этих средств?).

Как правило, наиболее консервативной стороной любой ОС являются не алгоритмы, а структуры системных данных, поэтому дальновидные разработчики заранее строят структуры «на вырост»: закладывают в них резервные поля, используют переменные вместо некоторых констант, устанавливают количественные ограничения с большим запасом и т.п.

Мобильность

Под ***мобильностью*** (portability) понимается возможность переноса программы (в данном случае ОС) на другую аппаратную платформу, т.е. на другой тип процессора и другую архитектуру компьютера. Здесь имеется в виду перенос с умеренными трудозатратами, не требующий полной переработки системы.

Свойство мобильности не столь однозначно положительно, как может показаться. Чтобы программа была мобильна, при ее разработке следует отказаться от глубокого использования особенностей конкретной архитектуры (таких, как количество и функциональные возможности регистров процессора, нестандартные команды и т.п.). Мобильная программа должна быть написана на языке достаточно высокого уровня (часто используется язык С), который можно реализовать на компьютерах любой архитектуры. Платой за мобильность всегда является некоторая потеря эффективности, поэтому немобильные системы распространены достаточно широко.

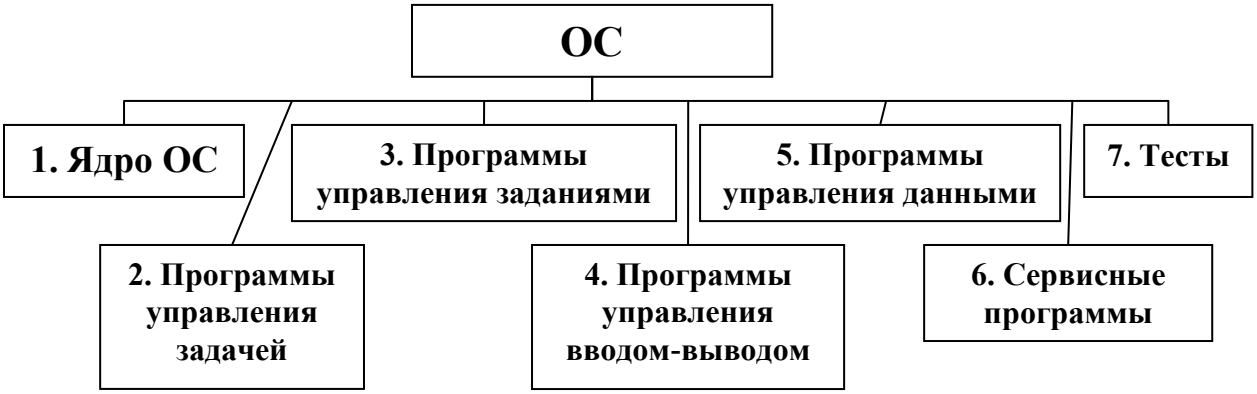
С другой стороны, история системного программирования усеяна останками замечательных, эффективных и удобных, но немобильных ОС, которые вымерли вместе с процессорами, для которых они предназначались. В то же время мобильная система UNIX продолжает процветать четвертый десяток лет, намного пережив те компьютеры, для которых она первоначально создавалась. Примерно 5-10% исходных текстов UNIX написаны на языка ассемблера и должны переписываться заново при переносе на новую архитектуру. Остальная часть системы написана на С и практически не требует изменений при переносе.

Некоторым компромиссом являются ***многоплатформенные*** ОС (например, Windows NT), изначально спроектированные для использования на нескольких аппаратных платформах, но не гарантирующие возможность переноса на новые, не предусмотренные заранее архитектуры.

1.4 Функции любой ОС.

1) выделение ресурсов и управление ими (время ***МП*** и объем ***ОЗУ*** – аппаратные ресурсы) и программные ресурсы (драйверы и необходимые системные обработчики)
2) загрузка программ в ОП
3) пересылка информации в устройствах связи
4) управление операциями ввода-вывода (драйверы)
5) обслуживание системы прерываний. ***Прерывание*** – это ситуация, вызывающая приостановку выполнения текущей команды ***МП*** и переключение на системную обработку данной ситуации
6) управление и поддержка файловой системы
7) конфигурирование технических средств – настройка ***ОС*** на конкретные устройства, которые могут быть использованы самой системой и пользователем (user)
8) Обработка командного языка
9) Защита и учет использования ресурсов
10) Диспетчеризация процессов вычислительной системы
11) Распределение памяти внешних Запоминающих Устройств (ЗУ)

1.5 Структура ОС



1. Ядро (супервизор, режим ядра – это особый режим при выполнении команд ***МП***, который позволяет напрямую обращаться к устройствам ПЭВМ и в любой раздел ОП) включает в себя драйверы стандартных ПУ и обработчики системных вызовов.
   Часто зашито в ***ПЗУ*** (Basic Input-Output System – BIOS – базовая система ввода-вывода).

2. Программы управления задачей – обеспечивают загрузку программных кодов в ***ОЗУ***, запуск на выполнение, выделение раздела ***ОЗУ***, а так же очищение раздела памяти из ***ОЗУ*** (выгрузка) после выполнения программы, либо оставляет программу резидентной в ***ОЗУ***.

3. Программы управления заданиями – обеспечивают мультипрограммный режим с использованием программы-планировщика заданий, распределяет разделы памяти (***ОЗУ***) между задачами, обеспечивая защищенность и безопасность работы каждой из них.

4. Программы управления вводом-выводом – обеспечивают операции ввода-вывода (за счет драйверов), обработку очередей запросов на ввод-вывод и распределение этих запросов между устройствами и наборами данных.

5. Программы управления данными – обеспечивают методы доступа к данным библиотекам, каталогам (связаны со средствами управления задачей), а также передают в ***ОЗУ*** и к устройствам ввода-вывода.

6. Сервисные программы – расширяют возможности ***ОС*** (утилиты) и пользовательского интерфейса.

7. Тесты – для диагностики и тестирования устройств и программ.

2 Раздел Общие принципы построения ОС

2.1 Машинно-зависимые свойства ОС

2.1.1 Архитектурные особенности модели микропроцессорной системы

Микропроцессор

Первые процессоры появились на самой заре зарождения компьютерных технологий. А бурное развитие микрокомпьютерной техники во многом являлось следствием появления первых *микропроцессоров.* Если раньше все необходимые элементы CPU были расположены на различных электронных схемах, то в микропроцессорах они впервые были объединены на одном-единственном кристалле. В дальнейшем под термином «процессор» мы будем иметь в виду именно микропроцессоры, поскольку эти слова давно превратились в синонимы.

Микропроцессор i4004 — прадедушка сегодняшних CPU

Одним из первых микропроцессоров был четырехразрядный процессор фирмы Intel i4004. Он имел смехотворные по нынешним временам характеристики, но для своего времени – начала 1970-х гг., его появление представляло собой настоящий технологический прорыв. Как можно догадаться из его обозначения, он был четырехразрядным и имел тактовую частоту около 0,1 МГц. И именно его прямой потомок, процессор i8088, был выбран фирмой IBM в качестве «мозга» первого персонального компьютера фирмы IBM PC.

Процессор i8088 использовавшийся в первом персональном компьютере фирмы IBM

Шли годы, характеристики CPU становились все более серьезными и внушительными, и, как следствие, становились все более солидными характеристики персональных компьютеров. Значительной вехой в развитии микропроцессоров стал i80386. Это был первый полностью 32-разрядный CPU, который мог адресовать к 4 ГБ оперативной памяти, в то время как большинство его предшественников могло работать максимум с

640 КБ ОЗУ. Подобная разрядность микропроцессоров настольных компьютеров продержалась довольно долго, почти два десятилетия. В середине 80-х объем ОЗУ в 4 ГБ казался фантастически огромным, но сейчас его можно считать небольшим для серьезного компьютера.

i80386 — первый полностью 32-разрядный CPU

Следующий микропроцессор компании Intel, 486DX, замечателен тем, что в нем впервые появился внутренний кэш – внутренняя оперативная память микропроцессора. Кроме того, в нем было применено много других усовершенствований, которые во многом определили дальнейшую эволюцию микропроцессоров. То же самое можно сказать и про следующий процессор компании Intel, Pentium.

Intel 486DX — первый процессор с внутренним кэшем

Процессор компании Intel — Pentium

Вместе с CPU Pentium 4 в ряду технологий, использующихся в микропроцессорах, появилась технология Hyper Threading. А процессоры Opteron от фирмы AMD и Pentium D от Intel открыли современную эпоху эволюции CPU, эпоху процессоров, имеющих несколько ядер. Сейчас на рынке представлено много CPU от различных производителей, но главными производителями до сих пор остаются две компании – Intel и AMD, причем на долю первой приходится более 80% рынка.

CPU Opteron от фирмы AMD и Pentium D от Intel

2.2 Устройство CPU

Любой CPU имеет вычислительное ядро (иногда их бывает несколько), а также кэш, то есть собственную оперативную память. Кэш обычно имеет два уровня – первый и второй (внутренний и внешний). Внутренний имеет меньший объем, но обладает большим быстродействием по сравнению с внешним. Емкость кэша второго уровня современных CPU составляет несколько мегабайт – больше, чем оперативная память первых персональных компьютеров!

В ядре CPU находится несколько функциональных блоков – блок управления, блок выборки инструкций, блок вычислений с плавающей точкой, блок целочисленных вычислений, и.т.д. Также в ядре располагаются главные регистры processor-a, в которых находятся обрабатываемые в определенный момент данные. В классической схеме микропроцессора архитектуры x86 этих регистров всего 16.

На сегодняшний день наибольшее распространение получили две основные разновидности процессоров – CISC (Complex Instruction Set Computing) и RISC (Reduced Instruction Set Computing). В CISC-процессорах мало внутренних регистров, но они поддерживают большой набор команд. В RISC-процессорах регистров много, зато набор команд ограничен. Традиционно микропроцессоры для персональных компьютеров архитектуры Intel x86 принадлежали к классу CISC-процессоров, однако в настоящее время большинство микропроцессоров представляют собой гибрид этих двух архитектур.

Если рассмотреть CPU на аппаратном уровне, то он является, по сути, огромной микросхемой, расположенной на цельном кристалле кремния, в которой содержатся миллионы, а то и миллиарды транзисторов. Чем меньше размеры транзисторов, тем больше их содержится на единицу площади CPU, и тем больше его вычислительная мощность. Кроме того, от размеров транзисторов зависит энерговыделение и энергопотребление процессора — чем меньше их размер, тем эти характеристики процессора меньше. Этот фактор немаловажен, так как CPU является наиболее энергоемким устройством современного ПК. Поэтому проблема уменьшения нагрева процессора входит в число самых важных, стоящих перед разработчиками ПК и самих процессоров.

Отдельно стоит сказать о корпусе, в котором находится CPU. Обычно материалом корпуса процессора служит керамика или пластик. Первоначально процессоры намертво впаивались в системную плату, сейчас же большинство вставляются в специальные гнезда – сокеты. Такой подход заметно упростил модернизацию системы пользователем – достаточно вставить в разъем другой CPU, поддерживаемый данной системной платой, и вы получите более мощный компьютер.

Сокет современного процессора

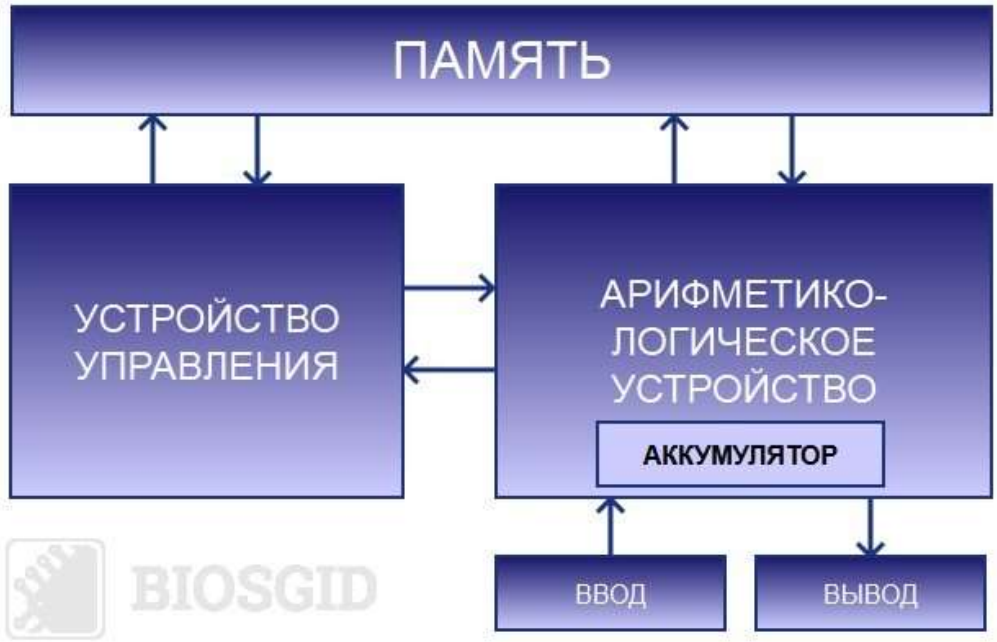
С другими устройствами процессор связан при помощи специальных каналов связи (шин) – шины памяти, шины данных и шины адреса. Разрядность последней очень важна, поскольку от этого параметра зависит объем доступной CPU, а значит, и программам, оперативной памяти.

2.3 Принцип работы МП

МП включает в себя:

1) АЛУ
2) УУ
3) Регистры внутренней памяти – это устройство для запоминания определенного кода (32 и 64 битные)
4) КЭШ (cache) – это стековая память для увеличения быстродействия (128kb, 256kb, 512kb и т.д.)

Для обработки данных управляющее устройство CPU получает из оперативной памяти или кэша процессора сами данные, а также команды, которые описывают процесс обработки данных. Данные помещаются во внутренние регистры микропроцессора, и над ними производятся операции при помощи арифметико-логического устройства в соответствии с поступившими командами.



Принцип работы процессора

Работу CPU синхронизируют так называемые тактовые сигналы. Наверняка каждому пользователю известно понятие тактовой частоты, которая отражает количество тактов работы процессора за секунду. Это значение во многом определят характеристики процессора. Тем не менее, производительность компьютера далеко не всегда пропорциональна его тактовой частоте. И дело тут не только в наличии у современных CPU нескольких ядер, а и в том, что разные процессоры имеют разную архитектуру и, как следствие, могут выполнять разное количество операций за секунду. Современные CPU могут выполнять несколько операций за один такт, тогда как у первых микропроцессоров на одну операцию, наоборот, могло уходить несколько тактов.

CPU архитектуры x86 исторически поддерживают следующие режимы работы процессора:

1. Реальный
2. Защищенный
3. Виртуальный
4. Режим супервизора

Этапы цикла выполнения:

1. Процессор выставляет число, хранящееся в регистре счётчика команд, на шину адреса и отдаёт памяти команду чтения.
2. Выставленное число является для памяти адресом; память, получив адрес и команду чтения, выставляет содержимое, хранящееся по этому адресу, на шину данных и сообщает о готовности.
3. Процессор получает число с шины данных, интерпретирует его как команду (машинную инструкцию) из своей системы команд и исполняет её.
4. Если последняя команда не является командой перехода, процессор увеличивает на единицу (в предположении, что длина каждой команды равна единице) число, хранящееся в счётчике команд; в результате там образуется адрес следующей команды.

Данный цикл выполняется неизменно, и именно он называется *процессом* (откуда и произошло название устройства).

Во время процесса процессор считывает последовательность команд, содержащихся в памяти, и исполняет их. Такая последовательность команд называется программой и представляет алгоритм работы процессора. Очерёдность считывания команд изменяется в случае, если процессор считывает команду перехода, — тогда адрес следующей команды может оказаться другим. Другим примером изменения процесса может служить случай получения команды останова или переключение в режим обработки прерывания.

Команды центрального процессора являются самым нижним уровнем управления компьютером, поэтому выполнение каждой команды неизбежно и безусловно. Не производится никакой проверки на допустимость выполняемых действий, в частности, не проверяется возможная потеря ценных данных. Чтобы компьютер выполнял только допустимые действия, команды должны быть соответствующим образом организованы в виде необходимой программы.

Скорость перехода от одного этапа цикла к другому определяется тактовым генератором. Тактовый генератор вырабатывает импульсы, служащие ритмом для центрального процессора. Частота тактовых импульсов называется тактовой частотой.

Реальный режим работы был единственным режимом, в котором работали все CPU до i80386. В этом режиме processor мог адресовать лишь 640 КБ ОЗУ. В результате появления защищенного режима процессор получил возможность работать с большими объемами оперативной памяти. Также существует разновидность защищенного режима – виртуальный режим, предназначенный для совместимости со старыми программами, написанными для процессоров 8086.

Режимы работы процессора также включают режим супервизора, который используется при работе в современных операционных системах. В этом режиме программный код имеет неограниченный доступ ко всем системным ресурсам.

***МП* выполняет команды следующим образом:**

1) осуществляет выборку команды из ***ОЗУ*** по адресу, который записан в счетчике адреса команд. При первоначальной загрузке программного кода в ***ОЗУ*** в счетчик команд записывается начальный адрес первой команды, таким образом, ***МП*** «знает» по какому адресу выбирать текущую команду.
2) Выбранная команда декодируется; в регистры внутренней памяти переписываются из ***ОЗУ*** переменные – операнды.
3) Команда исполняется ***МП***, результат записывается в регистр временных результатов. Счетчик команд корректируется на длину (в байтах) команды и таким образом получается адрес следующей команды, к которой ***МП*** вновь обращается за выборкой

.

МП содержит регистр ***PSW*** – слово состояния процессора, в данный регистр записываются биты кода состояния, которые задают приоритет ***МП*** и режим (пользователя или ядра), длину команды (в байтах), которая выполняется в ***МП*** и другую служебную информацию. Пользовательские программы могут читать информацию из ***PSW*** целиком, но писать только в отдельные поля. Режим пользователя ограничивает область памяти (адреса в которые можно записывать информацию, а так же обращения к ПУ).
Рассмотренный выше процесс выполнения команды характеризует простую модель обработки. Современные ***МП*** обладают возможностями выполнения нескольких команд одновременно, т.е. во время выполнения n-ой команды ***МП*** может декодировать команду n+1 и выбирать (считывать) команду n+2. Подобная ситуация называется конвейером, но при конвейерной модели усложняется создание компиляторов и системных программ.
Более передовой моделью является супер скалярный центральный процессор.

2.3.1 *Конвейерная архитектура*

Конвейерная архитектура (англ. *pipelining*) была введена в центральный процессор с целью повышения быстродействия. Обычно для выполнения каждой команды требуется осуществить некоторое количество однотипных операций, например: выборка команды из ОЗУ, дешифровка команды, адресация операнда в ОЗУ, выборка операнда из ОЗУ, выполнение команды, запись результата в ОЗУ. Каждую из этих операций сопоставляют одной ступени конвейера. Например, конвейер микропроцессора с архитектурой MIPS-I содержит четыре стадии:

- получение и декодирование инструкции,
- адресация и выборка операнда из ОЗУ,
- выполнение арифметических операций,
- сохранение результата операции.

После освобождения k-й ступени конвейера она сразу приступает к работе над следующей командой. Если предположить, что каждая ступень конвейера тратит единицу времени на свою работу, то выполнение команды на конвейере длиной в nступеней займёт nединиц времени, однако в самом оптимистичном случае результат выполнения каждой следующей команды будет получаться через каждую единицу времени.

Действительно, при отсутствии конвейера выполнение команды займёт nединиц времени (так как для выполнения команды по-прежнему необходимо выполнять выборку, дешифровку и т. д.), и для исполнения mкоманд понадобится $n \cdot m$единиц времени; при использовании конвейера (в самом оптимистичном случае) для выполнения mкоманд понадобится всего лишь $n + m$единиц времени.

Факторы, снижающие эффективность конвейера:

1. Простой конвейера, когда некоторые ступени не используются (например, адресация и выборка операнда из ОЗУ не нужны, если команда работает с регистрами).
2. Ожидание: если следующая команда использует результат предыдущей, то последняя не может начать выполняться до выполнения первой (это преодолевается при использовании внеочередного выполнения команд — out-of-order execution).
3. Очистка конвейера при попадании в него команды перехода (эту проблему удаётся сгладить, используя предсказание переходов).

Некоторые современные процессоры имеют более 30 ступеней в конвейере, что повышает производительность процессора, но, однако, приводит к увеличению длительности простоя (например, в случае ошибки в предсказании условного перехода). Не существует единого мнения по поводу оптимальной длины конвейера: различные программы могут иметь существенно различные требования.

Память. Структура памяти.

ОП (Основная память) состоит:

1) ***ПЗУ*** (ROM) – только для чтения, хранит тестовые программы диагностики устройств, программу начальной загрузки ***ОС***, которая считывает из стартового сектора системного диска системный загрузчик, а так же в ***ПЗУ*** находятся драйвера стандартных ПУ и обработчик прерываний.

Flash-***ОЗУ*** - является энергозависимой памятью, но позволяет переписывать информацию, тем самым исправлять ошибки, допущенные в ***ПЗУ***.

CMOS-память – используется для хранения текущей даты и времени.

2) Энергозависимая память – ***ОЗУ*** – главная рабочая область ЗУ, которая хранит программы и данные на сеанс работы.

ОЗУ – адресное пространство, которое позволяет адресовать любой байт кода (RAM). При загрузке программного кода в ***ОЗУ*** ему назначается базовый адрес (индекс). Все команды программного кода имеют относительную адресацию от 0. Т.о. если базовый адрес равен 10000, а код программы составляет 10000 байт, то предельный адрес будет равен 20000. При обработке команд ***МП*** проверяет, не является ли адрес текущей команды больше предельного, если нет – обращается к следующей.

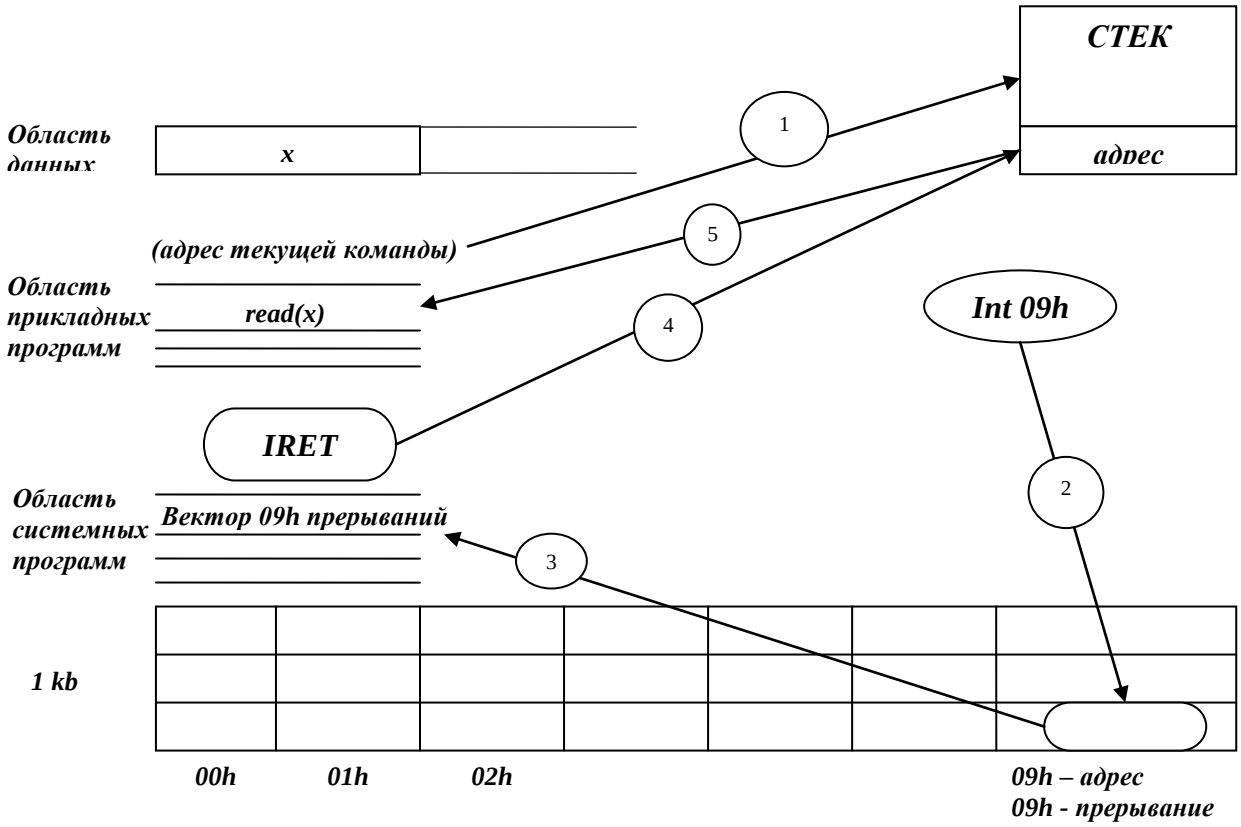
При мультипрограммном режиме необходимо обеспечить защиту данных, а так же перемещение программ в памяти. Этим занимается устройство управления памятью или диспетчер памяти, он находится в схеме микропроцессора, либо между ***МП*** и памятью. Защита данных в памяти обеспечивается таким образом, чтобы базовые адреса программных кодов не перекрывались. Данные программы могут находиться отдельно от программного кода, но диспетчер памяти отслеживает все адреса, относящиеся к данной программе. При переключении на другую задачу адреса прежней программы записываются в стек, а регистры ***МП*** обновляются в соответствии с новыми адресами другой программы.

2.3.2 Понятия прерывания. Система прерываний. Виды прерываний.

Система прерываний (Interrupt, Int) – совокупность аппаратных и программных средств по обработке прерываний.

Прерывание (Int) – возникает в ***МП*** при возникновении какой-либо ситуации, требующей реакции ***ОС***. Каждое прерывание идентифицируется своим номером и каждое из них обрабатывается специальной программой (драйвером прерываний). Различают аппаратные и программные прерывания.

00h - адрес обработчика прерываний
01h вектор прерываний 0000:0000
02h
...
09h



С каждым номером прерывания связана программа обработки данного прерывания, адрес первой команды обработчика – вектора прерываний – хранится в таблице векторов прерываний (IDT – Interrupt Description Table – 1 kb). Под каждый адрес отводится 4 байта: 0000:0004 – 0000:0008 и того всего может быть предусмотрено 256 прерываний.
Прерывания могут быть маскируемыми и немаскируемыми (выполняются всегда). В PSW имеются разряды, в которые можно занести код маски системы, при этом могут быть проигнорированы некоторые ситуации, которые бы вызвали прерывание.
Описание схемы обработки прерываний.

1) Если при выполнении очередной команды ***МП*** встретилась команда, например, READ(***fp***) - читать из файла – то возникнет прерывание. В СТЕК записывается адрес команды, которую ***МП*** должен выполнить следующей, содержимое текущего регистра PSW, а также адрес переменной ***(x)*** и указатель на файл ***fp*** из которого должны быть

прочитаны данные. Затем оператору READ выставляется прерывание с определенным номером. По этому номеру должен быть подключен обработчик системного вызова.

2) ***МП*** обращается в таблицу векторов прерываний и в соответствии с номером прерывания (умноженная на 4 байта) находит соответствующий вектор этого прерывания – адрес первой команды обработчика
3) С помощью специальной программы обслуживания прерывания ISR, ***МП*** переходит в режим ядра и обращается к системному обработчику данного прерывания драйвера устройства. Данная программа из Стека считывает всю необходимую информацию для выполнения указанной операции считывания из файла.
4) Последней командой любого обработчика прерываний является команда IRET (возврат). По данной команде содержимое Стека последовательно переписывается в регистры ***МП***, т.е. восстанавливается содержимое PSW и счетчика адреса команды, PC на прерванной команде.
5) После восстановления содержимого регистров ***МП*** продолжается обычный ход выполнения программы пользователя, при этом ***МП*** опять переходит в режим пользователя.

Обычно ***МП*** может одновременно иметь несколько прерываний: выполнение программы, сигнал от ПУ и т.д. В этом случае ***МП*** выбирает более приоритетное прерывание, а остальное блокирует на некоторое время.

Классы прерываний

1) Программное прерывание - генерируется в результате выполнения команд. Такими ситуациями могут быть – арифметическое переполнение регистров, деление на 0, выполнение некорректной команды, ссылка на область памяти, доступ к которой пользователю запрещен.
2) Прерывание по таймеру - генерируется таймером ***МП***. Данное прерывание позволяет ***ОС*** выполнять некоторые свои функции периодически через заданные промежутки времени.
3) Прерывание ввода-вывода - генерируется контроллером ввода-вывода. Сигнализирует о нормальном завершении операции или о наличии ошибок.
4) Аппаратное прерывание - генерируется при возникновении аварийных ситуаций – падение напряжения в сети, отсутствие четности в памяти.

Если нет аварийной ситуации, то более приоритетными являются программные прерывания, затем прерывания по таймеру или от другого ПК, менее приоритетные прерывания ввода-вывода.

При обработке прерываний возможно два подхода:

1. при получении нескольких прерываний обрабатывать текущее, а остальное запретить, при этом ***МП*** может и должен игнорировать любой новый сигнал прерывания. Все прерывания обрабатываются в строго последовательном порядке.
2. при втором подходе учитывается приоритет прерывания, что позволяет приостановить обработку прерывания с более низким приоритетом (например, устройству ввода-вывода принтера, диску и коммуникационной линии присвоены приоритеты 2, 4, 5) Если в некоторый момент времени ***МП***, обрабатывая прерывания от принтера, получил сигнал от сканера, то ***МП*** приступит к его обработке, при этом вся информация о состоянии принтера будет сохранена до окончания обработки текущего прерывания. Этим занимается программа ISR.

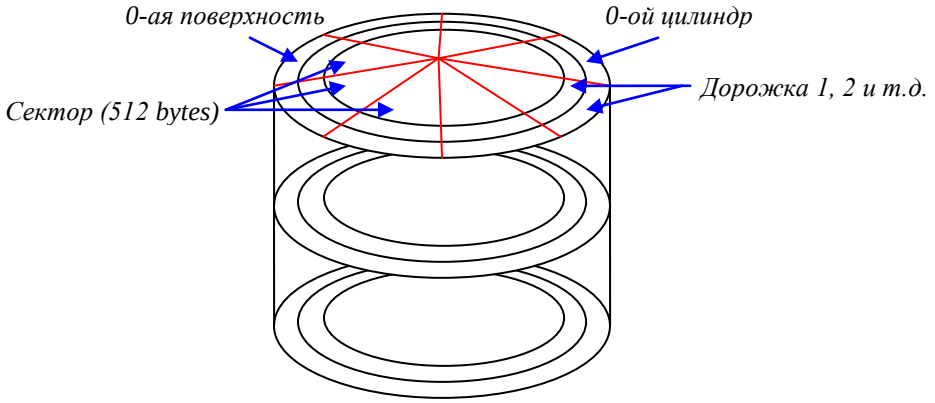
Для поддержки аппаратных прерываний имеется контроллер прерываний – микросхема – IRQ, которая обеспечивает связь с помощью 15 каналов прерываний. Номер прерывания, который присваивается каждому каналу прерываний, определяет ту программу (системный обработчик), которой необходимо передать управление после того, как ***МП*** принял сигнал прерывания. Таким образом, с помощью системы прерывания обеспечивается выполнение всех операций ввода-вывода путем обращения ***ОС*** к драйверу соответствующего ПУ.

2.3.3 Организация ввода-вывода (input-output).

Организация операций IO одна из важнейших функций ОС и обеспечивается комплексом аппаратных и программных средств. Устройства IO состоят из 2-х частей – из контроллера и из самого ПУ.

Контроллер – микросхема или набор микросхем на вставляемой в разъем плате и физически управляющим устройством.

Он принимает команды ОС (например, прочесть сектор 11190 с диска 2), контроллер должен преобразовать линейный номер сектора в номер цилиндра, номер сектора и головки считывающего устройства.



Физически адрес сектора складывается из: *адреса цилиндра, рабочей поверхности* – головки считывающего устройства, *и сектора.*

1 сектор равен 512 bytes (поддерживается BIOS). За один цикл считывается 1 сектор и обмен информации между ОЗУ и МД осуществляется секторами или блоками по 512 байт.

По типу обмена данными между ПУ и ОЗУ различают:

5) блочные ПУ – винчестер, CD-ROM, floppy диски
6) Символьные ПУ – обмен информацией посимвольно (принтер, графопостроитель, монитор, клавиатура, сканер, мышь)

Порты ввода/вывода.

Для организации взаимодействия с контроллером может быть использовано следующее аппаратное средство: COM - **порт.**

В этом случае контроллер или объединенные сетью контроллеры подключаются по протоколам **RS-232**, **RS-422**, **RS-485.**

Портами ввода-вывода (port input/output), вот два их определения из книжки:

1)“**порт ввода/вывода** – это 8,16 или 32-разрядный аппаратный регистр (не путать с регистрами процессора! - прим. от меня), имеющий определенное количество адресов в адресном пространстве ввода-вывода”

2) **порты ввода/вывода** - это пункты системного интерфейса ПК, через которые МикроПроцессоры (МП) обмениваются информацией с другими устройствами.

Всего у МП портов ввода/вывода может быть **65536**. Каждый порт ввода/вывода имеет свой адрес.

Адрес порта ввода/вывода – это **номер порта**, соответствующего адресу ячейки памяти, являющегося частью устройства ввода/вывода, использующего этот порт, а не частью основной памяти компьютера.

“ **Номер порта (number of port)**” - ну как объяснить? Вот десять портов (предположим) - так что им, каждому имя давать? Зачем? Когда их можно просто пронумеровать 0..9. На самом деле их количество ограничено размерностью адр. пр-ва ввода-вывода - FFFFh штук. Например, таймер имеет 4 порта - 40h..43h для различных каналов. (Не буду я тут схему работы таймера разбирать!!!) Фактически, обращаясь к портам i/o, вы общаетесь с устройствами почти напрямую. Для этого используются команды **in** и **out**:

in <аккумулятор - eax/ax/al> - из порта считывается байт/слово/дв. слово в аккумулятор.

out - содержимое аккумулятора пересылается в порт N. Если номер порта **<0FFh**, то можно задавать его непосредственно, иначе – через:

dx:xor ax,ax

mov dx,100h - порт с большим номером.

out dx,ax - выводим в этот порт нолик.

Кроме того, для вывода/ввода не по байтам, а строчками, есть команды:

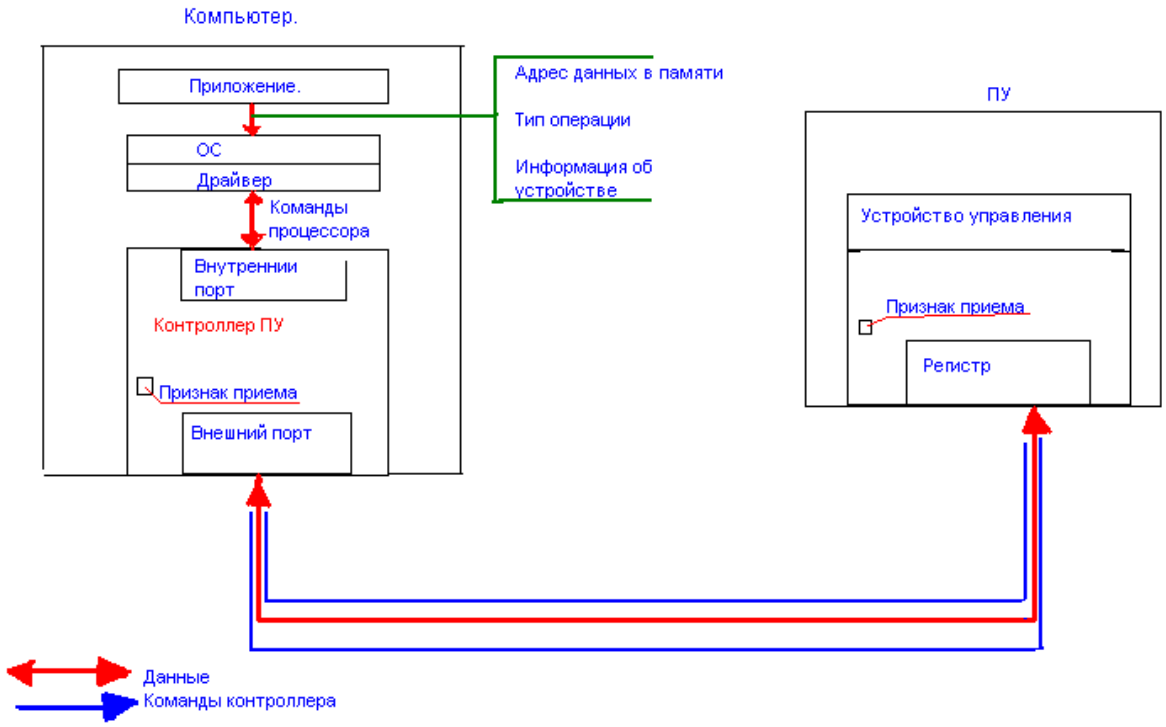
ins,outs (вернее **ins/insb/insw/insd** и ан-но **outs/.**) - перед вызовом их в **dx** вносим номер порта,

в **es:di** - строку для вывода,

а для **ins** - результат будет по адресу: **es:di**.

Контроллеры ввода/вывода.

Программа, выполняемая процессором, может обмениваться данными с помощью команд ввода/вывода с любыми модулями, подключенных к внутренней шине компьютера, в том числе и с **контроллерами** ПУ.



Контроллер ввода/вывода (controller input/ output) - используется для сбора данных с датчиков имеющих открытый коллектор, кнопок, переключателей или через блок гальванической развязки от датчиков другого типа, а также управления светодиодами или силовыми модулями типа Pwr3, PwrR и другими. Контроллер имеет 22 линии двунаправленного ввода/вывода.

Контроллер обеспечивает функции:

1. Управляет механизмом ПУ
2. преобразует коды символов в машинные (двоичные)
3. накапливает передаваемую информацию до полной записи
4. согласует по скорости работу ПУ с МП

Так как все типы контроллеров отличаются друг от друга, для управления ими требуется различное программное обеспечение (ПО). Программа, управляющая работой контроллера (отдает команды, получает ответы) называется драйвером.

Драйвер ввода/вывода (driver input/output) – это специальная резидентная программа, которая дополняет систему ввода и вывода, и обеспечивает обслуживание дополнительных внешних устройств. Он загружается в памяти компьютера при загрузке ОС, и хранится в файлах, имена которых имеют расширение: **.sys.**

Каждый производитель контроллеров должен поставлять драйверы для поддерживаемых им ОС.

Есть три способа установки драйвера в ядро:

1. Unix – заново компонуется ядро с новым драйвером и перезагружается
2. Windows – создается запись во входящим в ОС файле, в котором указывается, что требуется драйвер, если система его не находит при первоначальной загрузке, то он должен быть переписан с диска, после ОС должна быть перезагружена
3. Windows 2000, Windows XP – драйверы устанавливаются без перезагрузки ОС.

Каналы ввода/вывода.

Канал ввода/вывода (канал связи) – является общим звеном любой системы передачи информации. Он реализует процесс передачи информации по информационному каналу. Информационный канал (ИК)– это логический канал. Он устанавливается между двумя ЭВМ, соединенных физическим каналом.

ИК обеспечивает прием и передачу потока данных в виде кадров, в которые упаковываются информационные пакеты, обнаруживает ошибки передачи и реализует алгоритм восстановления информации в случае обнаружения сбоев или потерь данных.

По физической природе каналы связи бывают:

1) механические – обеспечивают передачу материальных носителей информации;
2) акустические – обеспечивают передачу звуковых сигналов;
3) оптические – обеспечивают передачу световых сигналов;
4) электрические – обеспечивают передачу электрические сигналы(Электрические каналы связи бывают проводные и беспроводные).

По форме представления передачи информации каналы связи делятся на:

1) аналоговые – передача информации представляется в непрерывной форме, т.е. в виде непрерывного ряда значений какой-либо физической величины;
2) дискретные – передача информации представляется в виде дискретных (цифровых, импульсных) сигналов той или иной физической природы.

Только возможностями контроллера не удается согласовать работу центральной части ПК и ПУ при обмене данными. Поэтому вводится еще понятие канала ввода-вывода (IO channel).

Понятие канала IO включает в себя: устройства ПУ, контроллер, драйвер, а так же внутренний блок управления ОС для файла или устройства.

Различают 2 типа канала:

1. мультиплексный – для работы с низкоскоростными устройствами (принтерами, сканерами, плоттерами)
2. селекторный – для работы с быстродействующими устройствами (CD-ROM)

Селекторные каналы работают в импульсном режиме, присылая или принимая в импульсе непрерывный поток данных.

ОС для каждого из каналов IO открывает процесс или файл.

Например, при загрузке MS-DOS автоматически открывается 5 стандартных процессов для передачи данных, связанных с файловыми манипуляторами.

0 – стандартный процесс, обеспечивающий ввод информации, связанный с файлом INPUT и связан с клавиатурой (CONsole)

1 – стандартный процесс OUTPUT для вывода информации на монитор (CONsole)

2 – ERROR - для вывода сообщений об ошибке, связан с монитором.

3 – стандартный выход для коммуникационного (последовательного) порта (AUX, COM1)

4 – стандартный выход, связанный с выводом на принтер (PRN, LPT1)

Процесс – это запущенная на выполнение программа с выделенными ей ресурсами. Каждый из перечисленных процессов имеет канал, который связывает указанный процесс с соответствующим устройством.

Процессы 0 и 1 можно переназначить на другие устройства (например, дисковые файлы), другие процессы 3 и 4 также можно переназначаются программными средствами.

К одному дисковому файлу можно открыть одновременно несколько каналов из одного или различных процессов (один канал для последовательного доступа к файлу, другой для прямого). Канал представлен в виде специализированного процессора, который занимается только организацией ввода-вывода.

Задачи канала:

Буферизация данных, подсчет байтов, введение счетчика адресуемой памяти.

Канал обеспечивает связь с центральным процессором с помощью 3-х полей:

1. CAW – Channel Address Word – адресное слово канала – содержит адрес начала программы работы канала.
2. CSW – Channel Status Word – слово состояния канала – содержит адрес данных, счетчик байтов, а также поле состояния, в котором фиксируется информация о занятости устройства, о завершении работы канала, о завершении канала ПУ, т.е. состоянии операции ввода-вывода
3. CCW – Channel Command Word – команда канала – содержит код команды (чтение, запись), адрес данных, определяющий место в ОЗУ, где искать данные или куда записать, поле счетчика байтов.

Связь с каналом реализуется через команды:

1) начать ввод-вывод – SIO
2) остановить ввод-вывод – MIO
3) опросить канал – TCH
4) опросить ввод-вывод – TIO

Это привилегированные команды, т.е. могут использоваться только модулем ОС в режиме ядра.

Порядок организации операций ввода-вывода:

1) Если в прикладной программе (ПП) встретилась команда read или write, то в МП происходит прерывание, организуется передача управления модулю ОС, при этом должен быть передан описатель устройства, с которым должна быть обеспечена связь. Каждая команда IO ориентирована на определение устройства.

2) Перед началом операции ввода-вывода модулем ОС посылается в CAW адрес начала программы канала, а также в CCW заносится адрес данных и значение счетчика байтов.

3) Модуль ОС посылает сигнал опроса через прерывание о состоянии канала для выполнения ввода-вывода.

4) Если канал свободен, то он отвечает соответствующим сигналом, который вносит код состояния в поле CSW.

5) Если канал свободен, то модуль ОС выдает команду SIO и канал запускает операцию ввода-вывода. МП в это время может передать управление другой задаче.

6) После окончания операции ввода-вывода канал через прерывание посылает сигнал МП об окончании операции, приняв данное прерывание, МП возвращается вновь на прерванную программу.

2.3.4 Управление заданиями (процессы, задачи). Очереди.

Процесс – минимальный программный объект, обладающий системными ресурсами.

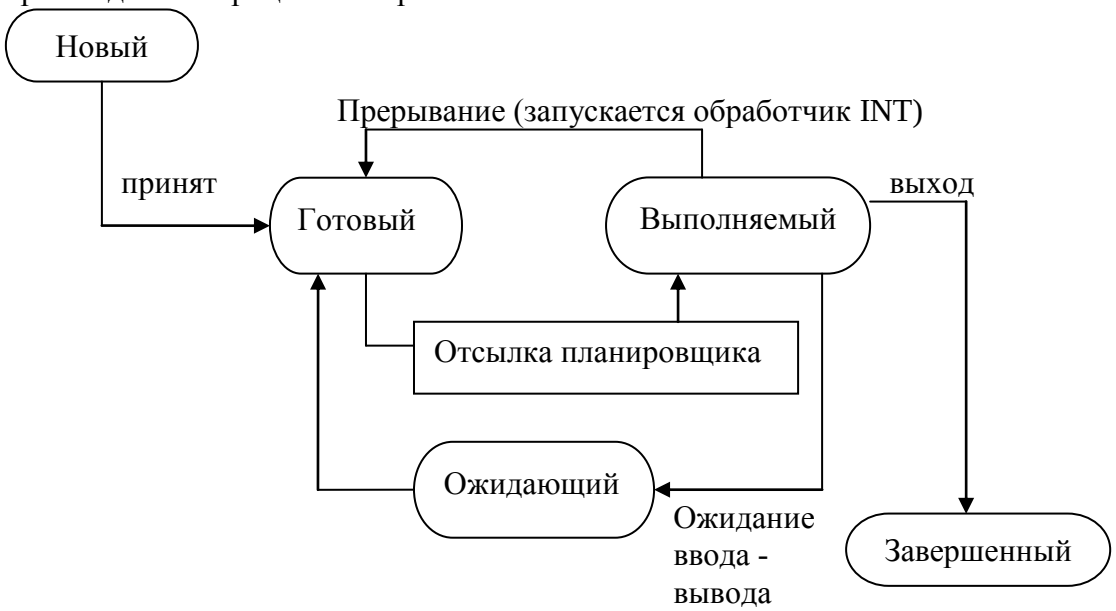
Программа – это план действий, с процессами действий, поэтому понятие процесса включает в себя:

1) программный код
2) данные
3) содержимое СТЭК.
4) Содержимое адресного и других регистров МП.

Для одной программы могут быть созданы несколько процессов в том случае, если с помощью одной программы в МП выполняется несколько несовпадающих последовательностей команд. Различают следующие состояния процесса:

- новый (процесс только что создан)
- выполняемый (команды программы выполняются в МП).
- ожидающий (процесс ожидает завершение некоторого события, например операция ввода – вывода).
- готовый (процесс ожидает освобождения МП).
- завершенный (процесс завершил свою работу).

Каждый процесс представлен в ОС набором данных, называемых таблица управления процессом PCB (Process Control Block). В PCB процесс описывается набором значений, параметров, характеризующих его текущее состояние и используемых ОС для управления прохождением процессом через ПК.



По времени развития процессы делятся на: последовательный, параллельный, комбинируемый.

По месту развития процессы делятся на : внутренние(реализуются на МП), внешние (на внешних процессорах (в сети) сервер).

По принадлежности к ОС: системные и пользовательские процессы.

По связанности различают процессы:

1) взаимосвязанные (информационные, управляющие, пространственно - временные).
2) Изолированные.
3) Информационно - независимые (общие ресурсы у них, но имеют собственные информационные базы).
4) Взаимодействующие.
5) Взаимосвязанные по ресурсам.
6) Конкурирующие.

Процессы могут, находится в отношении:

1) предшествования.
2) Приоритетности (процесса может быть переведен в активное состояние, если нет процессов с более высоким приоритетом).
3) Взаимного исключения (когда используются общий критический ресурс).

Ресурс – средство вычислительной системы, которая может быть выделена процессу на интервал времени.

Классификация ресурсов.

1) Делятся на: физические и виртуальные.
2) По времени существования ресурсы делятся: постоянные и временные.
3) Основные и второстепенные (допускаются альтернативное развитие процесса при их отсутствии).
4) Простые и составные: (доступен или занят – только два состояния).
5) Потребляемые и воспроизводимые (допускают многократное использование и освобождение).
6) Последовательное и параллельно – используемые ресурсы.
7) Жесткие (не допускают копирования) и мягкие (допускают тиражирование).

Управление процессами и распределение ресурсов между ними ОС осуществляет с помощью планировщика. ОС контролирует следующую деятельность, связанную с процессами:

а) создание и удаление.
б) планирование процессов.
в) синхронизация процессов.
г) коммуникация процессов.
д) разрешение тупиковых ситуаций.

Одним из методов планирования процессов, ориентированных на эффективную загрузку ресурсов являются методом очередей ресурсов.

Новые процессы находятся во входной очереди – очередь работ – заданий. Входная очередь располагается во внешней памяти, и ожидают освобождение ресурсов (место в ОЗУ). Готовые к выполнению процессы располагаются в ОЗУ и связаны с очередью готовых процессов и ожидают освобождения ресурса – процессорного времени.

Процесс в состоянии ожидания завершения операции ввода – вывода, находится в одной из очередей к оборудованию ввода – вывода.

Процесс мигрирует между различными очередями под управлением программы, которая называется планировщик.

ОС, обеспечивающий режим мультипрограммирования обычно включает 2 планировщика – долгосрочный и краткосрочный. Долгосрочный планировщик – планировщик заданий. На уровень долгосрочного планирования выносятся редкие системные действия требующие больших затрат системных ресурсов. Объектом является не отдельный процесс, а некоторые объединения процессов по функциональному назначению. Долгосрочный планировщик решает, какой из процессов во входной очереди должен быть переведен в очередь готовых процессов, при освобождении ресурсов памяти.

Краткосрочной планировщик – супервизор решает, какой из процессов в находящейся очереди готовых процессов должен быть передан на выполнение микропроцессоров. На уровне краткосрочного планирования выносятся частные и более короткие процессы. В некоторых ОС долгосрочный планировщик может отсутствовать, например, в системах разделения времени, при этом каждый новый процесс сразу помещается в ОЗУ. Выделение МП процессу производится многократно с целью достижения эффекта мультипрограммирования назначение – диспетчеризация.

Процессы могут взаимодействовать между собой. И называются взаимодействующими либо могут быть независимыми.

При взаимодействии рассматривают процесс производитель, процесс потребитель. При этом создается совместный буфер, заполняемый процессом производителя.

Буфер имеет фиксированный размер, поэтому процессы могут находиться в процессе ожидания, когда:

1) Буфер заполнен – ожидает процесс производитель.
2) Буфер пуст – ожидает процесс потребитель.

Буфер может предоставляться и поддерживаться самой ОС либо должен быть организован программистом, общий участок памяти.

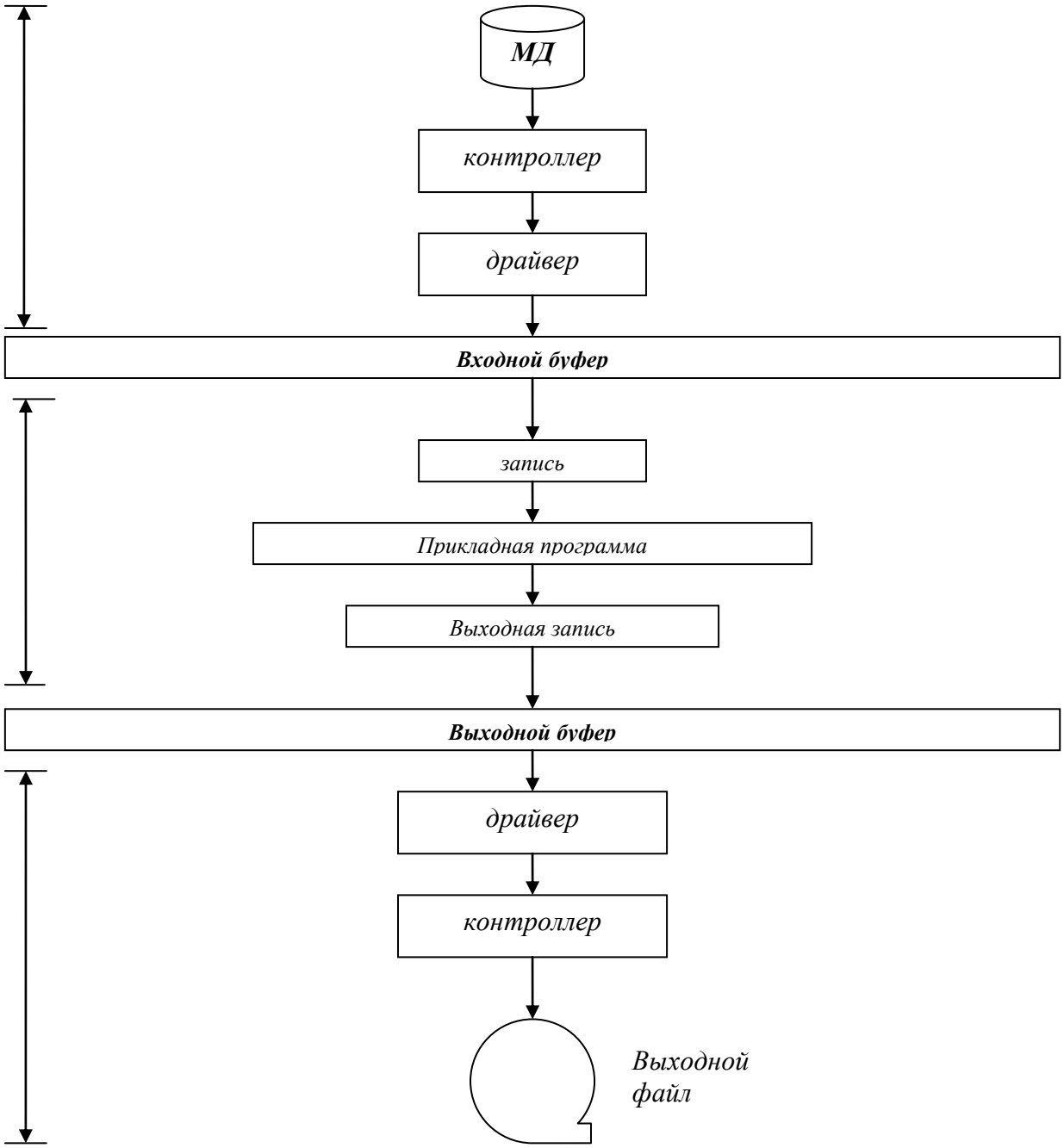
Взаимодействие заключается в передачи данных между процессами или в совместном использовании ресурсов. Обычно реализуется при помощи таких механизмов как:

1) Транспортеры, очереди, сигналы, семафоры, DDE, OLE, Clipboard. Взаимодействие аналогично считыванию или записи в файл. Расположен в памяти, при считывании данных из транспортера реализуется алгоритм, последовательные данные не подлежат повторному чтению данных. ОС контролирует порядок размещения данных на транспортере и наличие свободного места.
2) Очереди – механизм, который обеспечивает передачу или использование общих данных бес перемещения этих данных, а лишь по указателю этого элемента. Очередь используется вместе с механизмом общей памяти. Элемент очереди может быть считан с уничтожением или без уничтожения этого элемента. Чтение элементов очереди осуществляет только создающий очереди процесс, все другие процессы могут только записать элемент в очередь. Имя очереди имеет вид полной спецификации файла. Записывающий процесс осуществляет действия: открыть очередь, записать в очередь, закрыть очередь. Данный механизм позволяет эффективно передавать только элемент очереди, указывающий расположение этих данных.
3) Сигналы – являются механизмом передачи требования от одного процесса к другому на немедленное выполнение действия (является аналогом обработки прерывания). Характер выполняемых действий при возникновении сигнала: обработка системной ошибки, блокирование сигнала, передача управления подпрограмме.
4) Семафоры – являются механизмом передачи сообщений от одного потока к другому о наступлении некоторого события. Различают системные семафоры и семафоры ОЗУ.
5) В , MS Windows существует специальный механизм для взаимодействия процессов в реальном масштабе времени. DDE (динамический обмен данными) он стандартизует процесс обмена командами, сообщениями и объектами для обработки между задачами (организация печати). Другим интерфейсом для обмена данными является OLE – связь (Object Linking and Embedding) связывание и встраивание объектов. Данный интерфейс позволяет хранить объекты одной программой в объектах созданных другой программой, а также редактировать без нарушения цельностей информации и связей (текстовый документ Word и рисунок из Paint).
6) Буфер обмена (самый простой интерфейс межпрограммного обмена данными) Clipboard. Буфер обмена – область ОЗУ, которая может содержать в себе один информационный объект (рисунок и т.д.) с помощью системного вызова процесс может получить копию информации содержащейся в буфере обмена, или сам поместить в буфер.

2.4 Машинно-независимые свойства ОС.

2.4.1 Организация данных. Методы доступа.

Схема прохождения данных при обработке в ЭВМ.



Данные – любая информация, которая должна быть обработана ПК.
Поле – законченная по смыслу порция данных.
Запись – набор полей, содержащих множество данных, необходимых для выполнения 1 цикла программы.

Если поле – графа таблицы, то запись – строка:

ФИО	Год рождения	Адрес	Телефон
Босько И.Н.	24.11.1976	Г. Тосно	25983

Файл – набор соотносимых подобных записей, которая оформляется на внешних запоминающих устройствах (поименованная область на диске).

В ОП данные записываются в последовательных ячейках памяти. В зависимости от типа значения данные занимают определенное количество байтов. Все байты в ОП адресуются, поэтому можно обратиться на прямую к любому байту за выборкой необходимых данных. Тип адресации для каждого МП типа свой и указывается в архитектуре.

На ВЗУ данные хранятся в виде файлов и записываются в специально организованные области. Адрес включает в себя указание № цилиндра T, № поверхности H, № сектора S [t-h-s]. Информация на МД записывается по секторам и дорожкам, причем записи, принадлежащие одному файлу, могут находиться на разных дорожках.

FDD - Число дорожек составляет 40 или 80; 2 рабочие поверхности.

HDD - Может иметь 305-614 и другое количество цилиндров.

Количество рабочих поверхностей и цилиндров является аппаратной характеристикой дисковода.

Программно управляя контроллером дисковода можно изменить число цилиндров.

Число секторов на дорожке задается программно драйвером.

FDD – 8; 9; 15; 18

HDD – 17; 32 и более

Каждый сектор состоит из поля данных и поля служебной информации, которая ограничивает и идентифицирует его.

MSDOS поддерживает размер сектора 512 б.

BIOS Поддерживает 128; 256; 512; 1024

Число цилиндров, секторов и размер сектора устанавливаются при форматировании диска (формат, пробел, логическое имя диска).

Обмен информации между ОЗУ и МД физически осуществляется только секторами.

При создании файла ОС выделяет для файла дисковое пространство блоками – кластерами.

Кластер – минимальный объем дисковой памяти, выделенный файлу.

Методы доступа к данным – это алгоритм запоминания поиска записи в файле.

Существуют методы:

1. последовательный;
2. индексно-последовательный;
3. прямой.

В зависимости от метода файл должен иметь ту или иную логическую структуру для обеспечения доступа к записи файла.

(1) данный метод применяем к любой организации файлов, при этом к нужной записи обращаемся, просмотрев все предыдущие;

(2) все записи в файле связаны между собой в цепочку таким образом, что каждый их них имеет ссылку на номер последующей. Поэтому, обратившись к какой-либо записи можно раскрутить всю цепочку последующих. Недостаток в том, что сложно включить новые записи в цепочку;

(3) применим только к файлам, имеющим такую логическую структуру, при которой каждая запись строго фиксирована по количеству байтов, причем физически каждая запись имеет свой номер, начиная с 0, поэтому, указав номер интересующей нас записи можно напрямую обратиться к ней, не просматривая другие.

Устройства, поддерживающие *прямой метод доступа*: FDD, HDD

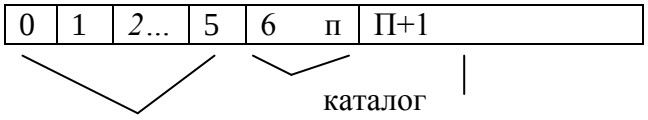
Последовательный доступ: strimer, терминальные и печатающие устройства.

2.4.2 Понятие файловой системы. Работа с файлами, планирование заданий

Файловая система – функциональная часть ОС, обеспечивающая выполнение операций над файлами и поддерживающая определенную файловую организацию.

Различают:

1) **Простейшая –** единственный каталог, в котором перечисляются сведения о файлах, а также указывается номер блока, с которым начинается файл. После каталога идет область данных – содержимое самих файлов.



ТОМ содержит логические блоки по 512 байт, которые пронумерованы.

Каталог размещен с 6 блока и содержит совокупность сегментов (сегмент = 2 блока).

Каждый сегмент каталога содержит заголовок:

- общее количество сегментов в каталоге;
- номер следующего сегмента;
- счетчик занятых сегментов;
- номер блока, начиная с которого начинается файл;

2. **Иерархическая** (древовидная) организация

Все дисковое пространство разделяется на кластеры.

Вся информация о состоянии кластеров на МД хранится в таблице распределения кластеров

FAT(File Allocation Table)

Иерархическая файловая структура – каждый MD содержит единственный каталог \, который включает подкаталоги и файлы. Минимальный объем, выделяемый дисковой памяти кластера (1,2 и более секторов). Вся информация о состоянии кластеров содержится в FAT (file allocation table) – таблица распределения кластеров, которая в двух экземплярах находится перед корневым каталогом. При создании файла номер начального кластера заносится в элемент каталога. А в FAT заносятся ссылки на последующие кластеры, принадлежащие данному файлу. Каждый диск хранит свою файловую структуру. Для ускорения доступа к файлу в иерархической файловой структуре необходимо вместе с составным именем файла (имя и расширение) передать маршрут поиска по каталогам.

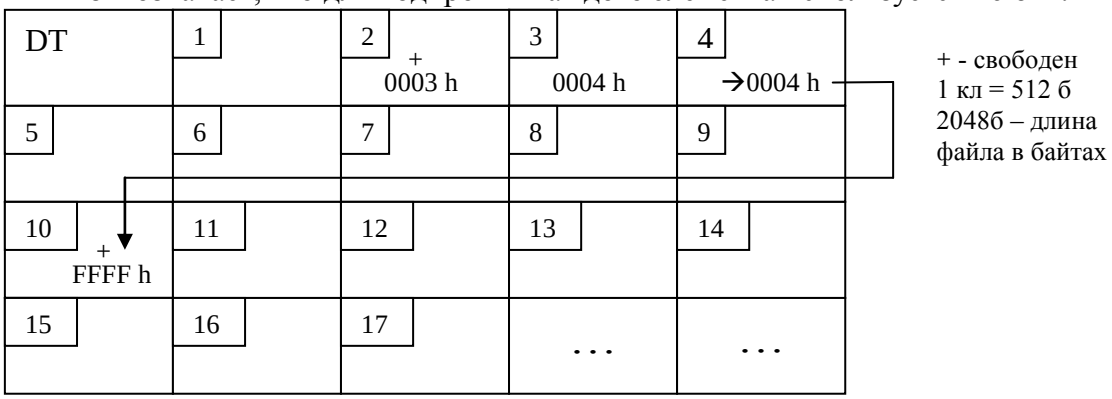
Каталог – это специальный файл, в котором регистрируются другие файлы. Все операции над файлами доступны через прерывания верхнего уровня 2Dh – 3Fh.

Чтобы воспользоваться услугами ФС по выполнению операции над элементами файловой структуры в программе следует выдать соответствующее прерывание (21h), а также предусмотреть загрузку необходимых аргументов в режиме МП. Сведения о файле ОС получает с помощью описателя файла. Получив управление, ФС идентифицирует запрос на выполнение операции (создать – CREATE, открыть – OPEN, закрыть – CLOSED, читать – READ, записать - WRITE), выбирает требуемый для этого драйвер и транслирует запрос в последовательность обращений к драйверу. Драйвер обеспечивает непосредственное управление контроллером МД по выполнению требуемой операции (через порт ввода - вывода), выдавая или принимая последовательность команд, а также анализируя операции их выполнения. После завершения операции драйвер возвращает управление и результаты своей работы ядру файловой системы, которая в свою очередь передает результаты программе, выдавший запрос.

Элементы FAT.

Дескриптор тома	1 доступный кластер	2 кластер	3 кластер	4 кластер
0	1	2	3	4	5

Смысл кодировки элементов в FAT.
0000h – свободен.
0001h – FFEFh – занят (указывается № кластера).
FFF0h – FFF6h – зарезервирован.
FFF7h – дефективный.
FFF8h – FFFFh – конец цепочки кластеров.
FAT 16 – означает, что для кодировки каждого элемента используется 16 бит.

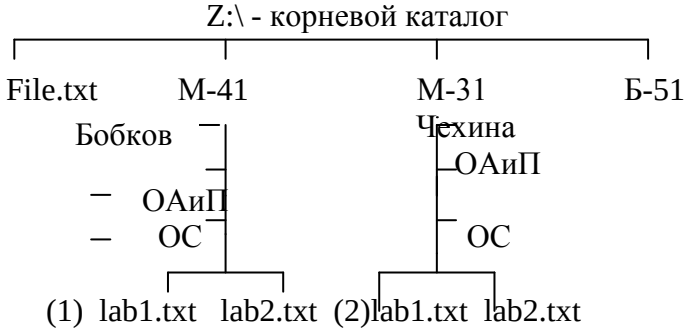


При создании файла в элемент каталога заносится: - file.txt – имя файла.
- атрибут – архивный.
- 19 октября 2004 – дата.
- 0002h – первый кластер, с которого начинается цепочка.

Таким образом, любой элемент каталога ФС может быть найден по указанию его имени, так как с именем связан номер начального кластера, с которого начинается файл, а затем ФС обращается к FAT, где указана вся цепочка кластеров, принадлежащих данному файлу, от номера 1 до последнего – FFFF. Вновь создаваемым или пополняемым файлом выделяются ближайшие к началу элементы FAT, но еще свободные и соответствующие им кластеры на МД. При создании файла формируется элемент в каталоге, содержащий информацию:

- имя файла;
- расширение (тип);
- атрибуты (только для чтения, архивный, скрытый, системный);
- дата создания; время создания;
- длина файла в байтах (для подкаталогов нулевая длина);
- номер начального кластера, который выделен для файла.

Кластеры, выделяемые при создании файла, могут находиться не в смежных областях МД, но все они соединены в цепочку, информация о которых содержится в FAT.



(1) Z:\М-41\ Бобков\ ОС\lab1.txt (2) Z:\М-31\ Чехина\ ОС\lab1.txt

Каждый диск хранит свою файловую структуру. Имеется единственный корневой каталог (\),в который могут входить другие каталоги и файлы.

Для ускорения доступа к файлу древовидной файловой системы необходимо вместе с основным именем указать маршрут его поиска по каталогам.

Каталог – специальный файл, в который регистрируются другие файлы.

При поиске файла на МД ФС обращается к элементу каталога и считывает номер начального кластера. Затем обращается к FAT.

FAT В ячейке таблицы по данному номеру начального кластера содержит номер следующего кластера из цепочки, принадлежащей данному файлу.

Так последовательно один за другим раскручиваются все кластеры для данного файла, конечный содержит код FFFF. Каждому кластеру соответствует физический адрес на МД, поэтому файловая система раскручивает всю цепочку кластеров из FAT, передает управление драйверу МД, который осуществляет считывание информации из файла.

FAT хранится в двух идентичных экземплярах. После завершения работы с файлом или МД обе копии обновляются.

Для ускорения работы при обращении к файловой системе FAT загружается в КЭШ – память, т.е. при активной работе с файлами, информация из FAT Считывается по обращению к Кэш- памяти. По завершении работы обе копии обновляются и FAT выгружается.

Все операции над файлами, также открытие и закрытие файлов доступны перед IRQ –я верхнего уровня 20h-3Fh (кроме 21h-23h).

Чтобы воспользоваться услугами файловой системы программно необходимо выдать соответствующие IRQ, ex, 21h и установить необходимые аргументы в регистрах МП.

Получив управление, файловая система идентифицирует запрос последовательность обращений к драйверу. Драйвер обеспечивает непосредственное управление устройства по выполнению требуемой операции. Через порт ввода/вывода, выдавая последовательность команд, а также принимая и анализируя результаты их выполнения. После завершения операции драйвер возвращает управление и результаты своей работы ядру файловой системы, которая передает результаты программе, выдавшей запрос.

Файловая система MSDOS реализует операции ввода/вывода синхронно, то есть выполнение программы приостанавливается до завершения операции. MSDOS поддерживает работу с FAT16 WINDOWS95, 98, FAT32.

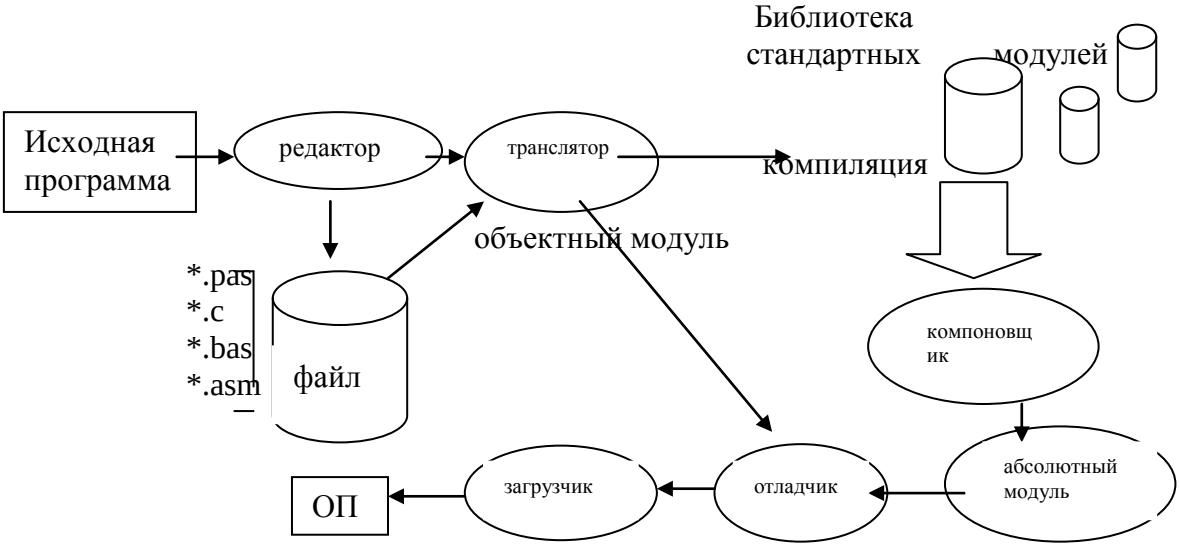
Монитор обслуживается файловой системой не через IRQ-я, а через память в виде адаптера непрерывно. В каждом сеансе работы формируется логический видеобуфер. Все запущенные на выполнение процессы имеют свою структуру в памяти видеобуфера. При необходимости можно переключиться между задачами, при этом активный процесс в данный момент имеет нулевую страницу видеопамяти и отображается на экране.

Файловые устройства:

Винчестер HDD, FDD, CD-ROM, CD-RW, Стример – блочного типа. Не файловые устройства все остальные – символьные требуется указать для работы с символьными устройствами в MSDOS подключение драйвера данного устройства device.

В каждом сеансе работы формируется логический видеобуфер. Все запущенные на выполнение процессы имеют свою страницу в памяти видеобуфера, при необходимости можно переключаться между задачами. При этом активный процесс в данной момент имеет нулевую страницу видеопамяти и отображается на экране.

2.4.3 Система программирования. Этапы прохождения задачи в системе программирования.



***Этапы прохождения задачи в системе программирования**:*

1. Исходный текст программы на каком-либо языке программирования вводится с помощью текстового редактора в ОП ПК после команды сохранить save формируется на МД файл, расширение соответствует типу файла, т.е. указывает на язык программирования;
2. Набранный текст программы должен быть переведен в машинные коды перед выполнением программы. *Транслятор* – программа переводчик исходного текста программы в машинные коды, а также проверяющий наличие синтаксических и соматических ошибок;
   Различают трансляторы:

- compiler – компилирующего типа или компилятор – проверяет исходный модуль программы на наличие синтаксических ошибок и переводит в машинные коды, если ошибок нет;
- интерпретирующего (интерпретатор) Interpreter по командно проверяет на наличие ошибок, переводит в машинные коды и выполняет.

   После компиляции получается объектный модуль программы (файл с расширением *.obj);

3. Компоновщик (редактор связи) объединяет объектный модуль программы и объектным модулем процедур и функций из библиотеки стандартных код программ, вызовы которых имелись по тексту исходной программы. Модули из библиотек вставляются целиком в текст исходной программы и каждой команде присваивается относительный адрес (относительно 0). После компоновки формируется абсолютный (загрузочный модуль программы) (расширение *.exe). Если программа выполняется в среде мультипрограммирования ОС, то при компоновке ей присваивают атрибуты:

- приоритет в системе;
- раздел в памяти;
- код защиты;
- признак выгружаем ости или резидентности;
- время цикла выполнения программы, если она должна выполнятся циклически.

4. Отладчик debugger – программа, которая позволяет выявлять логические ошибки в решении задачи. Программа запускается под управлением отладчика и с помощью точек останова можно просматривать выполнения команд программы, а также

содержимое регистров МП и содержимое переменных, используемых в данной программе;

5. Загрузчик организует загрузку абсолютного модуля программы в ОП, если программа была запущена в режиме отладки, то процессом выполнения ее будет управлять отладчик. Адрес загрузки программы будет передан МП в счетчик адреса команд. МП начинает по командное выполнение программы.

2.5 MSDOS Структура

2.5.1 Состав и назначение основных компонентов MS DOS

1.**BIOS (basic Input Output System) –** базовая система ввода/вывода. Является частью ПК, т.к. зашита в ПЗУ, является компонентом DOS или любой другой ОС, т.к. содержит драйверы стандартных ПУ.
BIOS Освобождает обращающиеся к ней программы и другие компоненты DOS от знания и учета особенности оборудования и деталей управления ПУ, обеспечивая независимость ПО от ПУ.

BIOS содержит:

1. драйверы стандартных ПУ;
2. тестовые программы проверки работоспособности АО;
3. программу начальной загрузки;
4. интерпретатор языка Basic (может и не быть).

BIOS реализует основные функции:

1. автоматическую проверку аппаратных компонентов при включении питания;
2. вызов системного загрузчика программой первичным загрузчиком, который обращается к стартовому сектору загрузочного диска и считывает программу системного загрузчика в ОЗУ.
3. Обслуживает систему вызовов или прерываний нижнего уровня, которая обеспечивает доступ к средствам BIOS, EMBIOS (00h – 1Fh).

Системный загрузчик (system bootstrap) программа 512б находится в стартовом секторе каждого диска, подготовленного в среде MSDOS

Основная функция – считывание с МД и загрузка в ОП двух частей DOS:

- Io.sys (IBMBIO.COM)
- Msdos.sys(IBMDOS.COM)

2.Модуль расширения BIOS (файл io.sys) является надстройкой над BIOS В его задачи входит:

1. организация интерфейса с BIOS, т.е. настройка на конкретную версию ОС;
2. логическая замена драйверов, находящихся в BIOS;
3. подключение внешних драйверов.

Драйверы могут быть внутренними и входят в состав модуля расширения EMBIOS они загружаются автоматически при загрузке DOS, а могут быть и внешними, т.е. существовать в виде отдельных файлов.

Подключение внешних драйверов обеспечивается командой device=c:\dos\mouse.com, причем указывается путь, где находится драйвер.

Данная команда может включатся в файл конфигурирования (config.sys), либо в файл автозапуска (autoexec.bat).

Доступ программ в средствах EMBIOS осуществляется через аппаратные прерывания нижнего уровня 00h-1Fh

BIOS, EMBIOS, внешние драйверы составляют систему ввода/вывода DOS, т.к. главная функция – обмен информацией с ПУ (это машинозависимая часть DOS).

3.Базовый модуль DOS (файл msdos.sys) – центральный компонент DOS, реализующий функции по управлению всеми ресурсами ПК, обеспечивает работу файловой системы, устройств (с помощью драйверов), выполнение программ пользователя, обслуживание некоторых специальных ситуаций, связанных с завершением программ, их искусственным прерыванием и обработкой ошибок.

Большинство программ работают под управлением DOS выдает прерывание верхнего уровня (20h-3Fh). обработчики этих прерываний в свою очередь могут выдавать прерывания нижнего уровня для связи с внешними устройствами.

Базовый модуль DOS - это машинно-независимая часть DOS.

Система ввода/вывода и базовый модуль DOS. Находится в ОЗУ резидентно и управляют ресурсами ПВМ.

4.Командный процессор (command Interpreter CI) (файл command.com). отвечает за поддержку пользовательского интерфейса и выполняет функции:

1. воспринимает команды DOS вводимые пользователем с клавиатуры;
2. выполняет внутренние команды;
3. обрабатывает командные файлы (*.bat);
4. загружает программы в ОЗУ для выполнения *.exe ; *.com- исполняемый файл;
5. функция обработки прерывания.

22h – завершение задачи

23h - нажатие ctrl-break (принудительное завершение задачи)

24 h - реакция на критическую ошибку.

Командный процессор состоит из следующих модулей:

1. резидентный модуль – находится в ОЗУ после запуска DOS и включает в себя обработку прерываний 22-24h И код подзагрузки транзитной части командного процессора;
2. транзитный (нерезидентный) модуль, который перекрывается выполняемыми программами. Содержит интерпретатор внутренних команд DOS и загрузчик программ в ОЗУ для исполнения.

При загрузке DOS на завершающем этапе файл командного процессора загружает на исполнение файл автозапуска, если он есть.

В файле могут быть созданы следующие действия:

- установка режимов работы DOS;
- завершение формирования окружения DOS;
- загрузка резидентных программ;
- установка даты и времени;
- установка маршрутов поиска исполняемых файлов;
- запуск программ оболочек DOS И другие.

5.Оболочки – облегчают работу пользователя в среде DOS: Norton Commander, Dos-shell, Volkow Commander.

6.Утелиты. Обеспечивают:

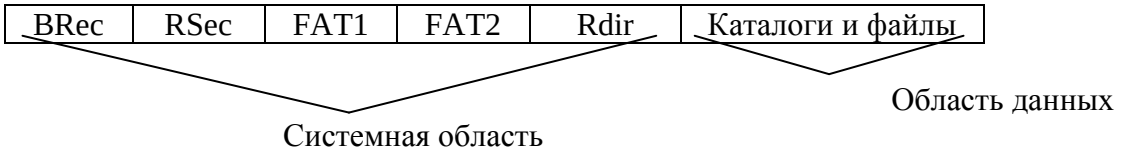
- выполнение внешних команд;
- реализуют сервисные услуги в интерактивном режиме.

7.Сервисные программы. Находятся в отдельных файлах или группе файлов, в функции которых входит разработка программ и подготовка текстовых документов.

Базовый модуль DOS, командный процессор, утилиты DOS – машинонезависимая часть DOS

DOS имеет модульную структуру. Основным механизмом функционирования DOS является система прерываний.

2.5.2 Структура логического диска



1. Boot Rec – загрузочная запись содержит блок параметров BIOS, в котором указана емкость дисков, месторасположение FAT И каталога; имеется идентификатор поставщика (№ версии, название фирмы) DOS, находится процедура загрузки (системный загрузчик SB);
2. RSec – зарезервированные сектора используемые DOS;
3. FAT1 и FAT2 – таблица размещения кластеров. При создании файла ему выделяются кластеры. Обе копии обновляются при закрытии файла или окончании работы с МД. Используется при работе только FAT1, но с помощью FAT2 можно восстановить FAT1 при порче;
4. каждый элемент каталога состоит из 8 полей и содержит 32 байта

Поле	Длина, б	Описание
0	8	Имя файла

00h – некогда не использовался
E5h – удален
2Eh – код точки – элемент каталога описывает данный каталог
O5h – имя файла или каталога

8	3	Расширение или тип файла
11	1	Атрибуты файла

00 – разрешено чтение и запись
01 – только чтение
02 – скрытый
04 – системный
08 – метка тома или диска
10 – подкаталог
20 – архивный файл

12	10	Зарезервировано DOS
22	2	Время создания
24	2	Дата создания файла или каталога
26	2	Отводится под номер 1 кластера
28	4	Длина файла

При создании подкаталога в родительском каталоге формируется запись, содержащая имя подкаталога, атрибут 10h и нулевую длину.

В самом подкаталоге формируется запись, в которой два первых байта имени файла начинается с кода 2E (2 точки), т.е. этот элемент описывает родительских каталог, эта запись используется для перемещения по дереву вверх.

2.5.3 Файловая система DOS

MS DOS Поддерживает иерархическую (древовидную) структуру файловой системы.

2.5.3.1 Работа в файловой системе. Команды MSDOS

Команда_X1_X2/A1/A2

Где: К – команда
X1,X2 – аргументы
A1, A2 - ключи

Различают внутренние команды и внешние (внутренние выполняет процессор, внешние существуют в виде отдельных файлов)

1. DIR– просмотр содержимого каталога постраничное краткое;

2. создать каталог MD _имя каталога;
3. сменить текущий каталог или директорий CD _имя каталога; CD_\ - войти в корневой каталог текущего диска;
4. смена магнитного диска A:;
5. удаление каталога RD_имя каталога. Для удаления каталога необходимо, чтобы он был пуст;
6. TREE (внешняя) выводит сообщение о структуре каталогов
7. DELTREE удаление (внешнее) подкаталогов;
8. COPY копирование файлов и подкаталогов: COPY _что _под каким именем
   Откуда куда
   Источник приемник:

- Copy f1.pas a: копировать на дискету из текущего каталога текущего диска;
- Copy *.pas PAS копировать все pas-файла в подкаталог с именем PAS текущего диска;
- Copy file1.pas+file2.pas+file3.pas file.pas объединить 3 файла в один (если имя объединенного файла не указано, то берется имя первого);
- Copy a:. B: *.bak копировать все с bak Расширением;
- Copy *.* a:\TEMP копировать все файлы из текущего каталога на дискете в подкаталог TEMP;
- Copy con f1.txt создать текстовый файл;
- Copy f1.txt con вывод содержимого файла на монитор;
- Copy f1.txt+con дополнение текстового файла;
- Copy f1.txt prn распечатать содержимое файла;
- Copy f1.txt/A f2.txt/B скопировать в файл с отсечением маркера без EOF;
- Copy a:. + b:. c: одноименные файлы на диске A и B скопировать на диск E;

9. удаление файла del_имя файла;
10. просмотр содержимого файла type_имя файла;
11. переименование, перенос файла ren _старое имя_ новое имя;
12. (внешняя) изменение атрибутов ATTRIB _ +-R -только для чтения
    +-H – скрытый
    +-S- системный
    +-A - архивный (+установка; -снятие);
13. COMP_СПФ1_СПФ2 – сравнение спецификаций файлов.

Команды обслуживания дисков (внешние):

ASSIGN_ старое имя = новое имя – переименование дисковода
FORMAT _ имя дисковода_/ключи
CHKDISK _ d: / ключи – проверка диска
/F – фиксирует все ошибки, которые обнаружены;
/V – вывод всех файлов и путей их доступа
DISK copy d: _d: - копирует содержимое с одной дискеты на другую
DISKCMP d: _d: - сравнение содержимых двух дискет

Команды конфигурирования системы:

1. SET устанавливает операционное окружение, при этом вводит различные имена с параметрами, которые затем могут использоваться прикладными программами;
2. PATH – установка маршрутов поиска исполняемых файлов (*.exe; *.com; *.bat);
3. PROMPT_ $d (дата), $v(версия) $p (выдает текущий каталог) $g (корневой каталог) – задание видопригалшения;
4. CLS – очистка экрана;
5. BREAK on[off] – отключение реакции на нажатие клавиш ctrl-break;
6. buffers=n[m] – установка количество буферов для обменных дисковых операций;
7. files=n количество одновременно открытых файлов;
8. stacks=n[s] количество и размер стеков для аппаратных прерываний;
9. device – подключение внешних драйверов;

10. shell= путь\ command.com команды переназначени командного процессора;
11. mode_ con_ deplay=l
    rate=r - установка скорости работы клавиатуры
    l – задержка регенирации
    r – частота регенирации символов;
12. install=file – загрузка и выполнение резидентной программы во время выполнения файла confic.sys;
13. find_СПФ ищет в файле строку с указанным текстом;
14. ncd – осуществляет быстрый поиск директория;
15. si – выдает информацию о конфигурации ПК.

2.5.4 Командные файлы

Файлы, составленные из команд MSDOS Имеют расширение (*.bat). командным файлом называется последовательность команд DOS, записанная в текстовый файл и выполняемая путем ввода спецификации этого файла, аналогично единственной команде DOS.

В командном файле могут использоваться до 10 замещающих параметров от %0 до %9, причем %0 всегда замещается указатель дисковода. При запуске командного файла, создании параметров, требуется в командной строке через пробел после имени файла указать фактические значения, которые будут подставлены вместо %1, %2 и т.д.

Пример: %1 %2 %2 %3

Пакетные команды:

1. REM_ сообщение – комментирует указанную строку, либо командный файл;
2. ECHO [on/off] _[сообщение] – производит включение или отключение отображения строк командного файла;
3. организация паузы PAUSE_сообщение – организуют приостановление выполнения очередной команды в командном файле и выводит стандартное сообщение «нажмите любую клавишу для продолжения».
4. Команды безусловного перехода.
   GOTO_метка затем любая строка должна быть отмечена меткой. Если командный файл не содержит строку с меткой указанный в GOTO, то его выполнение по GOTO завершается.

   Вывод на принтер файл %1 для остановки нажмите %1 нужна еще копия 1.
5. Команда условного перехода
   IF_Условие_ команда
   Позволяет включать условное ветвление в командный файл
   Условия:

- ERRORLEVEL _код получения ошибки или завершения команды;
- string1==string2
- exist_filespec

1) Число в строке ERRORLEVEL относится к значению возможного выходного кода, который может быть установлен ранее выполнявшейся программой (командой). Нормальное завершение равно 0, в противном случае ERRORLEVEL Выдает код ошибки;
2) Данное условие истина, если обе строки одинаковы.
   Строковое значение не могут включать символы : :;,= ;
3) IF EXIST_filespec проверяет существование файла указанной спецификации.
   Прервать выполнение данной команды Ctrl-Break

6. FOR позволяет организовать циклическое выполнение заданной команды DOS
   FOR_ %%переменная_IN _(набор файлов)_ DO _ команда

переменная – 1 буквенный символ. Если команда исполняется в командном файле, то ставится %% перед именем переменной. Если команда FOR Задается в командной строке то - %.

Команды FOR не могут быть вложенными.

При выполнении команды FOR одно символьной переменной последовательно присваиваются значения спецификации файлов (или дисков) из набора указанного в скобках после IN, для каждого из них выполняется команда, заданная после DO.

7. shift - позволяет использовать больше 10 заменяющих параметров в командном файле. Каждая команда shift Приводит к смещению параметров, указанных в командной строке влево на одну позицию;
8. choice /C:(перечисление возможных значений для ввода) _ «строка»

```
 echo 1- запуск Borland
 echo 2 - запуск Редактора
echo  3 – Форматирование
choice /C:123 «Ваш выбор:»
if errorlevel – 3 goto Format_A
if errorlevel – 2 goto Word_Pad
      C:\BP\BIN\bp.exe
Goto end
 : format_a
    format_a:
  goto end.
     …..
```

2.5.5 Утилита BE (batch enhancer) Файл be.exe.

Является дополнительным средством разработки командный файлов. Поддерживает 17 специальных команд и запускается из командных файлов.

BE_команда _ аргументы _ /переключатели

1. BE_SA – служит для установки режима отображения информации и цвета
   SA/N - цвет каймы экрана не устанавливать;
   SA/CLS – очистить после смены режима экран
   (SA BLUE ON RED - голубой на красном)
2. CLS_color - очистка экрана с установлением цвета;
3. ASK – предназначена для разработки интерактивных командных файлов путем организации разветвлений в них в соответствии с ответом пользователя на запрос.
   ASK_ ‘string’ default= кодов timeout=n color
   ASK ‘Ваш ответ Y/N:’ YN default=n timeout=180 black on red

   по команде ASK На экране начиная с текущей позиции выводится сообщение “string” указанным цветом и BE Ждет ответа пользователя. Варианты ответа должны быть специфицированы (Y или N), каждому из них соответствует код по номеру позиции (Y-код1, N-код 2) после нажатия клавиш утилита вырабатывает код возврата, который может быть проанализирован командой if errorlevel.

   код возврата перехватывается if errorlevel И проверяется. Если соответствует 2=N.
4. BEEP /Dm/Fn/Rp/Wq
   /Dm - Длительность в тиках. 1 тик соответствует 1/18 с
   /F - частота звукового сигнала
   /R - повторить сигнал p раз
   /W – осуществить задержку равную q тикам
5. DELAY _ n - осуществляет задержку выполнения очередной команды на n тиков

6. PRINTCHAR _символ_n _ [color] – выводит строку указанных символов в количестве n заданным цветом;
7. RONCOL R C ‘string’ [color] - позиционирует курсор на экране и выводит заданную строку заданным цветом
   r – номер строки
   c – номер позиции;
8. BOX t / br SINCLE/DOUBLE color - построение рамки на экране;
9. WINDOW t l b r [EXPLODE] _ [SHADOW] color – окно открывается на экране с указанием координат для вывода информации
   explode – эффект распахивающегося окна
   shadow – с тенью
10. Month_day – выдача текущего дня месяца
11. Week _ day – выдача текущего дня недели
12. GOTO_ метка
    File goto M– обеспечивает выполнение цвенария заданного командного файла не с начала, а с заданной метки.

2.5.6 Оболочка Total COMMANDER

Режимы отображаемой информации на панели задается через управляющее меню – вызов по F9.

Назначение функциональных клавиш:

F1 – помощь
F2 – меню
F3 – просмотр файла
F4 – редактирование файла
F5 – копирование
F6 – перенос / переименование
F7 – создать каталог
F8 – удалить каталог или файл
F9 – выход в управляющее меню
F10 – выход

Работа с файлами:

1. Выбор группы файлов INS «+» – открывается окно;
2. копирование с запросом SHIFT F5
1. Снять выделение INS «-«;
2. Перенос F6 (SHIFT F6);
3. F8 удаление;
4. изменение атрибутов F9, файл изменить атрибуты клавиша пробел или мышью отмечать крестиком установление атрибут;
5. настроить панель на показ скрытых файлов F9, команда , конфигурация, опции панели;
6. просмотр файла F3 CTRL+Q - быстрый просмотр;
7. редактирование F4

- F3 выделить блок
- SHIFT F3 снять выделение
- F5 скопировать выделенный блок перед курсором
- F6 переместить перед курсором
- F8 удалить выделенный блок
- F7 поиск подстроки в тексте
- F2 сохранить файл
- SHIFT F2 сохранить под другим именем

- SHIFT F4 создать новый файл
- ALT F10 добавить блок текста в файл
- ALT F5 вставка в редактируемый документ другой файл

8. поиск файла на диске ALT F7

Работа с каталогами:

1. F7 – создание;
2. F8 – удаление;
3. Alt F10 – дерево каталогов;
4. сравнить содержимое подкаталогов – F9 –команда- сравнение каталогов.

2.5.6 Общие сведения о компьютерных вирусах:

Компьютерный вирус – программа способная внедрятся в другие программы, а также способная к само воспроизводству.

Компьютерный вирус имеет фазы жизнедеятельности:

1) латентный период;
2) инкубационный (скрытно);
3) период проявления (действие).

- файловый вирус, идентифицирующий программные файлы;
- загрузочные вирусы, заражающие компоненты системной области;
- файлово-загрузочные.

Файловые вирусы могут внедряться:

- в com и exe Файлы;
- во внешние драйверы устройств;
- в OBJ (объектные) файлы;
- файлы с программами на языке программирования и режиме компиляции;
- командные файлы(bat);
- библиотечные модули.

Загрузочные вирусы могут заражать Boot Record на дискетах или системных дисках.

Файлово - загрузочные распространяются как в программных файлах так и на дискетах с данными.

Тело файлового вируса может размещаться в конце файла, в начале, в середине или в хвостовой части.

В зависимости от способа активизации различают:

- нерезидентные вирусы – тело вируса выполняется 1 раз при загрузке программы;
- резидентные, которые находятся в ОЗУ, подменяют некоторые обработчики прерываний, например при открытии и считывании файла.

Проявление вирусов:

- искажение программных файлов и файловых данных;
- искажение загрузочной записи;
- нарушение связанности файлов путем искажения FAT;
- форматирование диска или его частей;
- влияние на работу ПК.

По способу маскировки различают:

1) не маскирующиеся;
2) само шифрующиеся;
3) стелс-вирусы (не проявляются).

Симптомы наличия вируса:

1. увеличение числа файлов на дискете;
2. уменьшение объема свободной оперативной памяти;
3. замедление или ненормальная работа программы;

4. загорание лампочки дисковода, когда нет обращения;
5. увеличение времени доступа к магнитному диску;
6. появление на диске зарезервированных дефектных кластеров;
7. разрушение файловой структуры FAT.

Классификация антивирусных средств

Антивирусные программы выполняют следующие функции:

- защита данных (файловой структуры) от разрушения;
- обнаружение вируса;
- нейтрализация.

Различают:

- вирус фильтры – резидентная программа, обеспечивающая контроль выполнения действия вирусов (фиксированную подмену обработки прерываний);
- детекторы – программа, осуществляющая поиск вируса как в ОЗУ так и в ВЗУ;
- дезинфекторы (доктора) – программа, осуществляющая удаление вирусов с восстановлением или без восстановления среды обитания;
- иммунизаторы – программа, предотвращающая заражение среды обитания или памяти конкретным вирусом. Не уничтожается, а блокируется его способность к размножению.

Файлы восстановления:

- PC-Cein
- Kaspersky
- Spider Guard
- Avast
- NOD32
- **Одной из самых больших проблем, подстерегающих пользователей компьютеров, всегда были компьютерные вирусы, но в последние годы к ним добавились и шпионские** программы. Для домашнего компьютера, на котором нет конфиденциальной информации, это не очень большая проблема, но, все равно, не очень приятно, когда информация о владельце компьютера, куда-то передается без его ведома.
- Из этой статьи вы узнаете, как повысить уровень безопасности вашего компьютера, чтобы вы перестали беспокоиться о важных данных и не позволили ничему отправлять информацию с вашего компьютера без вашего разрешения.
- Со временем мы будем дополнять эту статью, так что следите за обновлениями. Можете присылать полезную информацию по этой теме и свои отзывы.
- **Пароли**
- Первое, с чего стоит начать – пароли. Здесь приводятся ключи реестра, связанные с паролями. Главное правило, которому нужно следовать, если вы хотите, чтобы ваши пароли были эффективными – не используйте слова, которые можно найти в словаре, или наборы цифр. Лучше всего комбинируйте цифры и буквы в разных регистрах и не храните ваши пароли в легкодоступных местах.
- **Запрашивать пароль при возвращении к работе из режима ожидания** *[HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\System\Power]* "PromptPasswordOnResume"=`1`
- **Требовать пароли только из букв и цифр** Этот ключ заставит вас всегда комбинировать в паролях буквы и цифры. *[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Network]* "AlphanumPwds"=`1`
- **Установка минимального количества символов в паролях** *[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Network]* "MinPwdLen"=hex:6

- **Отмена сохранения паролей в Internet Explorer** Если вы доверяете компании Микрософт в хранении паролей и другой конфиденциальной информации, то можете разрешить Windows хранить пароль доступа в Интернет на диске своего компьютера, но это не очень хорошая идея.
  [HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings] "DisablePasswordCaching"=`1`
- **Запрет хранения паролей**В прошлых версиях Windows 9x сохранение паролей было большой проблемой. Теперь это не так, Windows 2000 и XP защищают эту информацию значительно лучше. Но, опять же, вам решать, позволить операционной системе хранить пароли на диске или нет. Это касается паролей пользователей и сетевых паролей.
  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Network] "DisablePwdCaching"=`1`
-
- **Сеть**
- **Запрет доступа для анонимных пользователей**Анонимный пользователь может получить доступ к списку пользователей и открытых ресурсов, чтобы это запретить, можно воспользоваться этим ключом.
  [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LSA] "RestrictAnonymous"=`1`
-
- **Не показывать пароли при вводе**При попытке доступа к защищенному паролем ресурсу, Windows не скрывает пароль, который вы вводите. Этот ключ позволяет заменять символы пароля звездочками.
  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Network] "HideSharePwds"=`1`
- **«Прячем» компьютер от других пользователей в сети** Этот ключ позволяет включить режим, при котором в режиме обзора сети другие пользователи не будут видеть вашего компьютера.
  [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters] "Hidden"=`1`
- **Убираем следы своей работы за компьютером**
- **Очистка файла PageFile** Уничтожение при завершении работы всей информации, которая могла сохраниться в системном файле Page File.
  [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management] "ClearPageFileAtShutdown"=`1`
- **Автоматическое удаление временных файлов после работы в Интернет**0 заставит Internet Explorer удалять все временные файлы, такие как изображения с web-страниц и другую информацию, оставшуюся после работы в Интернет, а 1 позволит оставить эти файлы на диске.
  [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Cache] "Persistent"=`0`
- **Отменить сохранение списка документов, с которыми вы работали**
  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer] "NoRecentDocsHistory"=`1`
- **Отмена сохранения информации о действиях пользователя** Этот ключ запрещает записывать, с какими приложениями недавно работал пользователь, и к каким документам он получал доступ.
  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer] "NoInstrumentation"=`1`

- **Пакетный файл для удаления временной информации**Этот пакетный файл будет удалять всю временную информацию перед выключением компьютера. Чтобы создать его, нужно выполнить несколько простых действий:
- 1. Откройте Блокнот и введите этот текст: (не забудьте заменить username именем пользователя, под которым вы заходите в систему, а C:\Temp – названием своей папки с временными файлами).
- RD /S /q "C:\Documents and Settings\USERNAME\Local Settings\History" RD /S /q "C:\Documents and Settings\Default User\Local Settings\History" RD /S /q "C:\Temp\"
- 2. Сохраните этот файл на вашем диске C: под именем deltemp.bat. 3. Нажмите «Пуск->Выполнить». Введите gpedit.msc. В левой части окна «Групповая политика» выберите пункт «Конфигурация компьютера - Конфигурация Windows - Сценарии (запуск/завершение)» и выберите в правой части окна пункт «Завершение работы». В появившемся окне нажмите кнопку «Добавить» и укажите, где находится созданный вами файл. Теперь он будет запускаться перед каждым выключением компьютера.
- Естественно, здесь описаны только основные ключи реестра, которые относятся к безопасности ваших данных и доступу к ресурсам вашего компьютера.

2.6 Операционные системы семейства MS Windows

2.6.1 Этапы развития:

1) интерфейсные системы: Windows3.0 , 3.1 , 3.11
2) ОС Windows - 95, -98, -2000, -ME, -XP.

Характерные черты:

Многозадачные, многопотоковые ОС, которые обеспечивают управление сразу несколькими программами и каждая их этих программ может иметь несколько параллельных потоков (то есть независимо исполняемых частей).

Особенности организации работы:

Главным окном является рабочий стол, который содержит пиктограммы папок.

Папка – хранилище для файлов, каталогов, магнитных дисков, очереди заданий на печать и другие папки. Для отображения содержимого папок используется 2 типа окон:

- окно папки;
- окно "проводника"

"Проводник" – приложение, которое обеспечивает отображение файловой структуры и выполнение команд.

"Рабочий стол" – вершина иерархии всех папок. Имеет объектно-ориентированные свойства(если кликнуть правой кнопкой мыши на любую пиктограмму, то появится контекстное меню с указанием возможных действий.

WINDOWS NT - мультипрограммная ОС обеспечивает управление сразу несколькими программами, поддерживает работу как 32- разрядных приложений, для которых используется приоритетная многозадачность поддерживаемая средствами Windows 32, а также позволяет запускать 16 разрядные DOS- приложения.

Windows NT поддерживает четырехуровневые кольца защиты оперативной памяти:

- в 3-ем кольце размещаются прикладные программы и динамические компонуемые библиотеки, содержащие сервисные функции ОС;
- в 0-ом кольце – ядро ОС;
- в 1-ом кольце системные программы и драйверы;
- во 2-ом сервисные программы и драйверы.
- При такой организации критически важны компоненты ОС изолируется от прикладных программ, что обеспечивает большую стабильность ОС.

В кольце 3 размещены виртуальные машины, на которых исполняются прикладные программы.

Все программы Windows 16 разрядные работают каждая на своей виртуальной машине. 16-разрядные программы работают на виртуальной DOS Машине (VDM), причем все загруженные программы могут передавать данные через буфер обмена друг другу, а также приложениям Windows .

2.6.2 Как Windows выполняет программный код

Операционная система Windows для поддержки своей эффективности и целостности использует два режима: пользователя и ядра. Архитектура процессора Intel 80386 и следующих моделей определяет четыре уровня привилегий, называемых кольцами, для защиты кода и данных системы от случайного или преднамеренного изменения со стороны менее привилегированного кода. Такой метод выполнения кода называется моделью защиты Intel.

Уровень привилегий 0, известный *как режим ядра,* максимальный. Уровень привилегий 3, или *режим пользователя,* — минимальный. Когда код выполняется на некотором уровне привилегий, говорят, что он выполняется в соответствующем кольце. Операционные системы семейства Windows используют только кольца 0 и 3 (рис. 2.1).

Рис. 2.1 Кольца 0 и 3 в модели защиты Intel

Режим ядра Режим ядра (кольцо 0) — это наиболее привилегированный режим. Работая в нем, код имеет прямой доступ ко всей аппаратуре и всему адресному пространству.

Программное обеспечение, выполняющееся в режиме ядра:

- имеет прямой доступ к аппаратному обеспечению;
- имеет доступ ко всей памяти компьютера;
- не может быть вытеснено в страничный файл на жестком диске;
- выполняется с большим приоритетом, чем процессы режима пользователя.

В частности, в кольце 0 выполняется код ядра операционных систем Windows 95 и Windows NT. Поскольку компоненты режима ядра защищены архитектурно, процессор предотвращает их изменение другой программой. Хотя кольцо 0 предоставляет максимальную защиту, не следует запускать в кольце 0 что попало — ведь компоненты этого режима имеют доступ ко всей системе. Если программный компонент в режиме ядра потерпит крах, это может разрушить всю систему.

Поскольку одна из задач Windows 95 — максимальная обратная совместимость, многие старые 16-разрядные драйверы и приложения используют прямой доступ к аппаратуре. Windows NT не предоставляет таким приложениям требуемый уровень доступа, поэтому зачастую они не могут работать под управлением Windows NT Workstation и Windows NT Server.

Режим пользователя

Режим пользователя предоставляет меньше привилегий, нежели режим ядра, — в частности, он не обеспечивает прямой доступ к аппаратуре. Код, выполняющийся в кольце 3, ограничен выделенным ему адресным пространством, а для вызова системных сервисов использует интерфейс прикладного программирования (API) Windows.

Процессы режима пользователя характеризуются следующими особенностями.

- Не имеют прямого доступа к аппаратуре.

  Это защищает систему от неисправных приложений или неавторизованного доступа.

- Ограничены выделенным им адресным пространством.

  Таким образом операционная система обеспечивает свою целостность. Процессу выделяется определенная область адресов и запрещено выходить за эту область.

- Могут быть вытеснены из физической памяти в виртуальную память на жестком диске.

  Механизм виртуальной памяти позволяет использовать пространство на жестком диске как дополнительное ОЗУ. О виртуальной памяти подробно рассказано чуть позже в этой главе.

- Выполняются с меньшим приоритетом, чем компоненты режима ядра.

  Поскольку приоритет процессов режима пользователя ниже, они получают меньший доступ к процессору, чем процессы режима ядра. Это гарантирует, что операционная система не будет ожидать окончания работы такого процесса. Кроме того, неисправный программный компонент, выполняющийся в режиме пользователя, не вызовет крах всей системы и не повлияет на другие приложения, работающие параллельно.

2.6.3 Многозадачность

Многозадачность — способность операционной системы выполнять более одной программы (задачи) одновременно. Противоположный подход — однозадачность, когда один процесс должен быть завершен прежде, чем сможет начаться другой. MS-DOS — пример однозадачной среды, а Windows 95 и Windows NT — многозадачные среды.

Конечно же, и в многозадачной системе программы не выполняются одновременно — процессор переключается между ними. Благодаря этому Вы можете выполнить запрос к базе данных и продолжить работу с редактором текстов до тех пор, пока не появятся результаты запроса. Многозадачность, кроме того, позволяет компьютеру эффективно использовать время, которое иначе было бы потеряно в ожидании команды пользователя или ответа устройств ввода/вывода. Для понимания многозадачности необходимо сначала познакомиться с процессами и потоками.

Процессы и потоки

Приложение, разработанное для Windows, состоит из одного или более процессов (рис. 2.2). Процесс — это, попросту говоря, выполняемая программа. Ему принадлежат адресное пространство и выделенные ресурсы, а также один или более потоков, выполняющихся в его контексте.

Поток — это основная единица, которой операционная система выделяет процессорное время, и минимальный «квант» кода, который может быть запланирован для выполнения. Кроме того, это часть процесса, выполняющаяся в каждый момент времени. Поток работает в адресном пространстве процесса и использует ресурсы, выделенные процессу.

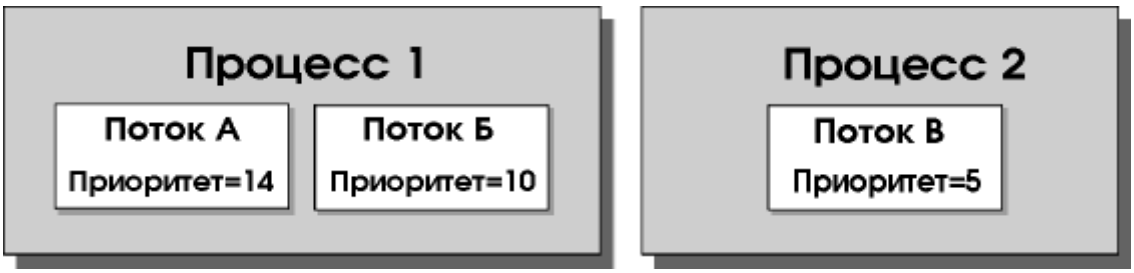


Рис. 2.2 Приоритеты потоков

Любой процесс содержит хотя бы один поток. Каждый процесс 16-разрядного Windows-приложения или программы MS-DOS имеет только один поток, тогда как процессы 32-разрядных Windows-приложений могут включать их несколько.

Примечание Ресурсами владеют процессы, а не потоки — последние только используют ресурсы, выделенные процессу. Например, если программа запросит порт, им будет управлять процесс. Любой поток процесса может обратиться к порту, но ни один из них не вправе самостоятельно запросить использование порта.

2.6.4 Вытесняющая и кооперативная многозадачность

Существуют два типа многозадачности: кооперативная (не вытесняющая) и вытесняющая (рис. 2.3). В кооперативной многозадачной среде (например, Windows 3.1) контроль над процессором никогда не отбирается у задачи — приложение должно самостоятельно отказаться от контроля над процессором, чтобы другое приложение заработало.

Вытесняющая многозадачность отличается от кооперативной тем, что операционная система может получить контроль над процессором без согласия выполняющегося приложения. Лишение приложения контроля над процессором называется *вытеснением.* Windows 95 и Windows NT используют вытесняющую многозадачность для MS-DOS и 32-разрядных Windows-приложений.

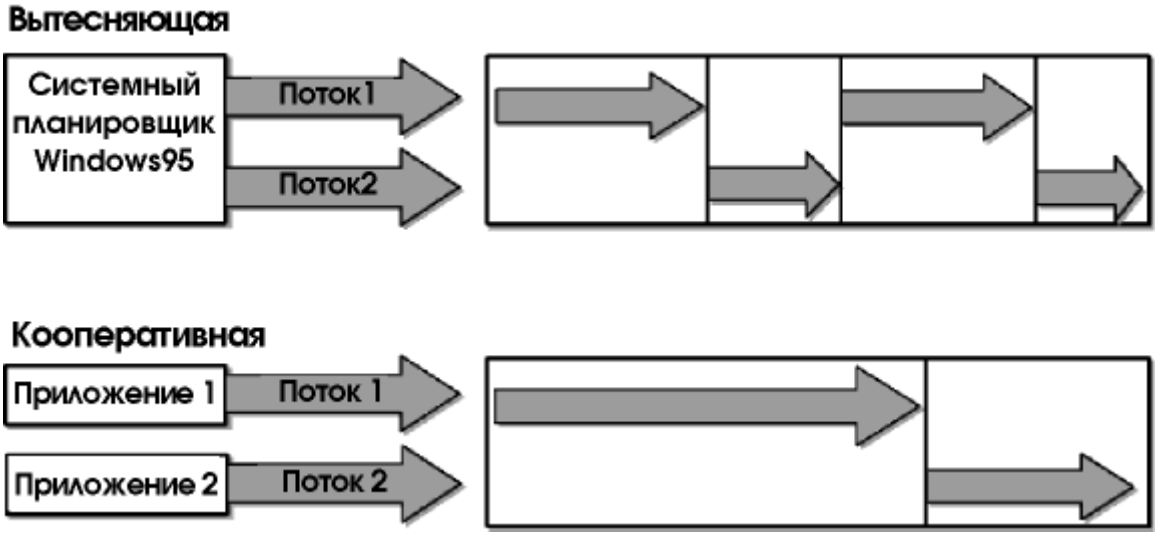


Рис.2.3 Вытесняющая и кооперативная многозадачность

Разработчики программ, выполняющихся под управлением кооперативной операционной системы, должны учитывать необходимость частого возврата управления процессором операционной системе. Программа, которая недостаточно часто отдает управление, блокирует кооперативную операционную систему.

Windows NT применяет вытесняющую многозадачность при выполнении 16-разрядных приложений Windows и MS-DOS. Windows NT обеспечивает полную защиту памяти 16-разрядных приложений, так как каждое из них выполняется в рамках собственной виртуальной машины. Windows 95, напротив, использует кооперативную многозадачность для всех 16-разрядных приложений — это необходимо для сохранения совместимости с 16-разрядными Windows-программами, которые сами контролируют свое выполнение.

Планирование

С помощью планирования операционная система определяет, какой поток использует процессор в данный момент времени. Windows реализует многозадачность, присваивая каждому потоку приоритет, что позволяет ему использовать ресурсы системы. Планирование основано на заранее заданной единице времени, называемой *квантом.* Фактическая продолжительность кванта времени зависит от конфигурации системы. Уровни приоритета находятся в диапазоне от 0 (наименьший приоритет) до 31 (наибольший приоритет). Поток с наибольшим приоритетом получает процессор в свое распоряжение (рис. 2.4).

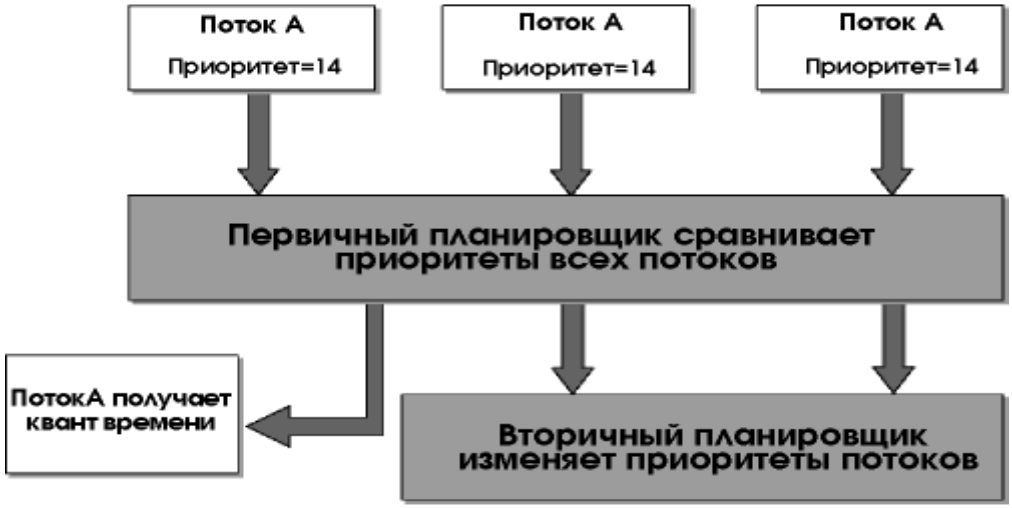


Рис. 2.4 Процесс планирования

Приоритет каждого потока определяется по:

- классу приоритета процесса, которому принадлежит поток;
- уровню приоритета потока внутри класса приоритета его процесса.

Классы приоритетов

Класс приоритета процесса и уровень приоритета потока определяют базовый приоритет потока. Уровни приоритетов Windows разделены на два класса:

- реального времени (приоритеты от 16 до 31) — используется для выполнения основных функций операционной системы и обычно не применяется для приложений;
- переменного приоритета (приоритет от 0 до 15) — определяет процессорный приоритет приложений; приоритет 0 возможен только для бесстраничного системного потока.

Уровни приоритетов

Процессам могут быть присвоены следующие базовые уровни приоритетов:

- низкий — запускает приложения с уровнем приоритета 4;
- обычный — запускает приложения с уровнем приоритета 7;
- высокий — запускает приложения с уровнем приоритета 13;
- реального времени — запускает приложения с уровнем приоритета 24.

Примечание: Не запускайте приложения с классом приоритета реального времени — это может привести к нестабильности в работе операционной системы.

2.6.5 Управление памятью

В Windows 95 и NT каждый процесс имеет собственное адресное пространство, что позволяет адресовать до 4 Гб памяти. Отметим, что Windows выделяет процессу 4 Гб адресов памяти, а не физического ОЗУ. Физическая память ограничена имеющимися системными ресурсами (ОЗУ и дисковое пространство). Windows выделяет каждому приложению 2 Гб адресов памяти, а другие 2 Гб резервируются для нужд ядра.

Большинство компьютеров не располагают 4 Гб ОЗУ, и по этой причине Windows использует механизм *виртуальной памяти.* Таким образом, Windows может перенести часть содержимого физической памяти на жесткий диск, когда объем доступного ОЗУ будет исчерпан. Этот процесс известен как *подкачка* (рис. 1.6).

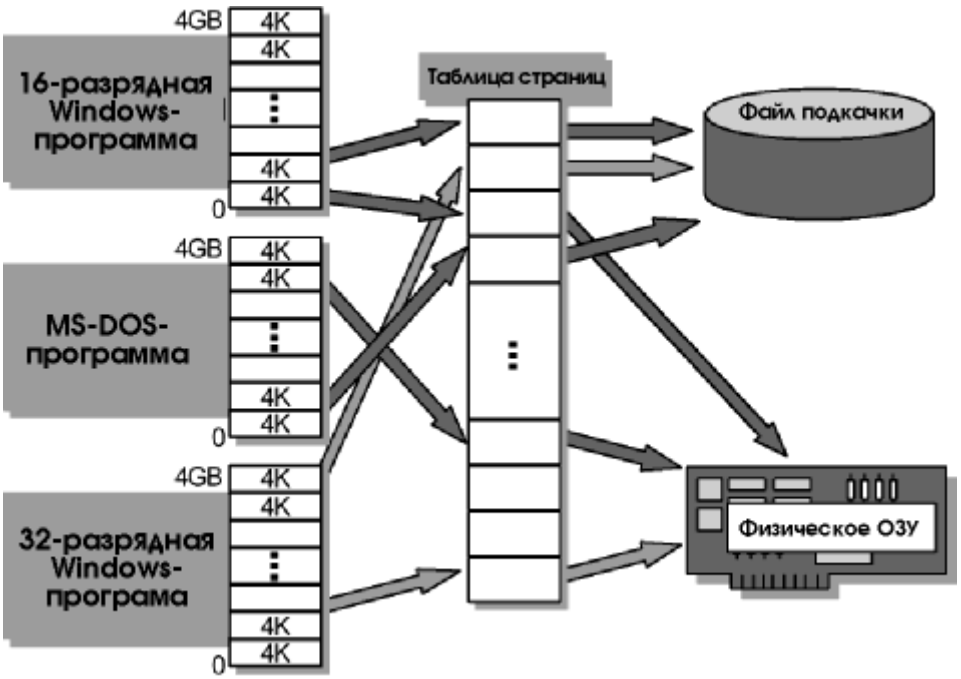


Рис. 2.5 Выделение виртуальной памяти приложениям

Виртуальные адреса, используемые процессом, не совпадают с адресами физической памяти. Для каждого процесса ядро поддерживает так называемую *таблицу страниц* — внутреннюю структуру, которая позволяет преобразовать виртуальные адреса в физические.

Виртуальная память

Процессоры Intel, начиная с модели 80386, позволяют отобразить область физической памяти на любую область 32-разрядных адресов. Виртуальная память Windows использует этот механизм для того, чтобы любая программа вела себя так, будто она имеет собственное физическое ОЗУ.

Windows для доступа к памяти применяет 32-разрядную линейную адресацию: приложения обращаются к памяти с помощью 32-разрядных адресов. Каждая программа имеет собственное виртуальное адресное пространство, которое диспетчер виртуальной памяти преобразует в адреса физического ОЗУ или в файле на жестком диске.

Постраничная подкачка Физическое и виртуальное (логическое) адресное пространство каждого процесса разделено на страницы — «кванты» памяти, размер которых зависит от компьютера. Например, для компьютеров x86 размер страницы составляет 4 кб. Ядро может перемещать страницы памяти в страничный файл на диске (Pagefile.sys) и обратно: таким образом, управление памятью становится более гибким. Когда страница перемещается в физическую память, ядро обновляет таблицы страниц соответствующих процессов. Если ядру требуется место в физической памяти, оно вытесняет самые старые страницы физической памяти в страничный файл. Манипуляции ядра с физической памятью совершенно незаметны (прозрачны) для приложений, которые работают только со своими виртуальными адресными пространствами.

2.6.6 Выполнение приложений

Windows 95 и Windows NT по-разному выполняют приложения, особенно 16-разрядные.

Механизм сообщений Windows В отличие от MS-DOS, Windows для управления приложениями использует модель сообщений. Сообщение генерируется всякий раз, когда происходит какое-то событие, например пользователь нажимает и отпускает клавишу на клавиатуре или передвигает мышь. Сообщение помещается в так называемую *очередь сообщений.* Активное приложение постоянно проверяет свою очередь и извлекает из нее поступившие сообщения.

Рис.2.6 Очереди сообщении

Обмен сообщениями в 16-разрядных версиях Windows

В Windows 3.1 очередь сообщений операционной системы — единая. Она обслуживает все 16-разрядные Windows-приложения, которые проверяют ее и извлекают адресованные им сообщения. Такое решение не лишено недостатков — например, если у какого-то приложения возникнут проблемы, оно может не позволить другим приложениям проверить очередь сообщений. В этом случае последние прекратят работу, пока не получат управление и не смогут проверить наличие адресованных им сообщений.

Обмен сообщениями в Windows 95

В Windows 95 проблемы единой очереди сообщений разрешены: у каждого выполняющегося Win32-приложения — своя очередь. Каждый поток в Win32-приложении имеет собственную очередь сообщений и, значит, никак не влияет на поведение других работающих приложений. Если Winl6- или Win32-приложение потерпит крах, остальные Win32-приложения будут действовать на основе вытесняющей многозадачности и смогут принимать поступающие сообщения из своих очередей.

Тем не менее в целях совместимости все 16-разрядные Windows-приложения под управлением Windows 95 используют общую очередь сообщений. Очевидно, если с одним из них что-то произойдет, остальным будет перекрыт доступ к очереди до тех пор, пока программа, вызвавшая проблему, не будет завершена.

Виртуальные машины

Windows NT выполняет приложения в рамках виртуальных машин (Virtual Machine, VM). Фактически VM — это создаваемая операционной системой среда для выполнения

приложения, которая полностью эмулирует все ресурсы компьютера. С точки зрения приложения, виртуальная машина — это полноценный компьютер, предоставляющий ему все имеющиеся ресурсы.

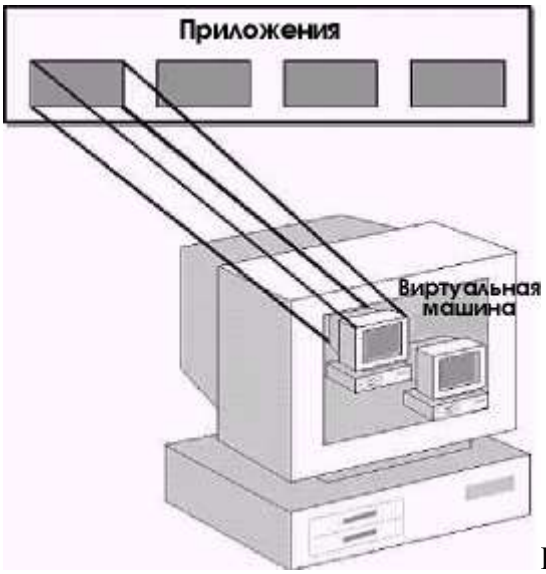


Рис. 2.7 Виртуальные машины

Каждое 16-разрядное Windows- и MS-DOS-приложение под управлением Windows NT выполняется в отдельном адресном пространстве, называемом виртуальной DOS-машиной (Virtual DOS Machine, VDM). При этом обеспечивается защита программы, а Windows NT может реализовать вытесняющую многозадачность для всех сервисов операционной системы и приложений.

В Windows 95 предусмотрено выполнение MS-DOS-приложений в отдельных VDM, однако, поскольку часть памяти доступна всем виртуальным машинам, MS-DOS-приложения представляют собой потенциальную угрозу стабильности системы.

2.6.7 Драйверы устройств в Windows

Драйвер устройства — это программный компонент, получающий команды от операционной системы и преобразующий их в команды конкретного устройства (рис. 2.8). Зачастую драйверы разрабатываются производителями аппаратного обеспечения, и компания Microsoft их напрямую не поддерживает.

Рис. 2.8 Архитектура драйверов устройств

Драйверы устройств позволяют разработчикам создавать аппаратно-независимые приложения. Другими словами, разработчику на стадии создания приложения не нужно заботиться о том, какую именно аппаратуру будет применять пользователь. Пользователь, в свою очередь, может корректировать конфигурацию компьютера, не затрагивая работоспособность приложений. Windows использует драйверы для таких компонентов, как:

- дисплеи;
- звуковые карты;
- устройства связи;
- принтеры;
- сетевые адаптеры.

В зависимости от того, для какой операционной системы семейства Windows разработан драйвер, он может принадлежать к одной из двух групп: защищенного режима и реального режима.

Драйверы защищенного и реального режима

Драйверы реального режима созданы для работы в реальном режиме операционной системы MS-DOS. Они не так безопасны и устойчивы, как драйверы защищенного режима, которые используют преимущества архитектуры защищенного режима процессоров 80386 и последующих моделей. Драйвер защищенного режима или драйвер виртуального устройства (Virtual Device Driver, VxD) обеспечивает быстрый разделяемый доступ к устройству. Кроме того, операционные системы семейства Windows выполняют 32-разрядный код защищенного режима более эффективно, чем 16-разрядный код реального режима.

Windows 95 поддерживает оба типа драйверов, а Windows NT — только драйверы защищенного режима. Компания Microsoft настоятельно рекомендует применять 32-разрядные драйверы защищенного режима везде, где возможно.

2.6.8 Интерфейс прикладного программирования Win32

Интерфейс прикладного программирования (Application Programming Interface, API) Win32 обеспечивает приложениям доступ ко всему спектру функций операционных систем семейства Windows. Функции, сообщения и структуры Win32 формируют последовательный и единообразный **API** для Windows 95 и Windows NT. Средства API Win32 позволяют разрабатывать приложения, успешно работающие на всех платформах и в то же время при надобности использующие уникальные особенности любой из них.

Многие функции API интегрированы в состав таких программ, как Visual Basic. Например, функцию API Win32 **MessageBox** можно вызвать непосредственно или через функцию Visual Basic **MsgBox.** Средствами Visual Basic обычно пользоваться легче, однако во многих случаях разработчики найдут непосредственное применение и самим функциям API Win32.

Основной код API Win32

Базовый код API Win32 содержится в трех библиотеках динамической загрузки (Dynamic Link Library, DLL): USER32, GDI32 и KERNEL32.

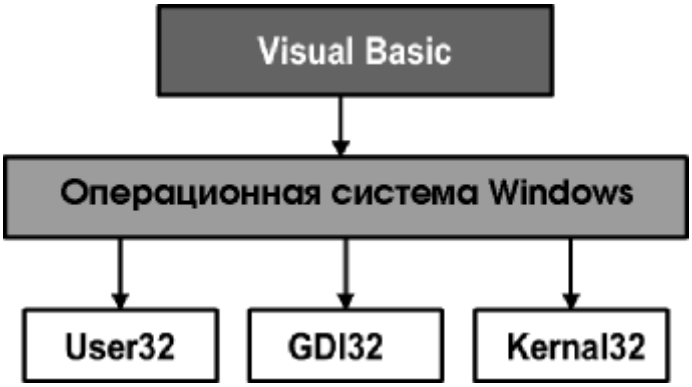


Рис.2.9 Библиотеки API Win32

USER32

User32.dll и User.exe создают и контролируют окна на экране, выполняя все запросы по созданию, перемещению, изменению размеров и уничтожению окон. **User.exe,** кроме того, обрабатывает запросы, относящиеся к значкам и другим элементам интерфейса пользователя, а также переадресует события, порожденные различными устройствами ввода, соответствующим приложениям.

GD132

Gdi32.dll и Gdi.exe контролируют интерфейс графических устройств (Graphics Device Interface, GDI). GDI выполняет графические операции при создании изображения на системном дисплее и других устройствах, включая:

- вывод на экран;
- вывод на принтер;
- включение/отключение пикселов.

KERNEL32 Kernel32.dll выполняет базовые функции операционной системы, в том числе:

- управление памятью;
- файловый ввод/вывод;
- загрузку программы;
- выполнение программы.

Примечание При объявлении функций API в Visual Basic 32-разрядные функции чувствительны к регистру символов, а эквивалентные 16-разрядные функции — нет. Это необходимо иметь в виду при преобразовании 16-разрядных приложений в 32-разрядные.

***32*- и 16-разрядные компоненты**

В Windows 95 включены 16-разрядные версии User, GDI и Kernel. Комбинация 16-разрядного и 32-разрядного кода позволяет сохранить совместимость с существующими

приложениями и драйверами и одновременно увеличить производительность системы по сравнению с Windows 3.1. Windows 95 использует 32-разрядный код везде, где это увеличивает производительность не в ущерб совместимости. Для включения в Windows 95 16-разрядных компонентов есть три основные причины:

- код для 16-разрядных систем обеспечивает обратную совместимость с приложениями и драйверами, разработанными для Windows 3.1;
- в некоторых случаях 16-разрядный код выполняется быстрее, чем аналогичный 32-разрядный;
- 32-разрядный код требует больше памяти, чем эквивалентный 16-разрядный.

Одна из основных задач Windows 95 — эффективная работа на компьютерах с ограниченным объемом ОЗУ, и применение 16-разрядного кода способствует решению этой задачи.

Подсистемы ввода/вывода и драйверы устройств, включая сетевые и файловые системы, являются полностью 32-разрядными, как и все компоненты управления памятью и планирования. Часто возникающая при этом проблема вызова 32-разрядной функции из 16-разрядного приложения (или наоборот) решается при помощи *шлюзования.*

Шлюзование

Эта операция выполняется, когда операционная система преобразует вызов 16-разрядной функции в вызов 32-разрядной. Процессы Windows 95 и Windows NT не могут содержать одновременно и 16-разрядный, и 32-разрядный код. Шлюз позволяет коду с одной стороны границы вызывать код с другой ее стороны. Каждая платформа использует один или несколько механизмов шлюзования:

- механизм *базовых шлюзов* позволяет 16-разрядному Windows-приложению в системе под управлением Windows 95 и Windows NT загрузить и вызвать 32-разрядную библиотеку;
- с помощью механизма *плоских шлюзов,* реализованного только в Windows NT, Win32-приложение загружает и вызывает 16-разрядную библиотеку и наоборот.

2.6.9 Реестр Windows

Реестр — это унифицированная база данных, содержащая информацию об аппаратной и программной конфигурации локального компьютера. Здесь же хранятся данные системы и приложений. Реестр выполняет ту же роль, что и INI-файлы в Windows 3.1. Windows 95 по-прежнему позволяет использовать INI-файлы, однако в первую очередь они обеспечивают обратную совместимость. Преимущества реестра — возможность присоединить к одному ключу множество элементов различных типов. Кроме того, сетевые средства обеспечивают доступ к реестру по сети для удаленного администрирования и диагностики.

Редактор реестра

В Windows 95 и Windows NT реестр можно просматривать и редактировать средствами редактора реестра REGEDIT.EXE, расположенного в папке Windows (рис. 2.10).

Будьте осторожны, изменяя элементы реестра при помощи редактора реестра, — он не распознает синтаксические и семантические ошибки и не предупреждает о создании некорректного элемента. Неверный элемент реестра может сделать систему неработоспособной. К счастью, в большинстве случаев Вам не придется модифицировать реестр напрямую — для изменения параметров системы чаще всего достаточно диспетчера устройств и других апплетов Панели управления.

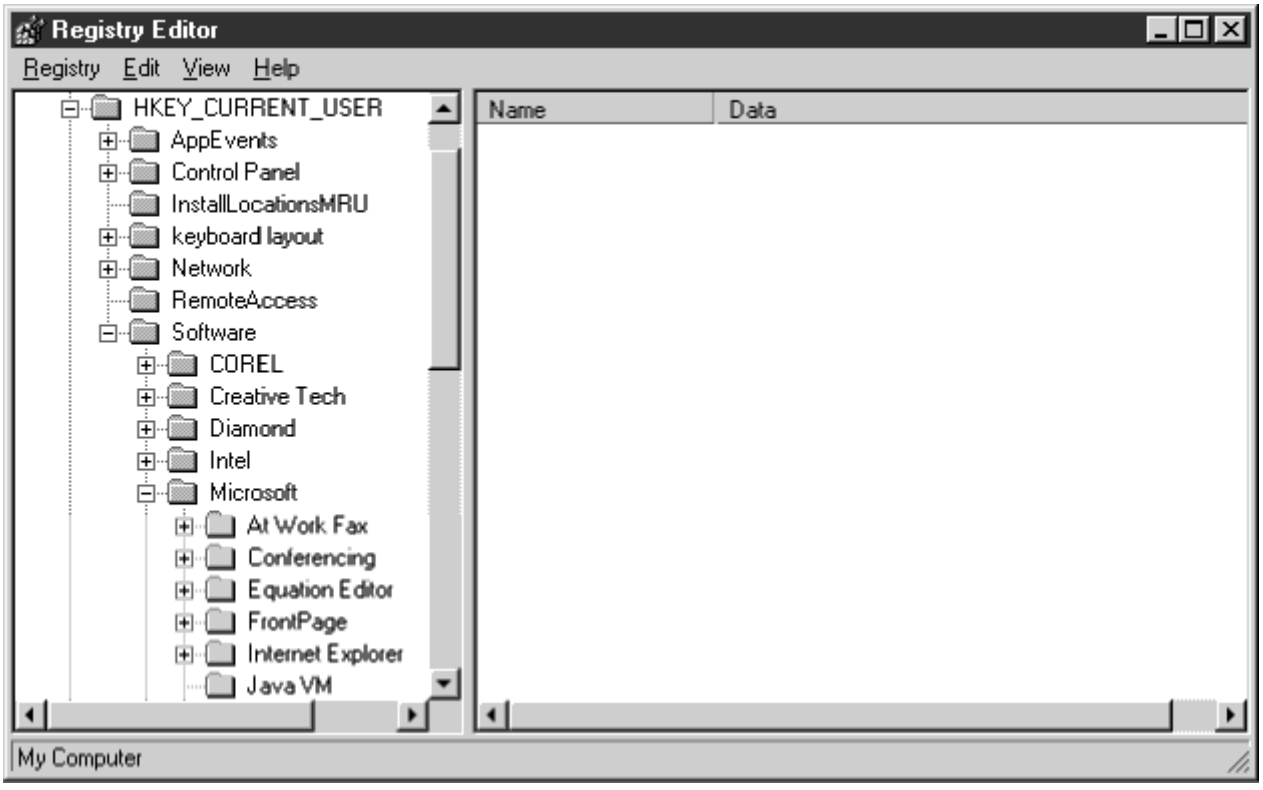


Рис. 2.10 Редактор реестра Windows

Структура реестра

Реестр — это древовидная иерархическая база данных. Он хранится в двух файлах, состав которых определяется конфигурацией системы. Обычно их два: один содержит настройки, специфичные для пользователя (файл USER.DAT), а другой — настройки, специфичные

для компьютера (обычно SYSTEM.DAT). Каждый узел иерархического дерева называется *ключом.* Реестр напоминает файловую систему: любой ключ может содержать вложенные ключи (аналог вложенных каталогов) и данные (аналог файлов). В ключе хранится произвольное число значений данных любого типа. Каждое значение называется *элементом реестра.* Компоненты ключей реестра перечислены ниже.

Таблица 2.5.1

Компонент ключа	Обязательный	Описание
Имя	Да	Строка, используемая для доступа к ключу. Должна быть уникальной среди других ключей того же уровня иерархии
Класс	Нет	Имя класса объекта. Предназначен для использования в коде методов класса, экземпляры которого хранятся в реестре, Приложениями обычно не используется
Дескриптор защиты	Нет	Ключи содержат стандартные дескрипторы защиты Windows NT, допускают управление доступом и могут быть подвергнуты аудиту
Время последней записи	Нет	Время, когда ключ был последний раз модифицирован. Любое изменение элемента считается изменением его родительского ключа
Элемент(ы)	Нет	Информация, хранящаяся в ключе: имя для идентификации значения, тип для определения типа данных и сами данные соответствующей длины и формата

HKEY CLASSES ROOT : Ключ HKEY_CLASSES_ROOT содержит те же данные, что и файл REG.DAT в Windows 3.1,— сведения о встраивании и связывании объектов (Object Linking and Embedding, OLE) и ассоциации файлов с приложениями, которые позволяют Windows запускать приложение, соответствующее выбранному файлу.

HKEY_LOCAL_MACHINE: Этот ключ содержит спецификации рабочей станции, драйверов и другие системные настройки, включая информацию о типах установленного оборудования, настройках портов, конфигурации программного обеспечения и т.п. Эта информация специфична для компьютера, а не для пользователя.

HKEY_CURRENT.CONFIG: Этот ключ содержит информацию о текущей конфигурации аппаратуры компьютера и используется в основном на компьютерах с несколькими аппаратными конфигурациями, например при подключении портативного компьютера к стыковочной станции и отключении от нее. Информация, содержащаяся в этом ключе, копируется из ключа HKEY_LOCAL_MACHINE.

HKEY_USERS: Это ключ содержит информацию обо всех пользователях данной рабочей станции. Здесь хранятся данные о каждом пользователе, а также типовые настройки, служащие шаблоном для новых ключей, создаваемых пользователем. Типовые настройки включают различные значения по умолчанию для программ, схем событий, конфигураций рабочего стола и т.п.

HKEY_CURRENT_USER: Этот ключ содержит настройки системы и программ, относящиеся к текущему пользователю. Он создается при регистрации пользователя в системе на основе информации из соответствующего раздела ключа HKEY_USERS. Именно здесь хранятся сведения о том, как данный пользователь сконфигурировал рабочую станцию — например, данные том, что каждый старт системы должен сопровождаться звуковым эффектом. Прочая информация может включать цветовые схемы, ярлыки, состояние рабочего стола и т.п.

HKEY_DYN_DATA: Этот ключ содержит динамическую информацию о состоянии различных устройств, причем она создается заново при каждом старте системы. Ключ HKEY_DYN_DATA используется как часть системы измерения производительности и для конфигурации устройств Plug-and-Play. Информация, содержащаяся здесь, меняется при добавлении новых устройств и удалении существующих. Для каждого устройства это данные о соответствующем аппаратном ключе, известных проблемах и текущем состоянии устройства. Ключ HKEY_DYN_DATA **также** содержит сведения о состоянии системы, формируемые с помощью утилиты System Monitor. Это ключ не входит в состав файлов реестра и всегда создается динамически.

Модификация реестра Windows

Работать с приложением гораздо приятнее, если при каждом запуске оно сохраняет информацию о действиях и предпочтениях пользователя. Эти данные можно применять и далее — например, сохранить имя последней базы данных, открытой пользователем, и указать его в качестве имени по умолчанию, когда пользователь обратится к базе данных в следующий раз.

Для сохранения параметров приложения в реестре применяются операторы Visual Basic **SaveSetting и GetSetting** и соответствующие функции API Windows. Они имеют следующий синтаксис:

- **SaveSetting** *(приложение, раздел, ключ, параметр)*
- **GetSetting** *(приложение, раздел, ключ, [по умолчанию])*

Пример В приведенном ниже коде с помощью оператора **SaveSetting** создаются элементы реестра для приложения, заданного в аргументе приложение, а затем применяется оператор **GetSetting** для получения значений параметров. Поскольку задано значение по умолчанию, возврат значения гарантирован.

Поместить настройки в реестр

SaveSetting "MyApp","Startup", "Top", 75

SaveSetting "MyApp","Startup", "Left", 50

использовать настройки из реестра для отображения текущей формы

```
Me.Left = GetSetting(appname := "MyApp",
        section := "Startup", key := "Left", default := "0")
Me.Top = GetSetting(appname := "MyApp",
        section := "Startup", key := "Top", default := "0")
```

Резюме

Изучение операционной системы Windows следует начинать с архитектуры системы. Операционная система Windows для поддержки своей эффективности и целостности использует два режима: пользователя и ядра.

Windows 95 и Windows NT — многозадачные операционные системы. Однако в действительности одновременно не выполняется больше одного процесса — процессор переключается между ними. В Windows 95 и NT каждый процесс имеет собственное адресное пространство, что позволяет адресовать до 4 Гб памяти.

Интерфейс прикладного программирования (API) Win32 обеспечивает приложениям доступ ко всему спектру функций операционных систем семейства Windows. Функции, сообщения и структуры Win32 образуют последовательный и единообразный API для Windows 95 и Windows NT. Средства API Win32 позволяют разрабатывать приложения, которые успешно работают на всех платформах, в то же время сохраняя возможность использовать уникальные особенности любой из них.

Реестр — это унифицированная база данных, где хранится информация об аппаратной и программной конфигурации локального компьютера. Преимущества реестра — возможность присоединить к одному ключу множество элементов различных типов. Кроме того, сетевые средства обеспечивают доступ к реестру по сети для удаленного администрирования и диагностики.

2.7 Идентификация и аутентификация

2.7.1.1 Основные понятия

Идентификацию и *аутентификацию* можно считать основой программно-технических средств безопасности, поскольку остальные сервисы рассчитаны на обслуживание именованных субъектов. *Идентификация* и *аутентификация* - это первая линия обороны, "проходная" информационного пространства организации.

Идентификация позволяет субъекту (пользователю, процессу, действующему от имени определенного пользователя, или иному аппаратно-программному компоненту) назвать себя (сообщить свое имя). Посредством ***аутентификации*** вторая сторона убеждается, что субъект действительно тот, за кого он себя выдает. В качестве синонима слова "*аутентификация*" иногда используют словосочетание "проверка подлинности".

(Заметим в скобках, что происхождение русскоязычного термина "*аутентификация*" не совсем понятно. Английское "authentication" скорее можно прочитать как "аутентикация"; трудно сказать, откуда в середине взялось еще "фи" - может, из *идентификации*? Тем не менее, термин устоялся, он закреплен в Руководящих документах Гостехкомиссии России, использован в многочисленных публикациях, поэтому исправить его уже невозможно.)

Аутентификация бывает ***односторонней*** (обычно клиент доказывает свою подлинность серверу) и ***двусторонней*** (***взаимной***). Пример *односторонней аутентификации* - процедура входа пользователя в систему.

В сетевой среде, когда стороны *идентификации*/*аутентификации* территориально разнесены, у рассматриваемого сервиса есть два основных аспекта:

- что служит ***аутентификатором*** (то есть используется для подтверждения подлинности субъекта);
- как организован (и защищен) обмен данными *идентификации*/*аутентификации*.

Субъект может подтвердить свою подлинность, предъявив по крайней мере одну из следующих сущностей:

- нечто, что он знает (пароль, личный *идентификационный* номер, криптографический ключ и т.п.);
- нечто, чем он владеет (личную карточку или иное устройство аналогичного назначения);
- нечто, что есть часть его самого (голос, отпечатки пальцев и т.п., то есть свои биометрические характеристики).

В открытой сетевой среде между сторонами *идентификации*/*аутентификации* не существует доверенного маршрута; это значит, что в общем случае данные, переданные субъектом, могут не совпадать с данными, полученными и использованными для проверки подлинности. Необходимо обеспечить защиту от пассивного и активного прослушивания сети, то есть от ***перехвата, изменения*** и/или ***воспроизведения*** данных. Передача паролей в открытом виде, очевидно, неудовлетворительна; не спасает положение и шифрование паролей, так как оно не защищает от *воспроизведения*. Нужны более сложные протоколы *аутентификации*.

Надежная *идентификация* и затруднена не только из-за сетевых угроз, но и по целому ряду причин. Во-первых, почти все *аутентификационные* сущности можно узнать, украсть или подделать. Во-вторых, имеется противоречие между надежностью *аутентификации*, с одной стороны, и удобствами пользователя и системного

администратора с другой. Так, из соображений безопасности необходимо с определенной частотой просить пользователя повторно вводить *аутентификационную* информацию (ведь на его место мог сесть другой человек), а это не только хлопотно, но и повышает вероятность того, что кто-то может подсмотреть за вводом данных. В-третьих, чем надежнее средство защиты, тем оно дороже.

Современные средства *идентификации*/*аутентификации* должны поддерживать концепцию *единого входа в сеть*. *Единый вход в сеть* - это, в первую очередь, требование удобства для пользователей. Если в корпоративной сети много информационных сервисов, допускающих независимое обращение, то многократная *идентификация*/*аутентификация* становится слишком обременительной. К сожалению, пока нельзя сказать, что *единый вход в сеть* стал нормой, доминирующие решения пока не сформировались.

Таким образом, необходимо искать компромисс между надежностью, доступностью по цене и удобством использования и администрирования средств *идентификации* и *аутентификации*.

Любопытно отметить, что сервис *идентификации* / *аутентификации* может стать объектом атак на доступность. Если система сконфигурирована так, что после определенного числа неудачных попыток устройство ввода идентификационной информации (такое, например, как терминал) блокируется, то злоумышленник может остановить работу легального пользователя буквально несколькими нажатиями клавиш.

2.7.1.2 Парольная аутентификация

Главное достоинство ***парольной аутентификации*** - простота и привычность. Пароли давно встроены в операционные системы и иные сервисы. При правильном использовании пароли могут обеспечить приемлемый для многих организаций уровень безопасности. Тем не менее, по совокупности характеристик их следует признать самым слабым средством проверки подлинности.

Чтобы пароль был запоминающимся, его зачастую делают простым (имя подруги, название спортивной команды и т.п.). Однако простой пароль нетрудно угадать, особенно если знать пристрастия данного пользователя. Известна классическая история про советского разведчика Рихарда Зорге, объект внимания которого через слово говорил "карамба"; разумеется, этим же словом открывался сверхсекретный сейф.

Иногда пароли с самого начала не хранятся в тайне, так как имеют стандартные значения, указанные в документации, и далеко не всегда после установки системы производится их смена.

Ввод пароля можно подсмотреть. Иногда для подглядывания используются даже оптические приборы.

Пароли нередко сообщают коллегам, чтобы те могли, например, подменить на некоторое время владельца пароля. Теоретически в подобных случаях более правильно задействовать средства управления доступом, но на практике так никто не поступает; а тайна, которую знают двое, это уже не тайна.

Пароль можно угадать "методом грубой силы", используя, скажем, словарь. Если файл паролей зашифрован, но доступен для чтения, его можно скачать к себе на компьютер и попытаться подобрать пароль, запрограммировав полный перебор (предполагается, что алгоритм шифрования известен).

Тем не менее, следующие меры позволяют значительно повысить надежность парольной защиты:

- ***наложение технических ограничений*** (пароль должен быть не слишком коротким, он должен содержать буквы, цифры, знаки пунктуации и т.п.);
- ***управление сроком действия паролей***, их периодическая смена;
- ограничение доступа к файлу паролей;
- ограничение числа неудачных попыток входа в систему (это затруднит применение "метода грубой силы");
- обучение пользователей;
- использование программных ***генераторов паролей*** (такая программа, основываясь на несложных правилах, может порождать только благозвучные и, следовательно, запоминающиеся пароли).
- Перечисленные меры целесообразно применять всегда, даже если наряду с паролями используются другие методы аутентификации.

2.7.1.3 Одноразовые пароли

Рассмотренные выше **пароли можно назвать многоразовыми**; их раскрытие позволяет злоумышленнику действовать от имени легального пользователя. Гораздо более сильным средством, устойчивым к пассивному прослушиванию сети, являются **одноразовые пароли.**

Наиболее известным программным *генератором одноразовых паролей* является система **S/KEY** компании Bellcore. Идея этой системы состоит в следующем. Пусть имеется **односторонняя функция** f (то есть функция, вычислить обратную которой за приемлемое время не представляется возможным). Эта функция известна и пользователю, и **серверу аутентификации**. Пусть, далее, имеется **секретный ключ** K, известный только пользователю.

На этапе начального администрирования пользователя функция f применяется к ключу K n раз, после чего результат сохраняется на сервере. После этого процедура проверки подлинности пользователя выглядит следующим образом:

- сервер присылает на пользовательскую систему число (n-1);
- пользователь применяет функцию f к секретному ключу K (n-1) раз и отправляет результат по сети на сервер аутентификации;
- сервер применяет функцию f к полученному от пользователя значению и сравнивает результат с ранее сохраненной величиной. В случае совпадения подлинность пользователя считается установленной, сервер запоминает новое значение (присланное пользователем) и уменьшает на единицу счетчик (n).

На самом деле реализация устроена чуть сложнее (кроме счетчика, сервер посылает затравочное значение, используемое функцией f), но для нас сейчас это не важно. Поскольку функция f необратима, *перехват* пароля, равно как и получение доступа к серверу *аутентификации*, не позволяют узнать секретный ключ K и предсказать следующий одноразовый пароль.

Система S/KEY имеет статус Internet-стандарта (RFC 1938).

Другой подход к надежной *аутентификации* состоит в *генерации нового пароля* через небольшой промежуток времени (например, каждые 60 секунд), для чего могут использоваться программы или специальные интеллектуальные карты (с практической точки зрения такие пароли можно считать одноразовыми). Серверу *аутентификации* должен быть известен алгоритм *генерации паролей* и ассоциированные с ним параметры; кроме того, часы клиента и сервера должны быть синхронизированы.

2.7.1.4 Сервер аутентификации Kerberos

Kerberos - это программный продукт, разработанный в середине 1980-х годов в Массачусетском технологическом институте и претерпевший с тех пор ряд принципиальных изменений. Клиентские компоненты Kerberos присутствуют в большинстве современных операционных систем.

Kerberos предназначен для решения следующей задачи. Имеется открытая (незащищенная) сеть, в узлах которой сосредоточены **субъекты** - пользователи, а также клиентские и серверные программные системы. Каждый субъект обладает секретным ключом. Чтобы субъект C мог доказать свою подлинность субъекту S (без этого S не станет обслуживать C), он должен не только назвать себя, но и продемонстрировать знание секретного ключа. C не может просто послать S свой секретный ключ, во-первых, потому, что сеть открыта (доступна для пассивного и активного прослушивания), а, во-вторых, потому, что S не знает (и не должен знать) секретный ключ C. Требуется менее прямолинейный способ демонстрации знания секретного ключа.

Система Kerberos представляет собой **доверенную третью сторону** (то есть сторону, которой доверяют все), владеющую секретными ключами обслуживаемых субъектов и помогающую им в попарной проверке подлинности.

Чтобы с помощью Kerberos получить доступ к S (обычно это сервер), C (как правило - клиент) посылает Kerberos запрос, содержащий сведения о нем (клиенте) и о запрашиваемой услуге. В ответ Kerberos возвращает так называемый **билет,** зашифрованный секретным ключом сервера, и копию части информации из билета, зашифрованную секретным ключом клиента. Клиент должен расшифровать вторую порцию данных и переслать ее вместе с билетом серверу. Сервер, расшифровав билет, может сравнить его содержимое с дополнительной информацией, присланной клиентом. Совпадение свидетельствует о том, что клиент смог расшифровать предназначенные ему данные (ведь содержимое билета никому, кроме сервера и Kerberos, недоступно), то есть продемонстрировал знание секретного ключа. Значит, клиент - именно тот, за кого себя выдает. Подчеркнем, что секретные ключи в процессе проверки подлинности не передавались по сети (даже в зашифрованном виде) - они только использовались для шифрования. Как организован первоначальный обмен ключами между Kerberos и субъектами и как субъекты хранят свои секретные ключи - вопрос отдельный.

Проиллюстрируем описанную процедуру.

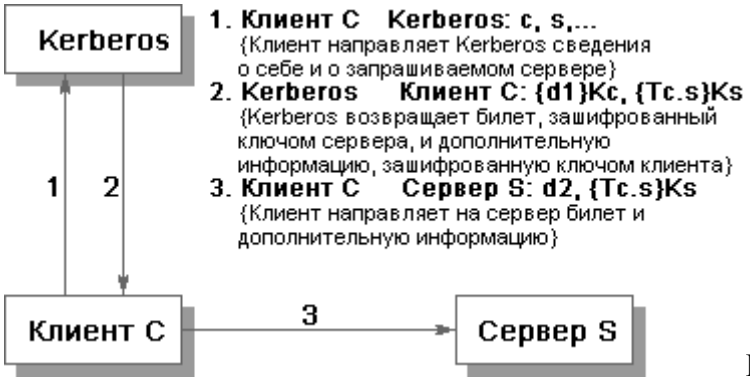


Рис. 10.1. Проверка сервером S подлинности клиента C.

Здесь c и s - сведения (например, имя), соответственно, о клиенте и сервере, d1 и d2 - дополнительная (по отношению к билету) информация, Tc.s - билет для клиента C на обслуживание у сервера S, Kc и Ks - секретные ключи клиента и сервера, {info}K - информация info, зашифрованная ключом K.

Приведенная схема - крайне упрощенная версия реальной процедуры проверки подлинности. Более подробное рассмотрение системы Kerberos можно найти, например, в статье В. Галатенко "Сервер аутентификации Kerberos (Jet Info, 1996, 12-13). Нам же важно отметить, что Kerberos не только устойчив к сетевым угрозам, но и поддерживает концепцию *единого входа в сеть*.

2.7.1.5 Идентификация/аутентификация с помощью биометрических данных

Биометрия представляет собой совокупность автоматизированных методов *идентификации* и/или *аутентификации* людей на основе их физиологических и поведенческих характеристик. К числу физиологических характеристик принадлежат особенности **отпечатков пальцев**, **сетчатки** и **роговицы** глаз, **геометрия руки и лица** и т.п. К поведенческим характеристикам относятся **динамика подписи** (ручной), стиль **работы с клавиатурой**. На стыке физиологии и поведения находятся анализ особенностей **голоса** и **распознавание речи.**

Биометрией во всем мире занимаются очень давно, однако долгое время все, что было связано с ней, отличалось сложностью и дороговизной. В последнее время спрос на биометрические продукты, в первую очередь в связи с развитием электронной коммерции, постоянно и весьма интенсивно растет. Это понятно, поскольку с точки зрения пользователя гораздо удобнее предъявить себя самого, чем что-то запоминать. Спрос рождает предложение, и на рынке появились относительно недорогие аппаратно-программные продукты, ориентированные в основном на распознавание отпечатков пальцев.

В общем виде работа с биометрическими данными организована следующим образом. Сначала создается и поддерживается база данных характеристик потенциальных пользователей. Для этого биометрические характеристики пользователя снимаются, обрабатываются, и результат обработки (называемый **биометрическим шаблоном**) заносится в базу данных (исходные данные, такие как результат сканирования пальца или роговицы, обычно не хранятся).

В дальнейшем для *идентификации* (и одновременно *аутентификации*) пользователя процесс снятия и обработки повторяется, после чего производится поиск в базе данных шаблонов. В случае успешного поиска личность пользователя и ее подлинность считаются установленными. Для аутентификации достаточно произвести сравнение с одним биометрическим шаблоном, выбранным на основе предварительно введенных данных.

Обычно биометрию применяют вместе с другими *аутентификаторами*, такими, например, как интеллектуальные карты. Иногда биометрическая аутентификация является лишь первым рубежом защиты и служит для активизации интеллектуальных карт, хранящих криптографические секреты; в таком случае биометрический шаблон хранится на той же карте.

Активность в области биометрии очень велика. Организован соответствующий консорциум (см. http://www.biometrics.org/), активно ведутся работы по стандартизации разных аспектов технологии (формата обмена данными, прикладного программного интерфейса и т.п.), публикуется масса рекламных статей, в которых биометрия преподносится как средство обеспечения сверхбезопасности, ставшее доступным широким массам.

На наш взгляд, к биометрии следует относиться весьма осторожно. Необходимо учитывать, что она подвержена тем же угрозам, что и другие методы аутентификации. Во-первых, биометрический шаблон сравнивается не с результатом первоначальной обработки характеристик пользователя, а с тем, что пришло к месту сравнения. А, как известно, за время пути... много чего может

произойти. Во-вторых, биометрические методы не более надежны, чем база данных шаблонов. В-третьих, следует учитывать разницу между применением биометрии на контролируемой территории, под бдительным оком охраны, и в "полевых" условиях, когда, например к устройству сканирования роговицы могут поднести муляж и т.п. В-четвертых, биометрические данные человека меняются, так что база шаблонов нуждается в сопровождении, что создает определенные проблемы и для пользователей, и для администраторов.

Но главная опасность состоит в том, что любая "пробоина" для биометрии оказывается фатальной. Пароли, при всей их ненадежности, в крайнем случае можно сменить. Утерянную аутентификационную карту можно аннулировать и завести новую. Палец же, глаз или голос сменить нельзя. Если биометрические данные окажутся скомпрометированы, придется как минимум производить существенную модернизацию всей системы.

2.7.2 Управление доступом

2.7.2.1 Основные понятия

С традиционной точки зрения средства управления доступом позволяют специфицировать и контролировать действия, которые субъекты (пользователи и процессы) могут выполнять над **объектами** (информацией и другими компьютерными ресурсами). В данном разделе речь идет о логическом управлении доступом, которое, в отличие от физического, реализуется программными средствами. Логическое управление доступом - это основной механизм многопользовательских систем, призванный обеспечить конфиденциальность и целостность объектов и, до некоторой степени, их доступность (путем запрещения обслуживания неавторизованных пользователей).

Рассмотрим формальную постановку задачи в традиционной трактовке. Имеется совокупность субъектов и набор объектов. Задача логического управления доступом состоит в том, чтобы для каждой пары "субъект-объект" определить множество допустимых операций (зависящее, быть может, от некоторых дополнительных условий) и контролировать выполнение установленного порядка.

Отношение "субъекты-объекты" можно представить в виде **матрицы доступа**, в строках которой перечислены субъекты, в столбцах - объекты, а в клетках, расположенных на пересечении строк и столбцов, записаны дополнительные условия (например, время и место действия) и разрешенные виды доступа. Фрагмент матрицы может выглядеть, например, так:

Таблица 10.1. Фрагмент матрицы доступа

	Файл	**Программа**	**Линия связи**	**Реляционная таблица**
Пользователь 1	orw с системной консоли	e	rw с 8:00 до 18:00	
Пользователь 2				a

"o" - обозначает разрешение на передачу **прав доступа** другим пользователям,

"r" - чтение,

"w" - запись,

"e" - выполнение,

"a" - добавление информации

Тема логического управления доступом - одна из сложнейших в области информационной безопасности. Дело в том, что само понятие объекта (а тем более

видов доступа) меняется от сервиса к сервису. Для операционной системы к объектам относятся файлы, устройства и процессы. Применительно к файлам и устройствам обычно рассматриваются права на чтение, запись, выполнение (для программных файлов), иногда на удаление и добавление. Отдельным правом может быть возможность передачи полномочий доступа другим субъектам (так называемое право владения). Процессы можно создавать и уничтожать. Современные операционные системы могут поддерживать и другие объекты.

Для систем управления реляционными базами данных объект - это база данных, таблица, представление, хранимая процедура. К таблицам применимы операции поиска, добавления, модификации и удаления данных, у других объектов иные виды доступа.

Разнообразие объектов и применимых к ним операций приводит к принципиальной децентрализации логического управления доступом. Каждый сервис должен сам решать, позволить ли конкретному субъекту ту или иную операцию. Теоретически это согласуется с современным объектно-ориентированным подходом, на практике же приводит к значительным трудностям. Главная проблема в том, что ко многим объектам можно получить доступ с помощью разных сервисов (возможно, при этом придется преодолеть некоторые технические трудности). Так, до реляционных таблиц можно добраться не только средствами СУБД, но и путем непосредственного чтения файлов или дисковых разделов, поддерживаемых операционной системой (разобравшись предварительно в структуре хранения объектов базы данных). В результате при задании матрицы доступа нужно принимать во внимание не только принцип распределения привилегий для каждого сервиса, но и существующие связи между сервисами (приходится заботиться о согласованности разных частей матрицы). Аналогичная трудность возникает при экспорте/импорте данных, когда информация о правах доступа, как правило, теряется (поскольку на новом сервисе она не имеет смысла). Следовательно, обмен данными между различными сервисами представляет особую опасность с точки зрения управления доступом, а при проектировании и реализации разнородной конфигурации необходимо позаботиться о согласованном распределении прав доступа субъектов к объектам и о минимизации числа способов экспорта/импорта данных.

При принятии решения о предоставлении доступа обычно анализируется следующая информация:

- идентификатор субъекта (идентификатор пользователя, сетевой адрес компьютера и т.п.). Подобные идентификаторы являются основой **произвольного (или дискреционного) управления доступом;**
- атрибуты субъекта (метка безопасности, группа пользователя и т.п.). Метки безопасности - основа **принудительного (мандатного) управления доступом.**

Матрицу доступа, ввиду ее разреженности (большинство клеток - пустые), неразумно хранить в виде двухмерного массива. Обычно ее хранят по столбцам, то есть для каждого объекта поддерживается список "допущенных" субъектов вместе с их правами. Элементами списков могут быть имена групп и шаблоны субъектов, что служит большим подспорьем администратору. Некоторые проблемы возникают только при удалении субъекта, когда приходится удалять его имя из всех списков доступа; впрочем, эта операция производится нечасто.

Списки доступа - исключительно гибкое средство. С их помощью легко выполнить требование о гранулярности прав с точностью до пользователя. Посредством списков несложно добавить права или явным образом запретить доступ (например, чтобы наказать нескольких членов группы пользователей). Безусловно, списки являются лучшим средством произвольного управления доступом.

Подавляющее большинство операционных систем и систем управления базами данных реализуют именно произвольное управление доступом. Основное достоинство произвольного управления - гибкость. Вообще говоря, для каждой пары "субъект-объект" можно независимо задавать права доступа (особенно легко это делать, если используются **списки управления доступом**). К сожалению, у "произвольного" подхода есть ряд недостатков. Рассредоточенность управления доступом ведет к тому, что доверенными должны быть многие пользователи, а не только системные операторы или администраторы. Из-за рассеянности или некомпетентности сотрудника, владеющего секретной информацией, эту информацию могут узнать и все остальные пользователи. Следовательно, произвольность управления должна быть дополнена жестким контролем за реализацией избранной политики безопасности.

Второй недостаток, который представляется основным, состоит в том, что права доступа существуют отдельно от данных. Ничто не мешает пользователю, имеющему доступ к секретной информации, записать ее в доступный всем файл или заменить полезную утилиту ее "троянским" аналогом. Подобная "разделенность" прав и данных существенно осложняет проведение несколькими системами согласованной политики безопасности и, главное, делает практически невозможным эффективный контроль согласованности.

Возвращаясь к вопросу представления матрицы доступа, укажем, что для этого можно использовать также функциональный способ, когда матрицу не хранят в явном виде, а каждый раз вычисляют содержимое соответствующих клеток. Например, при принудительном управлении доступом применяется сравнение меток безопасности субъекта и объекта.

Удобной надстройкой над средствами логического управления доступом является **ограничивающий интерфейс**, когда пользователя лишают самой возможности попытаться совершить несанкционированные действия, включив в число видимых ему объектов только те, к которым он имеет доступ. Подобный подход обычно реализуют в рамках системы меню (пользователю показывают лишь допустимые варианты выбора) или посредством ограничивающих оболочек, таких как restricted shell в ОС Unix.

В заключение подчеркнем важность управления доступом не только на уровне операционной системы, но и в рамках других сервисов, входящих в состав современных приложений, а также, насколько это возможно, на "стыках" между сервисами. Здесь на первый план выходит существование единой политики безопасности организации, а также квалифицированное и согласованное системное администрирование.

2.7.2.2 Ролевое управление доступом

При большом количестве пользователей традиционные подсистемы управления доступом становятся крайне сложными для администрирования. Число связей в них пропорционально произведению количества пользователей на количество объектов. Необходимы решения в объектно-ориентированном стиле, способные эту сложность понизить.

Таким решением является **ролевое управление доступом** (РУД). Суть его в том, что между пользователями и их привилегиями появляются промежуточные сущности - роли. Для каждого пользователя одновременно могут быть активными несколько ролей, каждая из которых дает ему определенные права (см. рис. 10.2).

Рис. 10.2. Пользователи, объекты и роли.

Ролевой доступ нейтрален по отношению к конкретным видам прав и способам их проверки; его можно рассматривать как объектно-ориентированный каркас, облегчающий администрирование, поскольку он позволяет сделать подсистему разграничения доступа управляемой при сколь угодно большом числе пользователей, прежде всего за счет установления между ролями связей, аналогичных наследованию в объектно-ориентированных системах. Кроме того, ролей должно быть значительно меньше, чем пользователей. В результате число администрируемых связей становится пропорциональным сумме (а не произведению) количества пользователей и объектов, что по порядку величины уменьшить уже невозможно.

Ролевой доступ развивается более 10 лет (сама идея ролей, разумеется, значительно старше) как на уровне операционных систем, так и в рамках СУБД и других информационных сервисов. В частности, существуют реализации ролевого доступа для Web-серверов.

В 2001 году Национальный институт стандартов и технологий США предложил проект стандарта ролевого управления доступом (см. http://csrc.nist.gov/rbac/), основные положения которого мы и рассмотрим.

Ролевое управление доступом оперирует следующими основными понятиями:

- **пользователь** (человек, интеллектуальный автономный агент и т.п.);
- **сеанс работы пользователя;**
- **роль** (обычно определяется в соответствии с организационной структурой);
- **объект** (сущность, доступ к которой разграничивается; например, файл ОС или таблица СУБД);
- **операция** (зависит от объекта; для файлов ОС - чтение, запись, выполнение и т.п.; для таблиц СУБД - вставка, удаление и т.п., для прикладных объектов операции могут быть более сложными);
- право доступа (разрешение выполнять определенные операции над определенными объектами).

Ролям приписываются пользователи и права доступа; можно считать, что они (роли) именуют отношения "многие ко многим" между пользователями и правами. Роли могут быть приписаны многим пользователям; один пользователь может быть приписан нескольким ролям. Во время сеанса работы пользователя активизируется подмножество ролей, которым он приписан, в результате чего он становится обладателем объединения прав, приписанных активным ролям. Одновременно пользователь может открыть несколько сеансов.

Между ролями может быть определено отношение частичного порядка, называемое наследованием. Если роль r2 является наследницей r1, то все права r1 приписываются r2, а все пользователи r2 приписываются r1. Очевидно, что **наследование ролей** соответствует наследованию классов в объектно-ориентированном программировании, только правам доступа соответствуют методы классов, а пользователям - объекты (экземпляры) классов.

Отношение наследования является иерархическим, причем права доступа и пользователи распространяются по уровням иерархии навстречу друг другу. В общем случае наследование является множественным, то есть у одной роли может быть несколько предшественниц (и, естественно, несколько наследниц, которых мы будем называть также преемницами).

Можно представить себе формирование **иерархии ролей**, начиная с минимума прав (и максимума пользователей), приписываемых роли "сотрудник", с постепенным уточнением состава пользователей и добавлением прав (роли "системный администратор", "бухгалтер" и т.п.), вплоть до роли "руководитель" (что, впрочем, не значит, что руководителю предоставляются неограниченные права; как и другим ролям, в соответствии с принципом **минимизации привилегий**, этой роли целесообразно разрешить только то, что необходимо для выполнения служебных обязанностей). Фрагмент подобной иерархии ролей показан на рис. 10.3.

Рис. 10.3. Фрагмент иерархии ролей.

Для реализации еще одного упоминавшегося ранее важного принципа информационной безопасности вводится понятие **разделения обязанностей**, причем в двух видах: статическом и динамическом.

Статическое разделение обязанностей налагает ограничения на **приписывание пользователей ролям**. В простейшем случае членство в некоторой роли запрещает приписывание пользователя определенному множеству других ролей. В общем случае данное ограничение задается как пара "множество ролей - число" (где множество состоит, по крайней мере, из двух ролей, а число должно быть больше 1), так что никакой пользователь не может быть приписан указанному (или большему) числу ролей из заданного множества. Например, может существовать пять бухгалтерских ролей, но политика безопасности допускает членство не более чем в двух таких ролях (здесь число=3).

При наличии наследования ролей ограничение приобретает несколько более сложный вид, но суть остается простой: при проверке членства в ролях нужно учитывать приписывание пользователей ролям-наследницам.

Динамическое разделение обязанностей отличается от статического только тем, что рассматриваются роли, одновременно активные (быть может, в разных сеансах) для данного пользователя (а не те, которым пользователь статически приписан). Например, один пользователь может играть роль и кассира, и контролера, но не одновременно; чтобы стать контролером, он должен сначала закрыть кассу. Тем

самым реализуется так называемое **временное ограничение доверия**, являющееся аспектом минимизации привилегий.

Рассматриваемый проект стандарта содержит спецификации трех категорий функций, необходимых для администрирования РУД:

- **Административные функции** (создание и сопровождение ролей и других атрибутов ролевого доступа): создать/удалить роль/пользователя, приписать пользователя/право роли или ликвидировать существующую ассоциацию, создать/удалить отношение наследования между существующими ролями, создать новую роль и сделать ее наследницей/предшественницей существующей роли, создать/удалить ограничения для статического/динамического разделения обязанностей.
- **Вспомогательные функции** (обслуживание сеансов работы пользователей): открыть сеанс работы пользователя с активацией подразумеваемого набора ролей; активировать новую роль, деактивировать роль; проверить правомерность доступа.
- **Информационные функции** (получение сведений о текущей конфигурации с учетом отношения наследования). Здесь проводится разделение на обязательные и необязательные функции. К числу первых принадлежат получение списка пользователей, приписанных роли, и списка ролей, которым приписан пользователь.

Все остальные функции отнесены к разряду необязательных. Это получение информации о правах, приписанных роли, о правах заданного пользователя (которыми он обладает как член множества ролей), об активных в данный момент сеанса ролях и правах, об операциях, которые роль/пользователь правомочны совершить над заданным объектом, о статическом/динамическом разделении обязанностей.

Можно надеяться, что предлагаемый стандарт поможет сформировать единую терминологию и, что более важно, позволит оценивать РУД-продукты с единых позиций, по единой шкале.

2.7.2.3 Возможный подход к управлению доступом в распределенной объектной среде

Представляется, что в настоящее время проблема управления доступом существует в трех почти не связанных между собой проявлениях:

- традиционные модели (дискреционная и мандатная);
- модель "песочница" (предложенная для Java-среды и близкой ей системы Safe-Tcl);
- модель фильтрации (используемая в межсетевых экранах).

На наш взгляд, необходимо объединить существующие подходы на основе их развития и обобщения.

Формальная постановка задачи разграничения доступа может выглядеть следующим образом.

Рассматривается множество объектов (в смысле объектно-ориентированного программирования). Часть объектов может являться **контейнерами**, группирующими объекты-компоненты, задающими для них общий контекст, выполняющими общие функции и реализующими перебор компонентов. Контейнеры либо вложены друг в друга, либо не имеют общих компонентов.

С каждым объектом ассоциирован набор интерфейсов, снабженных **дескрипторами (ДИ).** К объекту можно обратиться только посредством ДИ. Разные интерфейсы могут предоставлять разные методы и быть доступными для разных объектов.

Каждый контейнер позволяет опросить набор ДИ объектов-компонентов, удовлетворяющих некоторому условию. Возвращаемый результат в общем случае зависит от вызывающего объекта.

Объекты изолированы друг от друга. Единственным видом межобъектного взаимодействия является вызов метода.

Предполагается, что используются надежные средства аутентификации и защиты коммуникаций. В плане разграничения доступа локальные и удаленные вызовы не различаются.

Предполагается также, что разрешение или запрет на доступ не зависят от возможного параллельного выполнения методов (синхронизация представляет отдельную проблему, которая здесь не рассматривается).

Разграничивается доступ к интерфейсам объектов, а также к методам объектов (с учетом значений **фактических параметров** вызова). **Правила разграничения доступа (ПРД)** задаются в виде **предикатов** над объектами.

Рассматривается задача разграничения доступа для выделенного контейнера СС, компонентами которого должны являться вызывающий и/или вызываемый объекты. ДИ этого контейнера полагается общеизвестным. Считается также, что между внешними по отношению к выделенному контейнеру объектами возможны любые вызовы.

Выполнение ПРД контролируется **монитором обращений.**

При вызове метода мы будем разделять действия, производимые вызывающим объектом (инициация вызова) и вызываемым методом (прием и завершение вызова).

При инициации вызова может производиться преобразование ДИ фактических параметров к виду, доступному вызываемому методу (**"трансляция интерфейса"**). Трансляция может иметь место, если вызываемый объект не входит в тот же контейнер, что и вызывающий.

Параметры методов могут быть входными и/или выходными. При приеме вызова возникает информационный поток из входных параметров в вызываемый объект. В момент завершения вызова возникает информационный поток из вызываемого объекта в выходные параметры. Эти потоки могут фигурировать в правилах разграничения доступа.

Структурируем множество всех ПРД, выделив четыре группы правил:

- политика безопасности контейнера;
- ограничения на вызываемый метод;
- ограничения на вызывающий метод;
- **добровольно налагаемые ограничения.**

Правила, общие для всех объектов, входящих в контейнер С, назовем политикой безопасности данного контейнера.

Пусть метод M1 объекта O1 в точке P1 своего выполнения должен вызвать метод M объекта O. Правила, которым должен удовлетворять M, можно разделить на четыре следующие подгруппы:

- правила, описывающие требования к **формальным параметрам** вызова;
- правила, описывающие требования к семантике M;

- реализационные правила, накладывающие ограничения на возможные реализации M;
- правила, накладывающие ограничения на вызываемый объект O.

Метод M объекта O, потенциально доступный для вызова, может предъявлять к вызывающему объекту следующие группы требований:

- правила, описывающие требования к фактическим параметрам вызова;
- правила, накладывающие ограничения на вызывающий объект.

Можно выделить три разновидности предикатов, соответствующих семантике и/или особенностям реализации методов:

- утверждения о фактических параметрах вызова метода M в точке P1;
- предикат, описывающий семантику метода M;
- предикат, описывающий особенности реализации метода M.

Перечисленные ограничения можно назвать добровольными, поскольку они соответствуют реальному поведению объектов и не связаны с какими-либо внешними требованиями.

Предложенная постановка задачи разграничения доступа соответствует современному этапу развития программирования, она позволяет выразить сколь угодно сложную политику безопасности, найти баланс между богатством выразительных возможностей и эффективностью работы монитора обращений.

Ключевые термины

Буферизация вывода (spooling) – хранение для каждого задания буфера его вывода (в виде области памяти или файла), накопление в буфере выводимой заданием информации и ее вывод полностью на устройство (принтер) при завершении задания.

Диспетчер (dispatcher) – ранняя упрощенная версия операционной системы , -системная программа, управляющая прохождением пакета вводимых заданий.

Единая система ЭВМ (ЕС ЭВМ) – семейство отечественных mainframe-компьютеров 1970-х – 1980-х годов, разработанных путем копирования американских компьютеров серии IBM 360.

Задание (job) – пользовательская программа, введенная в систему с внешнего носителя или с терминала.

Мобильная (переносимая) ОС – операционная система, используемая на нескольких семействах компьютеров путем переноса ее кода (возможно, с небольшими изменениями).

Монитор – упрощенный вариант операционной системы; программа, осуществляющая поочередную обработку пользовательских заданий, с последовательной передачей управления от задания к заданию, по мере их завершения.

Мультипрограммирование (multi-programming) – одновременная обработка операционной системой нескольких пользовательских заданий.

Однозадачная операционная система – ОС, обрабатывающая, выполняющая и хранящая в оперативной памяти в каждый момент времени только одно пользовательское задание (программу).

Откачка и подкачка заданий (swapping) – загрузка задания с диска в оперативную память при его активизации и его выгрузка из памяти на диск при неактивности задания; выполняется в режиме разделения времени.

Пакетная обработка (batch mode) – обработка пакета заданий, введенных пользователями, с учетом их приоритетов и требуемых ими ресурсов.

Планирование загрузки процессора (CPU scheduling) – реализация в ОС алгоритмов выбора очередного задания их набора загруженных в память заданий и выделения кванта времени центрального процессора очередному выбранному заданию.

Разделение времени (time sharing) – поддержка операционной системой одновременной работы в системе нескольких пользователей с терминалов, управление прохождением своих заданий, выполнение их ввода, редактирования, компиляции, выполнения, отладки, визуализации результатов.

Распределения памяти для пользовательских заданий – выделение памяти операционной системой для загружаемого пользовательского задания и ее освобождение после завершения каждого задания.

Резидентная программа - программа, постоянно находящаяся в оперативной памяти по фиксированным адресам.

Система мини-ЭВМ (СМ ЭВМ) - семейство отечественных миникомпьютеров 1970-х – 1980-х годов, разработанных путем копирования американских компьютеров серии PDP 10 – PDP 11.

Тег – числовой код типа данных, хранящихся в рассматриваемом слове памяти, по которому аппаратура контролирует правильность выполнения операции над данными.

Управление процессами – параллельное (или поочередное - на однопроцессорном компьютере) выполнение пользовательских процессов; возможность явного запуска параллельных процессов, управления ими и их синхронизации.

Фрагментация памяти - дробление свободной памяти на мелкие несмежные участки, вследствие неточного совпадения размеров свободных и требуемых при запросах к ОС участков памяти.

Boot loader - загрузчик одной из нескольких ОС, установленных на некотором компьютере, управляемый специальным меню при включении компьютера.

Double bootable system - компьютер, на котором установлены две (или более) операционных системы, при включении которого пользователю выдается начальное меню для уточнения, какую именно ОС требуется запустить.

Hard real-time – система реального времени, в которой при нарушении временных ограничений может возникнуть критическая ошибка (отказ) управляемого ею объекта.

Soft real-time – система реального времени, в которой нарушение временных ограничений не приводит к отказу управляемого ею объекта.

Веб-сервер (Web server) – компьютер и программное обеспечение, предоставляющее доступ клиентам через WWW к Web-страницам, расположенным на компьютере-сервере.

Вычислительная среда – интегрированная распределенная компьютерная система для решения задач в каких-либо проблемных областях.

Драйвер – низкоуровневая системная программа для управления каким-либо внешним устройством (например, жестким диском).

Многоядерный (multi-core) компьютер – компьютерная система, основанная на тесно связанных друг с другом процессорах (**ядрах**), находящихся в одном кристалле, разделяющих ассоциативную память (кэш) второго уровня и работающих на общей памяти.

Облачные вычисления – модель вычислений, основанная на **динамически масштабируемых (scalable)** и **виртуализованных** ресурсах (данных, приложениях, ОС и др.), которые доступны и используются как **сервисы** через Интернет и реализуются с помощью высокопроизводительных **центров обработки данных (data centers).**

Параллельная компьютерная система – мультипроцессорная система, состоящая из нескольких непосредственно взаимодействующих процессоров.

Распределенная система (distributed system) – компьютерная система, в которой вычисления распределены между несколькими физическими процессорами (компьютерами), объединенными между собой в сеть**.**

Сетевой адаптер (сетевая карта) – устройство для подключения компьютера к локальной сети.

Гибридный процессор – новый, все шире распространяющийся подход к архитектуре компьютеров, при котором процессор имеет гибридную структуру – состоит из (многоядерного) центрального процессора **(CPU)** и (также многоядерного) графического процессора (GPU – Graphical Processor Unit)**.**

Карманный портативный компьютер (КПК, органайзер) - миниатюрный компьютер, помещающийся на ладони или в кармане, по своим параметрам почти сравнимый с ноутбуком, предназначенный для повседневного использования с целью записи, хранения и чтения информации, в том числе – мультимедийной, и коммуникации через Интернет.

Кластеры компьютеров – группы компьютеров, физически расположенные рядом и соединенные друг с другом высокоскоростными шинами и линиями связи.

Контроллер – устройство, осуществляющая аппаратное подключение периферийного устройства к магистрали на физическом уровне

Многоцелевые компьютеры (компьютеры общего назначения, mainframes) – традиционное историческое название для компьютеров, распространенных в 1950-х – 1970-х гг., использовавшихся для решения любых задач.

Многоядерный компьютер (multi-core computer) – наиболее распространенная в настоящее время (2010 г.) архитектура компьютеров, при которой каждый процессор имеет несколько ядер (cores), объединенных в одном кристалле и параллельно работающих на одной и той же общей памяти, что дает широкие возможности для параллельных вычислений.

Мобильное устройство (мобильный телефон, коммуникатор) – карманное устройство, предназначенное для голосовой связи, обмена короткими сообщениями, а также для чтения, записи и воспроизведения мультимедийной информации и коммуникации через Интернет.

Настольный компьютер – персональный компьютер, размещаемый на рабочем столе и используемый на работе или дома.

Носимый компьютер – сверхминиатюрный компьютер, встроенный в одежду или имплантированный в тело человека, предназначенный для обработки информации от датчиков, управления специализированными устройствами (например, кардиостимулятором), или выдачи рекомендаций по навигации и выполнению других типовых действий человеком.

Операционная система – базовое системное программное обеспечение, управляющее работой компьютера и являющееся посредником (интерфейсом) между аппаратурой, прикладным программным обеспечением и пользователем компьютера, обеспечивающая планирование и эффективное использование его ресурсов.

Память – часть компьютера, хранящая данные и программы.

Подсистема управления ресурсами – компонент операционной системы, управляющий вычислительными ресурсами компьютера.

Портативный компьютер (ноутбук, лаптоп) – миниатюрный компьютер, по своим параметрам не уступающий настольному, но по своим размерам свободно помещающийся в небольшую сумку и предназначенный для использования в поездке, дома, на даче.

Прикладное программное обеспечение – программы, предназначенные для решения различных классов задач.

Распределенная система – вычислительная система, состоящая из нескольких компьютеров, объединенных в проводную или беспроводную сеть.

Система реального времени – вычислительная система, предназначенная для управления техническим, военным или другим объектом в режиме реального времени.

Суперкомпьютер – мощный многопроцессорный компьютер, производительностью до нескольких петафлопс (10^{15} вещественных операций в секунду), предназначенный для решения задач, требующих больших вычислительных мощностей, например, моделирование, прогнозирование погоды.

Управляющая программа – компонент операционной системы, управляющая исполнением других программ и функционированием устройств ввода-вывода.

Устройства ввода-вывода – устройства компьютера, обеспечивающие ввод информации в компьютер и вывод результатов работы программ в форме, воспринимаемой пользователем или другими программами

Центральный процессор – центральная часть компьютера, выполняющая его команды (инструкции) и обеспечивающая управление аппаратными средствами.

Ядро – низкоуровневая основная компонента любой операционной системы, выполняемая аппаратурой в привилегированном режиме, загружаемая при запуске ОС и резидентно находящаяся в памяти

UNIX - первая **мобильная ОС** для миникомпьютеров, разработанная в 1970 г. Б. Керниганом и Д. Ритчи на новом языке программирования Си.

Сканер – устройство для оцифровки бумажных изображений, например, подписанных или рукописных документов.

Графическая оболочка – подсистема ОС, реализующая графический пользовательский интерфейс пользователей и системных администраторов с операционной системой.

Процесс (process) - пользовательская программа при ее исполнении в компьютерной системе.

Скрипт (script) – командный файл, содержащий часто используемые последовательности команд ОС.

Стек – системный резидентный массив в памяти, создаваемый операционной системой для поддержки выполнения процедур некоторого процесса и хранящий их

ROM BIOS (Read-Only Memory of the Basic Input-Output System) – постоянная память, входящая в состав BIOS, системного модуля компьютера, которому передается управление непосредственно после его включения; содержит часть драйверов для модулей аппаратуры.

Авторизация - предоставление операционной системой пользователю или программе какого-либо определенного набора **полномочий (permissions)**,например, возможности чтения или изменения файлов в файловой системе с общим доступом.

Виртуальная память – расширение основной памяти путем хранения ее образа на диске и организации подкачки в основную память фрагментов (страниц или сегментов) памяти процесса и ее откачки на диск по мере необходимости.

Директория (directory) – каталог ссылок на группу файлов или других директорий, каждый (каждая) из которых имеет в данной директории свое уникальное символьное **имя.**

Заголовок файла – начальная часть файла, в которой хранятся его **атрибуты.**

Защита (protection) - механизм управления доступом программ, процессов и пользователей к системным и пользовательским ресурсам.

Набор данных (data set) – то же, что и **файл** (в терминологии фирмы IBM).

Память файла – совокупность его элементов, хранящихся во внешней памяти (например, на диске).

Путь (**path**) – символьная строка для поиска файла по имени в иерархии директорий.

Файл (file) – совокупность логически взаимосвязанной информации, расположенная во внешней памяти.

Фрагментация – дробление памяти на мелкие свободные части, вследствие неточного совпадения длин имеющихся свободных и требуемых пользовательскому процессу областей памяти.

OS/2 – ОС, разработанная фирмой IBM для персональных компьютеров PS/2.

POSIX (Portable Operating Systems of unIX type) – стандарт для библиотек, системных вызовов и системных программ для операционных систем типа UNIX.

Win32 - Библиотеки (**API**) ОС Windows для 32-разрядных процессоров.

Байт-код (bytecode) – команды виртуальной Java-машины, построенные на основе

Виртуальная машина – программный интерфейс, полностью аналогичный интерфейсу обычного компьютера без базового программного обеспечения.

Виртуальная машина Java (JVM) – виртуальная машина, исполняющая Java байт-код.

Загрузка (booting) – запуск компьютера посредством загрузки ядра ОС.

Инсталляция – установка ОС на конкретный компьютер.

Интерфейс прикладного программирования (application programming interface – API) - набор библиотечных функций, реализующий некоторую функциональность, используемую программой.

Микроядро (micro-kernel) – принцип разработки ОС, который заключается в переносе максимально возможного числа модулей из системного в пользовательское "пространство", т.е. ОС разрабатывается таким образом, что большинство ее модулей выполняются в пользовательском режиме, а размер ядра минимизируется.