Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Операционные системы

сборник учебно-методических материалов

для специальности 09.02.07 "Информационные системы и программирование"

Благовещенск

2020

Печатается по решению
редакционно-издательского совета
факультета математики и информатики
Амурского государственного
университета

Составитель: Галаган Т.А., Операционные системы: сборник учебно-методических материалов для специальности 09.02.07 – Благовещенск: Амурский гос. ун-т, 2020. 65 с.

КРАТКОЕ ИЗЛОЖЕНИЕ ЛЕКЦИОННОГО МАТЕРИАЛА

ОСНОВНЫЕ ПОНЯТИЯ

Операционная система (ОС) – это комплекс специальных программных средств, предназначенных для управления загрузкой, запуском и выполнением других (пользовательских) программ, а также для планирования и управления вычислительными ресурсами ЭВМ.

В ее состав входят взаимосвязанные программы, неоднородные по характеру и многоплановые по уровню. Этот комплекс программ динамичен по своему составу: из него можно удалять и в него добавлять некоторые составные части.

Операционные системы относятся к системному программному обеспечению.

ОС предназначена для решения двух основных задач:

- предоставление пользователю (программисту) вместо реальной аппаратуры компьютера расширенной виртуальной машины, с которой удобней работать и которую легче программировать;
- повышение эффективности использования компьютера путем рационального управления его ресурсами в соответствии с некоторыми критериями.

Основные функции операционной системы:

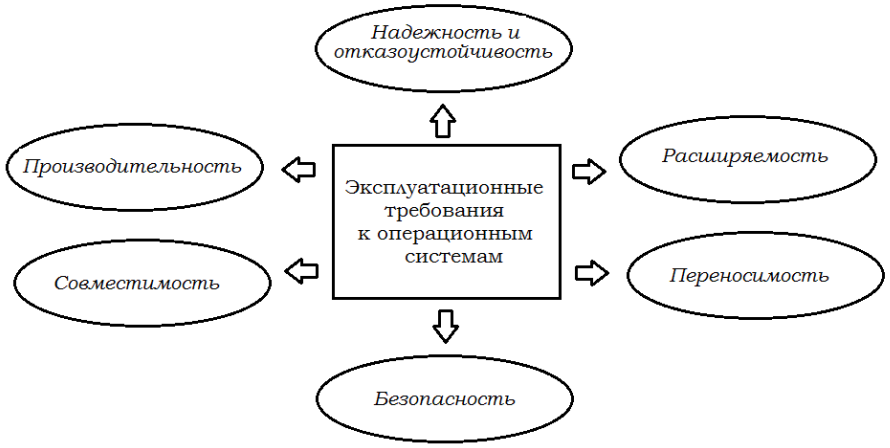
1. Прием от пользователя заданий или команд и их обработка. Как правило, это команды запуска и приостановки программ, операции с файлами.
2. Загрузка в оперативную память, подлежащих исполнению программ.
3. Распределение памяти и организация виртуальной памяти.
4. Запуск программ.
5. Идентификация всех программ и данных.
6. Прием и выполнение различных запросов от выполняющих приложений.
7. Обслуживание всех операций ввода-вывода.
8. Обеспечение работы систем управления файлами.
9. Обеспечение режима мультипрограммирования, т.е. организация параллельного выполнения двух и более программ на одном процессоре, создающая видимость их одновременного выполнения.
10. Планирование и диспетчеризация задач в соответствии с заданными стратегией.
11. Организация механизма обмена сообщениями и данными между выполняющимися программами.
12. Обеспечение сохранности данных, защита приложений друг от друга, защита самой ОС от исполняемых приложений.
13. Аутентификация и авторизация пользователя (проверка правильности имени и пароля пользователя и назначение определенных прав).
14. Предоставление услуг на случай частичного сбоя системы.

Классификация операционных систем

Существует множество классификаций операционных систем, основные из них представлены на рисунке.

Требования к современным операционным системам

Операционная система должна поддерживать мультипрограммирование, мультизадачность, виртуальную память, многопользовательский графический интерфейс и удовлетворять эксплуатационным требованиям.



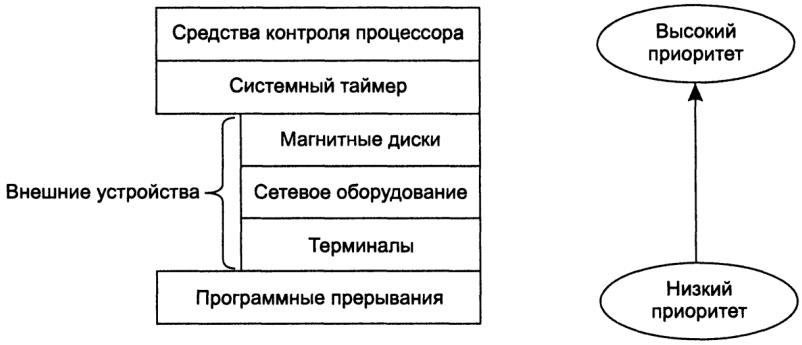
Функционирование ОС после загрузки ЭВМ осуществляется как реакция на происходящие события, наступление которых сигнализируется прерываниями – *Interrupt.* Источниками прерываний могут быть как аппаратура, так и программы. Каждое прерывание обрабатывается соответственно обработчиком прерываний (*Interrupt handler*), входящим в состав ОС.

Классы прерываний

Тип	**Возникновение**	**Примеры**
Внешние (асинхронные) прерывания	происходят вне прерываемого процесса	прерывания внешних устройств, ввода-вывода, нарушение питания
Внутренние (синхронные) Прерывания	связаны с работой процессора и синхронны с его операциями	нарушение адресации, деление на нуль, ошибки защиты памяти
Программные прерывания	выполняются по соответствующей команде; введены для переключения на системные программные модули	переключение процессора в привилегированный режим работы

Этапы обработки прерывания

Приоритеты прерываний



Процессор может обладать средствами защиты от прерываний: отключение системы прерываний, маскирование (запрет) отдельных сигналов. Программное управление этими средствами позволяет ОС регулировать обработку сигналов прерываний: откладывать их обработку или игнорировать, что позволяет реализовывать различные дисциплины диспетчеризации прерываний.

АРХИТЕКТУРА ОПЕРАЦИОННОЙ СИСТЕМЫ

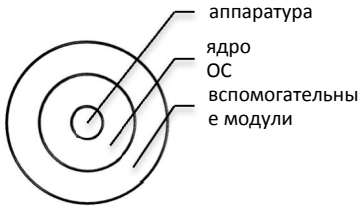
Архитектура ОС – структурная организация на основе различных программных модулей.

В макроядерной (монолитная) архитектуре все модули поделены на две группы. Ядро – модули, выполняющие основные функции операционной системы, решающие внутрисистемные задачи организации вычислительного процесса, такие как переключение контекстов, обработка прерываний и др.

Вспомогательные модули ОС выполняют полезные, но менее обязательные функции. Их обычно подразделяют на следующие группы:

- утилиты – программы, решающие отдельные задачи управления и сопровождения компьютерной системы, такие, например, как программы сжатия дисков, архивирования данных на магнитную ленту;
- системные обрабатывающие программы – текстовые или графические редакторы, компиляторы, компоновщики, отладчики;
- программы представления пользователю дополнительных услуг – специальный вариант пользовательского интерфейса, калькулятор и даже игры;
- библиотеки процедур различного назначения, упрощающие разработку приложений, например библиотека математических функций, библиотека ввода-вывода и т.д.

Вычислительная система, работающая под управлением макроядерной ОС, представлена на рисунке.



Многослойная структура ядра

Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем любого типа, в том числе и программных. Каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, образующих межслойный интерфейс. На основе же нижележащего слоя слой строит свои функции – более сложные и мощные, но которые являются примитивными для вышележащего уровня. Строгие правила касаются лишь взаимодействия слоев, а между модулями внутри слоя связи могут быть произвольными.

Многослойная организация ОС существенно упрощает разработку и модернизацию системы.

Поскольку ядро представляет собой сложный многофункциональный комплекс, то многослойный подход распространяется и на структуру ядра.

Ядро может состоять из следующих слоев:

Средства аппаратной поддержки ОС. Т.е. средства, которые непосредственно участвуют в организации вычислительных процессов, такие как: система прерываний, средства защиты областей памяти, средства переключения контекстов процессов и т. д.

Машинно-зависимые компоненты ОС. Это программные модули, в которых отражается специфика аппаратной платформы компьютера. В идеале этот слой полностью экранирует вышележащие слои от особенностей аппаратуры, что позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ.

Базовые механизмы ядра. Это наиболее примитивные операции ядра, такие как программное переключение контекстов процессов, диспетчеризация прерываний, перемеще-

ние страниц из памяти на диск и т.п. Модули данного слоя не принимают решений о распределении ресурсов, а лишь исполняют принятые «наверху» решения.

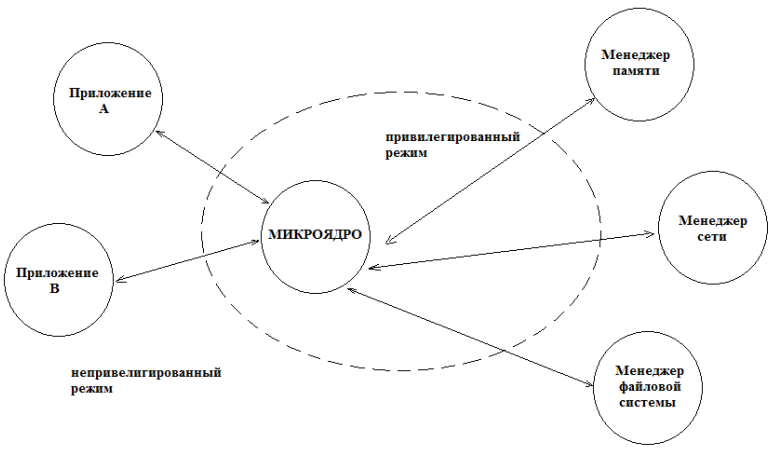
Менеджеры ресурсов. Мощные функциональные модули, реализующие стратегические задачи по управлению основными ресурсами вычислительной системы. Каждый из менеджеров (диспетчеров) ведет учет свободных и используемых ресурсов определенного типа и планирует их распределение в соответствии с запросами приложений. Например, менеджер виртуальной памяти управляет перемещением страниц из оперативной памяти на диск и обратно. Он должен отслеживать интенсивность обращений к станицам, время пребывания их в памяти, состояния процессов, использующих данные и другие, на основании которых он время от времени принимает решения о том, какие страницы необходимо выгрузить, а какие загрузить. Для исполнения принятых решений менеджер обращается к нижележащему слою.

Интерфейс системных вызовов. Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс ОС. Функции API предоставляют доступ к системным ресурсам в удобной и компактной форме, без указания деталей их физического расположения.

Приведенное разбиение ядра ОС на слои является достаточно условным. В реальной системе количество слоев и распределение между ними функций может быть иным. Выбор количества слоев ядра является ответственным и сложным делом: увеличение числа слоев ведет к некоторому замедлению работы ядра за счет дополнительных накладных расходов на межслойное взаимодействие, а уменьшение числа слоев ухудшает расширяемость и логичность системы. Обычно операционные системы, прошедшие долгий путь эволюционного развития, например UNIX, имеют неупорядоченное ядро с небольшим числом четко выделенных слоев, а у сравнительно молодых ОС, таких как, Windows NT, ядро разделено на большее число слоев и их взаимодействие формализовано в гораздо большей степени

Микроядерная архитектура

Альтернативой классическому способу построения операционной системы является микроядерная архитектура, в которой в привилегированном режиме остается работать только очень небольшая часть ОС, называемая микроядром. В состав микроядра обычно входят машинно-зависимые модули, выполняющие базовые (но не все!) функции: управление виртуальной памятью; поддержка заданий и потоков; взаимодействие между процессами; управление вводом-выводом и прерываниями; сервисы хоста (главного компьютера или компьютера, имеющего IP-адрес) Все остальные высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме



ПРОЦЕССЫ И ПОТОКИ

Внутренние единицы работы операционной системы:
процесс – задача на стадии выполнения;
поток (нить) – последовательность команд, часть процесса.

Процесс рассматривается ОС как заявка на потребление всех видов ресурсов, кроме одного – процессорного времени. Процессорное время распределяется ОС между – *потоками.*

При управлении процессами операционная система использует два основных типа информационных структур:

дескриптор процесса, содержащий информацию, необходимую ядру в течение всего жизненного цикла процесса (о состоянии процесса, о расположении образа процесса в оперативной памяти и на диске, о значении отдельных составляющих приоритета, а также о его итоговом значении – глобальном приоритете, об идентификаторах пользователя, создавшего процесс, о родственных процессах и другая информация);

контекст процесса, содержащий менее оперативную, но более объемную часть информации, необходимую для возобновления выполнения процесса с прерванного места (содержимое регистров процессора, коды ошибок выполняемых процессором системных вызовов, информация обо всех файлах, открытых процессом, о незавершенных операциях ввода-вывода).

Контекст процесса также как дескриптор доступен только программам ядра, однако он хранится не в области ядра, а непосредственно примыкает к образу процесса и перемещается вместе с ним при необходимости из оперативной памяти на диск и обратно.

Для реализации мультипрограммирования на протяжении существования процесса выполнение его потоков может быть многократно прервано и продолжено. Переход от выполнения одного потока к другому осуществляется в результате планирования и диспетчеризации.

Планирование включает определение момента времени для смены текущего потока, а также выбор нового потока для выполнения. *Диспетчеризация* заключается в реализации найденного в результате планирования решения, т.е. в переключении процесса с одного потока на другой.

Диспетчеризация включает в себя: сохранение контекста текущего потока; загрузку контекстов нового потока, выбранного в результате планирования; запуск нового потока на выполнение.

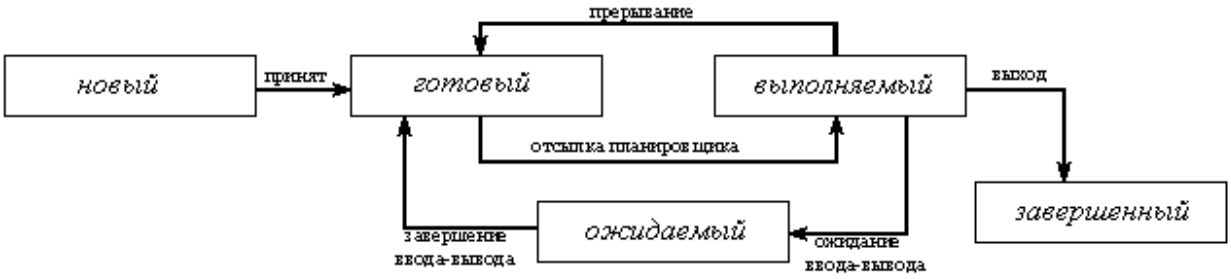
ОС выполняет планирование потоков, принимая во внимание их состояние. В мультипрограммной системе различают три основных состояния потока:

- выполнение – активное состояние потока, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессом;
- ожидание – пассивное состояние потока, находясь в котором, поток заблокирован по своим внутренним причинам (ждет осуществления некоторого события, чаще всего завершения ввода-вывода, или освобождения некоторого ресурса);
- готовность – пассивное состояние процесса, но при этом поток заблокирован в связи с внешними обстоятельствами.

Иногда еще добавляют состояние новый, т.е. только созданный поток, и завершенный.

В течение жизни поток переходит из одного состояния в другое в соответствии с алгоритмом планирования.

Типовая диаграмма переходов для состояний процесса.



Квантом называют ограниченный непрерывный период процессорного времени, которой предоставляется каждому потоку поочередно для выполнения.

Диаграмма переходов в алгоритме квантования

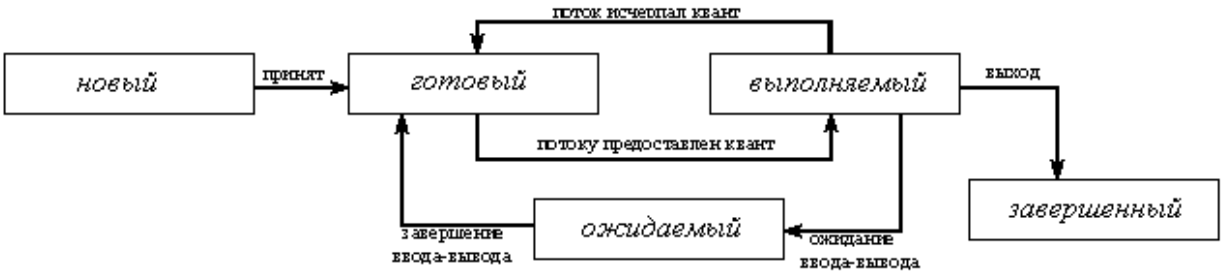
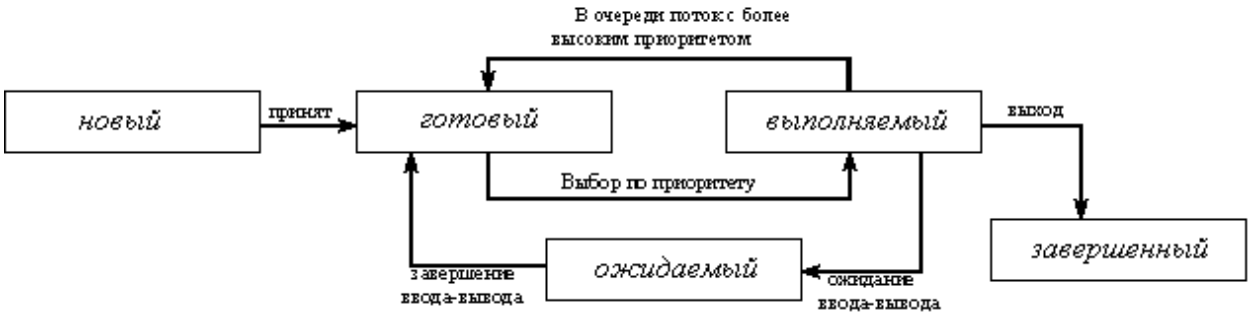


Диаграмма переходов в алгоритме с абсолютным приоритетом

Диаграмма переходов в алгоритме с относительным приоритетом.



В ОС также широко применяются алгоритмы планирования смешанной структуры. Например, в основе планирования лежит квантование, но величина кванта и/или порядок выбора потока из очереди готовых определяется приоритетами потоков. Примером служит ОС Windows NT, в которой квантование сочетается с динамическими абсолютными

приоритетами. На выполнение выбирается готовый поток с наивысшим приоритетом. Ему выделяется квант времени. Если во время выполнения в очереди готовых процессов появляется поток с более высоким приоритетом, то он вытесняет выполняемый поток. Вытесненный поток возвращается в очередь готовых, причем он становится впереди всех потоков, имеющих такой же приоритет

В мультипрограммных ОС потребность в синхронизации процессов и потоков связана с совместным использованием аппаратных и информационных ресурсов. Средства ОС, обеспечивающие эту синхронизацию, часто называются средствами межпроцессного взаимодействия. – IPC (Inter Process Communication).

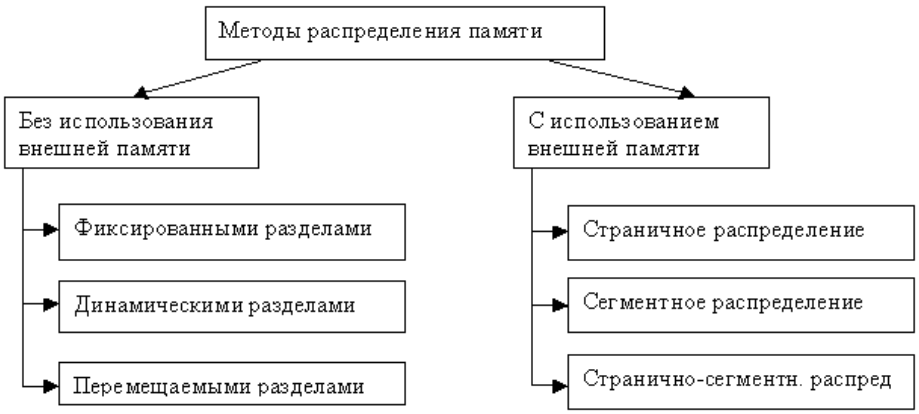
Выполнение потоков в мультипрограммной среде всегда происходят асинхронно, независимо друг от друга. Моменты прерывания потоков, время нахождения их в очередях, порядок выбора потоков для выполнения – случайные события, зависящие от стечения многих обстоятельств. В лучшем случае можно лишь оценить их вероятностные характеристики, например вероятность их завершения за данный период времени.

Любое взаимодействие потоков, связанное с разделением ресурсов, обменом данными, совместным использованием аппаратных ресурсов, обеспечивается их синхронизацией. Она заключается в согласование скоростей путем приостановки потока до наступления некоторого события и последующей его активизации.

Для синхронизации потоков прикладных программист может использовать либо собственные средства и приемы синхронизации, либо средства операционной системы.

Управление оперативной памятью

Классификация методов распределения памяти



Наименование	Принцип действия	Достоинства	Недостатки
1	2	3	4
фиксированными разделами	Границы разделов устанавливаются при старте не изменяются	Простота	не позволяет выполнить процессы, образы которых не помещаются ни в один из разделов
динамическими разделами	Каждому вновь поступающему на выполнение приложению на этапе создания процесса выделяется вся необходимая ему память.	Выполнение процессов большого объема	Если достаточный объем памяти отсутствует, то приложение не принимается, и процесс не создается. Фрагментация памяти
перемещаемыми разделами	перемещение всех занятых участков в строну старших или младших адресов так, чтобы вся свободная память образовывала единую свободную память.	Отсутствие фрагментации	процедура сжатия может потребовать значительного времени
страничная	организует перемещение данных	простота	сложность организации

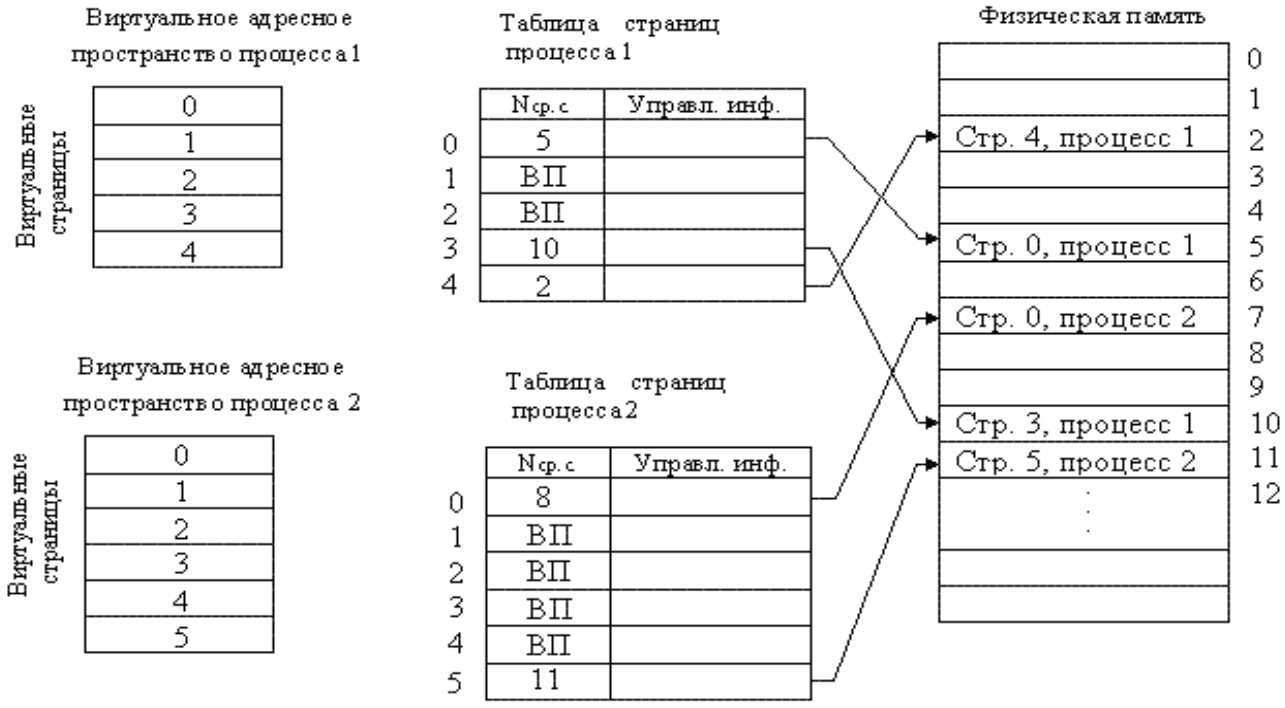
1	2	3	4
виртуальная память	между памятью и диском страницами – частями виртуального адресного пространства, фиксированного и сравнительно небольшого размера		защиты данных
сегментная виртуальная память	предусматривает перемещение данных сегментами – частями виртуального адресного пространства произвольного размера, полученными с учетом смыслового значения данных	возможность использования разделяемых данных разными процессами, защита данных	избыточный объем перемещаемой информации Фрагментация памяти
сегментно-страничная организация	использует двухуровневое деление: виртуальное адресное пространство делится на сегменты, а затем сегменты делятся на страницы	сочетает преимущества страничного и сегментного распределений	

Страничное распределение памяти

Виртуальное адресное пространство каждого процесса делится на части одинакового фиксированного для данной системы размера, называемые *виртуальными страницами*. В общем случае размер виртуального адресного пространства не кратен размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

Вся оперативная память машины также делится на части того же размера, называемые *физическими страницами.* Размер страницы выбирается равным степени двойки: 512, 1024, 4096 байт и т. д. Это позволяет упростить механизм преобразования адресов.

При создании процесса ОС загружает в оперативную память несколько его виртуальных страниц. Копия всего виртуального адресного пространства находится на диске. Смежные виртуальные страницы необязательно находятся на смежных физических сегментах. Для каждого процесса ОС создает *таблицу страниц* – информационную структуру, содержащую записи обо всех виртуальных страницах процесса.



Дескриптор страницы и содержит следующую информацию:

- номер физической страницы, в которую загружена данная виртуальная страница;
- признак присутствия, устанавливаемый в единицу, если виртуальная страница

находится в оперативной памяти;

- признак модификации страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;

- признак обращения к странице, называемой также битом доступа, который также устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Признаки присутствия, модификации и обращения в большинстве моделей современных процессоров устанавливаются аппаратно, схемами процессора при выполнении операции с памятью.

Сами таблицы страниц, также как и описываемые ими страницы хранятся в оперативной памяти. Адрес таблицы включается в контекст соответствующего процесса.

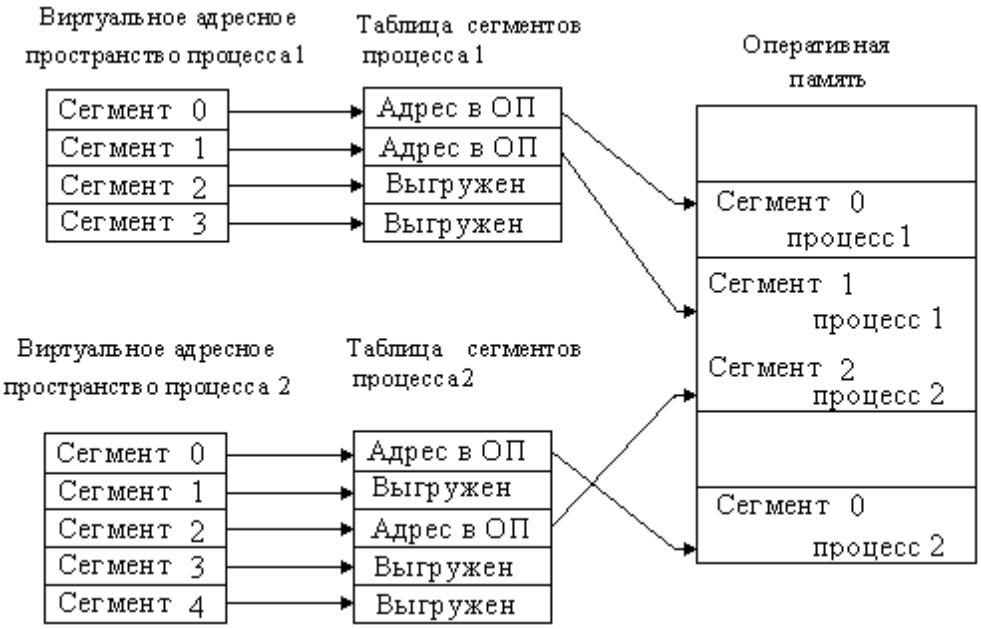
При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц, и из него извлекается описывающая страницу информация. Далее анализируется признак присутствия, и, если данная виртуальная страница находится в ОП, выполняется преобразование виртуального адреса - в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое *страничное прерывание*. Выполняющийся процесс переводится в состояние ожидание, и активизируется другой процесс из очереди процессов, находящихся в состоянии готовности. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в ОП. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то на основании принятой в данной системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти.

После выбора выгружаемой страницы обнуляется ее бит присутствия и анализируется ее бит модификации. Если выталкиваемая страница была модифицирована, то ее новая версия переписывается на диск. Если нет, запись на диск не производится, т.к. предполагается, что на диске уже есть данная копия. Физическая станица объявляется свободной. Из соображений безопасности в некоторых системах освобождаемая страница обнуляется, с тем, чтобы невозможно было использовать содержимое выгруженной страницы.

Сегментное распределение памяти

При страничной организации виртуальное адресное пространство процесса делится на равные части механически, без учета смыслового значения данных, что не позволяет обеспечивать дифференцированный доступ к разным частям программы. Деление на «осмысленные» части делает возможным совместное использование фрагментов программ разными процессами. При отображении в физическую память сегменты, содержащие коды подпрограмм из разных виртуальных пространств, могут проецироваться на одну область памяти.

Деление виртуального адресного пространства на сегменты осуществляется компилятором на основе указаний программиста или, по умолчанию, в соответствии с принятыми в системе соглашениями. Размер сегмента определяется с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т.п. Максимальный размер сегмента определяется разрядностью виртуального адреса.



При загрузке процесса в ОП помещается только часть его сегментов, полная копия виртуального адресного пространства находится в дисковой памяти. Для каждого загружаемого сегмента ОС подыскивает непрерывный участок свободной памяти подходящего размера. Смежные в виртуальной памяти сегменты одного процесса могут занимать в ОП несмежные участки. Если во время выполнения процесса происходит обращение по виртуальному адресу, содержащемуся в сегменте, отсутствующему в памяти, происходит прерывание. ОС приостанавливает процесс, запускает на выполнение следующий процесс из очереди и параллельно организует загрузку нужного сегмента с диска. При отсутствии места ОС выбирает сегмент на выгрузку, используя критерии аналогичные критериям выбора страниц при страничном распределении.

Во время загрузки образа процесса в ОП система создает *таблицу сегментов* процесса, в которой указывается:

- базовый физический адрес сегмента в ОП,
- размер сегмента,
- правила доступа к сегменту,
- признаки модификации, присутствия и обращения к данному сегменту,
- другая информация.

Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок ОП.

Виртуальный адрес при сегментной организации памяти представлен парой, содержащей номер сегмента и смещение в сегменте. Физический адрес получается сложением базового адреса сегмента, который определяется по номеру сегмента из таблицы сегментов и смещения в сегменте, что замедляет процедуру преобразования адресов.

Сегментно-страничное распределение

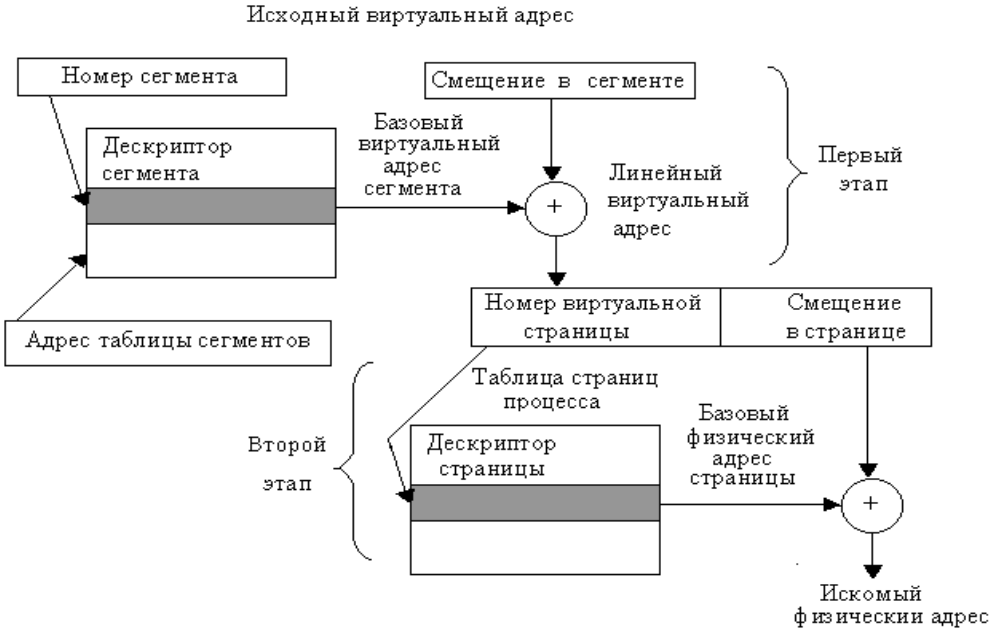
Данный метод представляет собой комбинацию страничного и сегментного механизмов управления памятью и направлен на реализацию достоинств обоих подходов.

Виртуальное адресное пространство процесса разделено на сегменты так же как при сегментной организации памяти, что позволяет определять разные права доступа к разным частям кодов и данных программы.

Перемещение же данных между памятью и диском осуществляется не сегментами, а страницами. Для этого каждый виртуальный сегмент и физическая память делятся на

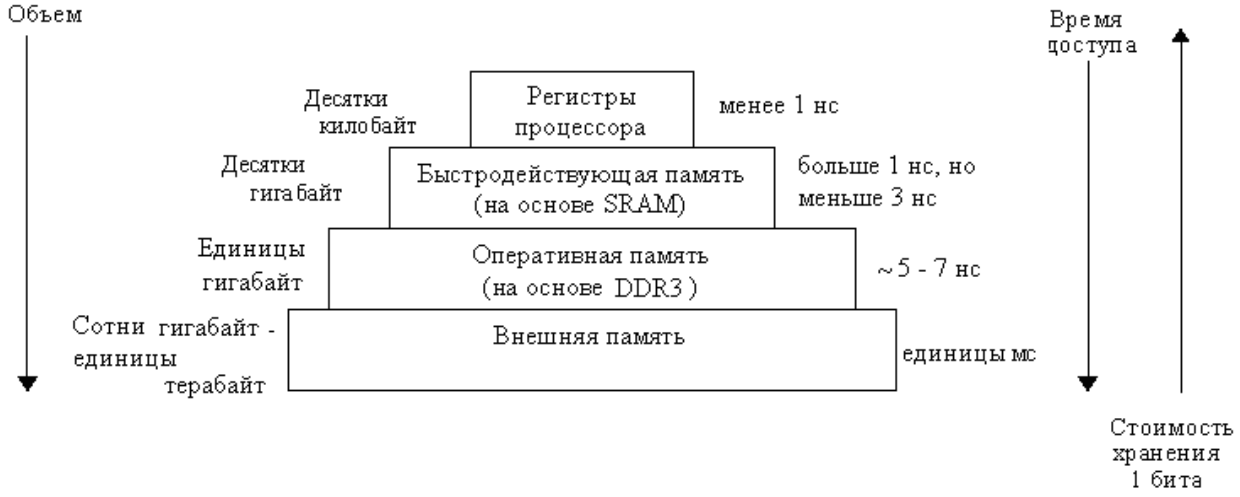
страницы равного размера, что позволяет более эффективно использовать память, сократив до минимума фрагментацию

Преобразование виртуального адреса в физический при сегментно-страничной организации памяти



Иерархия запоминающих устройств

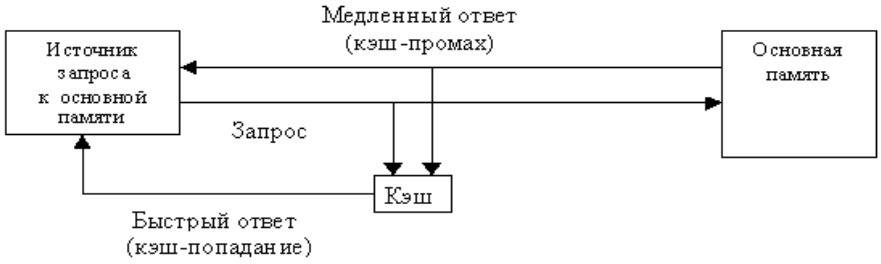
Память вычислительной машины представляет собой иерархию запоминающих устройств (ЗУ), отличающихся средним временем доступа к данным, объемом и стоимостью хранения одного бита.



Кэш-память – способ совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который за счет динамического копирования в «быстрое» запоминающее устройство наиболее часто используемой информации из «медленного» позволяет уменьшить среднее время доступа к данным и экономить более дорогую быстродействующую память. Неотъемлемым свойством кэш-памяти является ее прозрачность для программ и пользователей. Система не требует никакой внешней информации об интенсивности использования данных; ни пользователи, ни программы не принимают никакого участия в перемещении данных из ЗУ одного типа в ЗУ другого типа, все делается автоматически системными средствами.

Кэш-памятью часто называют не только способ организации работы двух типов ЗУ, но и одно из устройств – «быстрое» ЗУ. Оно стоит дороже и, как правило, имеет сравнительно небольшой объем. «Медленное» ЗУ будем далее называть основной памятью

Принцип действия кэш-памяти



Структура кэш-памяти

Адрес данных в основной памяти	Данные	Управляющая информация

При каждом обращении к основной памяти по физическому адресу просматривается содержимое кэш-памяти:

- если данные обнаруживаются в кэш-памяти, т.е. произошло кэш-попадание, они считываются, и результат передается источнику запроса;
- если нужные данные отсутствуют в кэш-памяти (кэш-промах), они считываются из основной памяти, передаются источнику запроса и одновременно с этим копируются в кэш-память.
-

ФАЙЛОВАЯ СИСТЕМА

Одной из основных задач ОС является предоставление пользователю удобств пользователю при работе с данными, хранящимися на дисках. Для этого ОС подменяет физическую структуру хранящихся данных некоторой удобной для пользователя логической моделью. Логическая модель файловой системы материализуется в виде дерева каталогов, выводимого на экран такими утилитами, как Norton Commander или Windows Explorer, в символьных составных именах файлов, в командах работы с файлами. Базовым элементом этой модели является файл, который может характеризоваться как логической, так и физической структурой.

Файловая система включает:

- совокупность всех файлов на диске;
- наборы структур данных, используемых для управления файлами, такие как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;
- комплекс системных программных средств, реализующих различные операции над файлами, такие как создание, удаление, чтение и т.д.

ТИПЫ ФАЙЛОВ

В разных источниках по информатике и вычислительной технике определения термина «файл» так же, как и термина «операционная система» могут варьироваться. Наиболее распространенной является формулировка, что «файл – именованный массив информации, хранящийся на носителе».

Файловой системой могут использоваться и другие атрибуты файлов, такие как:

- текущий размер файла;
- максимальный размер файла;
- длина записи;
- времена создания, последнего доступа и последнего изменения;
- владелец файла;
- создатель файла;
- информация о доступе к файлу;
- пароль для доступа к файлу;
- признак только для «чтения»;
- признак «скрытый файл»;
- признак «архивный файл»;
- признак «временный»;
- признак блокировки.

Для логической организации файлов используют каталоги. Каталог содержит файлы, объединенные по какому-либо признаку. *Каталог* – это файл, содержащий информацию о входящих в него файлах. Все каталоги, находящиеся на носителе, образуют иерархическую структуру.

В зависимости от файловой системы структура каталогов может быть древовидной, когда один файл может входить только в один каталог, и сетевой, когда один файл может входить в различные каталоги. Каталоги устанавливают соответствие между именами файлов и их характеристиками, используемыми ФС для управления файлами, например тип файла, его расположение на диске, права доступа и т.д.

Специальные файлы – фиктивные файлы, ассоциированные с устройствами ввода-вывода, которые используются для унификации механизма доступа к файлам и внешним устройствам. Они позволяют выполнять операции ввода-вывода посредством обычных команд записи в файл или вывода из файла.

Специальное системное программное обеспечение, реализующее работу с файлами по принятым спецификациям файловой системы, часто называют *системой управления файлами.* Она отвечает за создание, уничтожение, организацию, чтение, запись, модификацию и перемещение файловой информации, а также за управление доступом к файлам и за управление ресурсами, которые используются файлами. Назначение системы управления файлами — предоставление более удобного доступа к данным, организованным как файлы, то есть вместо низкоуровневого доступа к данным с указанием конкретных физических адресов используется логический доступ с указанием имени файла и записи в нем.

Благодаря системам управления файлами пользователям предоставляются следующие возможности:

- создание, удаление, переименование (и другие операции) файлов из своих программ или посредством специальных управляющих программ, реализующих функции интерфейса пользователя;
- работа с недисковыми периферийными устройствами как с файлами;
- обмен данными между файлами, устройствами, между файлом и устройством (и наоборот);

- работа с файлами путем обращений к программным модулям системы управления файлами (часть API ориентирована именно на работу с файлами);

- защита файлов от несанкционированного доступа.

Как правило, все современные операционные системы имеют соответствующие системы управления файлами. А некоторые - возможность работы с несколькими файловыми системами (либо с одной из нескольких, либо сразу с несколькими одновременно). В этом случае говорят о *монтируемых файловых системах* (монтируемую систему управления файлами можно установить как дополнительную), и в этом смысле они самостоятельны.

Очевидно, что система управления файлами, будучи компонентом операционной системы, не является независимой от нее, поскольку активно использует соответствующие вызовы API. С другой стороны, системы управления файлами сами дополняют API новыми вызовами. Следует заметить, что любая система управления файлами разрабатывается для работы в конкретной операционной системе.

Таким образом, термин *файловая система* определяет, прежде всего, принципы доступа к данным, организованным в файлы. Тот же термин используют и по отношению к конкретным файлам, расположенным на том или ином носителе данных. А термин *система управления файлами* следует употреблять по отношению к конкретной реализации файловой системы, то есть это — комплекс программных модулей, обеспечивающих работу с файлами в конкретной операционной системе.

ФАЙЛОВАЯ СИСТЕМА NTFS

NTFS (New Technology File System) — файловая система новой технологии. Действительно, файловая система NTFS по сравнению FAT16 (и даже FAT32)содержит ряд значительных усовершенствований и изменений. С точки зрения пользователей файлы попрежнему хранятся в каталогах (называемых *папками* (folders)). Однако в ней появилось много новых особенностей и возможностей.

Основные возможности файловой системы NTFS

При проектировании NTFS особое внимание было уделено надежности, механизмам ограничения доступа к файлам и каталогам, расширенной функциональности, поддержке дисков большого объема и пр. Начала разрабатываться эта система в рамках проекта OS/2 v.3, поэтому она переняла многие интересные особенности файловой системы HPFS.

Надежность

Высокопроизводительные компьютеры и системы совместного использования должны обладать повышенной надежностью, которая является ключевым элементом структуры и функционирования NTFS. Система NTFS обладает определенными средствами самовосстановления. Она поддерживает различные механизмы проверки целостности системы, включая ведение журналов транзакций, позволяющих воспроизвести файловые операции записи по специальному системному журналу. При протоколировании файловых операций система управления файлами фиксирует в специальном служебном файле (журнале) происходящие изменения. В начале операции, связанной с изменением файловой структуры, делается соответствующая пометка. Если во время файловых операций происходит сбой, то из-за упомянутой отметки операция остается помеченной как незавершенная. При выполнении процедуры проверки целостности файловой системы после перезагрузки машины эти незавершенные операции отменяются, файлы возвращаются в исходное состояние. Если же операция изменения данных в файлах завершается нормальным образом, то в файле журнала эта операция отмечается как завершенная.

Поскольку NTFS разрабатывалась как файловая система для серверов, для которых очень важно обеспечить бесперебойную работу без перезагрузок, в ней, как и HPFS, для повышения надежности был введен механизм аварийной замены дефектных секторов ре-

зервными. Другими словами, если обнаруживается сбой при чтении данных, то система постарается прочесть эти данные, переписать их в специально зарезервированное для этой цели пространство диска, а дефектные сектора пометить как плохие и более к ним не обращаться.

Ограничения доступа к файлам и каталогам

Файловая система NTFS поддерживает объектную модель безопасности операционной системы Windows NT и рассматривает все тома, каталоги и файлы как самостоятельные объекты. Система NTFS обеспечивает безопасность на уровне файлов и каталогов. Это означает, что разрешения доступа к томам, каталогам и файла могут зависеть от учетной записи пользователя и тех групп, к которым он принадлежит. Каждый раз, при обращении к объекту файловой системы разрешения пользователя на доступ проверяются по *списку управления доступом* (ACL) для данного объекта. Если пользователь обладает необходимым уровнем разрешений, его запрос удовлетворяется; в противном случае запрос отклоняется. Эта модель безопасности применяется как при локальной регистрации пользователей на компьютерах с Windows NT, так и при удаленных сетевых запросах.

Расширенная функциональность

Система NTFS проектировалась с учетом возможного расширения. В ней воплощены многие дополнительные возможности — повышенная отказоустойчивость, эмуляция других файловых систем, мощная модель безопасности, параллельная обработка потоков данных и создание файловых атрибутов, определяемых пользователем. Эта система также позволяет сжимать как отдельные файлы, так и целые каталоги. В последней, пятой, версии NTFS введена возможность шифрования хранимых файлов. Следует заметить, что у шифрующей файловой системы пока больше недостатков, чем достоинств, поэтому на практике ее применять не рекомендуется.

Наконец, в системах Windows 2000/XP в случае использования файловой системы NTFS можно включить квотирование, при котором пользователи могут хранить свои файлы только в пределах отведенной им квоты на дисковое пространство.

Поддержка дисков большого объема

Система NTFS создавалась с расчетом на работу с большими дисками. Она уже достаточно хорошо проявляет себя при работе с томами объемом 300-400 Мбайт и выше. Чем больше объем диска и чем больше на нем файлов, тем больший выигрыш мы получаем, используя NTFS вместо FAT16 или FAT32. Максимально возможные размеры тома (и размеры файла) составляют 16 Эбайт (один экзабайт равен 2^{64} байт, или приблизительно 16 000 млрд гигабайт), в то время как при работе под Windows NT/2000/XP диск с FAT16 не может иметь размер более 4 Гбайт, а с FAT32 — 32 Гбайт. Количество файлов в корневом и некорневом каталогах при использовании NTFS не ограничено. Поскольку в основу структуры каталогов NTFS заложена эффективная структура данных, называемая «двоичным деревом», время поиска файлов в NTFS не связано линейной зависимостью с их количеством (в отличие от систем на базе FAT). Наконец, помимо немыслимых размеров томов и файлов, система NTFS также обладает встроенными средствами сжатия, что позволяет экономить дисковое пространство и размещать в нем больше файлов. Напомним, что сжатие можно применять как к отдельным файлам, так и целым каталогам и даже томам (и впоследствии отменять или назначать их по своему усмотрению).

Структура тома с файловой системой NTFS

Одним из основных понятий, используемых при работе с NTFS, является понятие *тома* (volume). Том означает логическое дисковое пространство которое может быть воспринято как логический диск, то есть том может иметь букву (буквенный идентификатор) диска. Частным случаем тома является логический диск. Возможно также создание отказоустойчивого тома, занимающего несколько разделов, то есть поддерживается использование RAID-технологии. RAID — это сокращение от Redundant Array of Inexpensive Disks, что дословно переводится как «избыточный массив недорогих дисков». RAID-технология

позволяет получать дисковые подсистемы из нескольких обычных дисков, которые обладают либо существенно более высоким быстродействием, либо более высокой надежностью, либо тем и другим одновременно. К сожалению, в файловой системе NTFS5, применяемой в Windows 2000/XP, для использования RAID-технологии в случае, когда эти системы устанавливаются не поверх старой системы Windows NT 4.0, а заново, требуются так называемые *динамические диски.* Это фирменный закрытый стандарт распределения дискового пространства, не имеющий ничего общего с тем промышленным стандартом, который использует главную загрузочную запись. Основным недостатком нового стандарта от Microsoft является абсолютная несовместимость с другими операционными системами. Другими словами, если жесткий диск с помощью оснастки Управление дисками был преобразован в динамический, то на этот компьютер более не удастся установить никакую операционную систему, а установленные ранее системы, отличные от Windows 2000/XP/2003, не смогут даже запуститься. Кроме этого, обратное преобразование динамического диска до так называемой «базовой модели» (так компания Microsoft назвала промышленный стандарт описания логической структуры диска) невозможно без полной потери данных. Единственным достоинством динамической модели дисков является возможность преобразования томов или изменения размера логического диска прямо «на лету», то есть без последующей обязательной перезагрузки операционной системы. Технологию изменения размеров дисковых томов «на лету» разработала фирма Veritas Software. Компания Microsoft лицензировала эту технологию, ввела дополнительные ограничения на ее использование и назвала динамическими дисками.

Как и многие другие файловые системы, NTFS делит все полезное дисковое пространство тома на кластеры — блоки данных, адресуемые как единицы данных. Файловая система NTFS поддерживает размеры кластеров от 512 байт до 64 Кбайт; неким стандартом же считается кластер размером 2 или 4 Кбайт. К сожалению, при увеличении размера кластера свыше 4 Кбайт становится невозможным сжимать файлы и каталоги.

Все дисковое пространство в NTFS делится на две неравные части. Первые 12 % диска отводятся под так называемую зону MFT (Master File Table — главная таблица файлов). Эта зона предназначена для таблицы MFT (с учетом ее будущего роста), представляющей собой специальный файл со служебной информацией, позволяющей определять местонахождение всех остальных файлов. Запись каких-либо данных в зону MFT невозможна — она всегда остается пустой, чтобы при росте MFT по возможности не было фрагментации. Остальные 88 % тома представляют собой обычное пространство для хранения файлов.

Структуру данных, называемую главной таблицей файлов, можно рассматривать как файл. В этом файле MFT хранится информация обо всех остальных файлах диска, в том числе и о самом файле MFT. Таблица MFT поделена на записи фиксированного размера в 1 Кбайт, и каждая запись соответствует какому-либо файлу (в общем смысле этого слова). Первые 16 файлов носят служебный характер и недоступны через интерфейс операционной системы — они называются *метафайлами,* причем самый первый метафайл — это сам файл MFT. Часть диска с метафайлами — единственная часть диска, имеющая строго фиксированное положение. Копия этих же 16 записей таблицы MFT (для надежности, поскольку они очень важны) хранится в середине тома. Оставшаяся часть файла MFT может располагаться, как и любой другой файл, в произвольных местах диска — восстановить его положение можно с помощью самого файла MFT. Для этого достаточно взять первую запись таблицы MFT.

MFT	Зона MFT	Зона размещения файлов и каталогов	Копия первых 16 записей MFT	Зона размещения файлов и каталогов

Структура тома NFTS

Первые 16 файлов NTFS (метафайлы) являются служебными; каждый из них отвечает за какой-либо аспект работы системы. Метафайлы находятся в корневом каталоге тома NTFS.

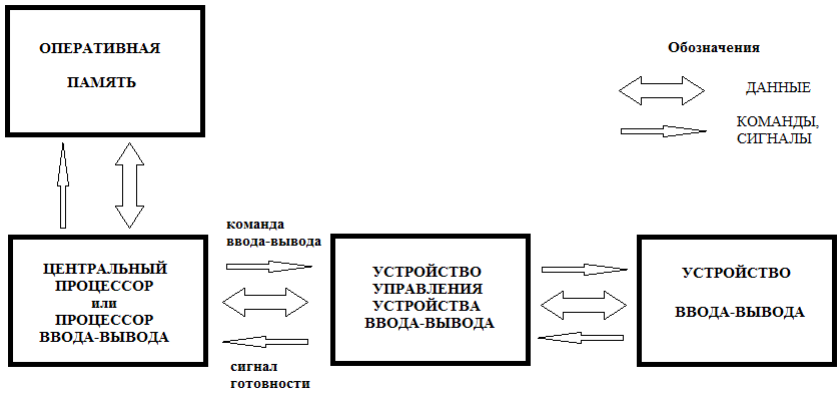
Подсистема ввода-вывода

Существуют разделяемые и неразделяемые устройства ввода-вывода. Примерами разделяемого устройства могут служить накопитель на магнитных дисках, устройство чтения компакт-дисков. Это устройства с прямым доступом. Примеры неразделяемых устройств – принтер, накопитель на магнитных лентах. Это устройства с последовательным доступом.

Режимы управления вводом-выводом

Существует два основных режима ввода-вывода: *режим обмена с опросом готовности* устройства ввода-вывода и *режим обмена с прерываниями.*

В первом случае центральный процессор посылает команду устройству управления. Устройство управления исполняет команду, транслируя сигналы, понятные ему и центральному устройству, в сигналы, понятные устройству ввода-вывода. После выполнения команды устройство ввода-вывода (или его устройство управления) выдает *сигнал готовности,* который сообщает процессору о возможности выполнения новой команды для продолжения обмена данными. Однако, поскольку быстродействие устройства ввода-вывода намного меньше быстродействия центрального, сигнал готовности приходится очень долго ожидать, постоянно опрашивая соответствующую линию интерфейса, что приводит к нерациональному использованию процессорного времени. Гораздо выгоднее сигнал готовности трактовать как запрос на прерывание от устройства ввода-вывода, как и делается во втором случае.



Таблицы ввода-вывода

Наименование	Назначение	Содержимое
таблица оборудования	хранение информации обо всех устройствах системы	тип устройства, символьное имя, характеристики; способ подключения устройства (интерфейс, разъем, порты, линия запроса прерывания); номер и адрес канала; информацию о драйвере, адреса секции запуска и секции продолжения драйвера; имя или адрес буфера; установка тайм-аута и ячейки для счетчика тайм-аута; состояние устройства; поле указателя для связи задач, ожидающих

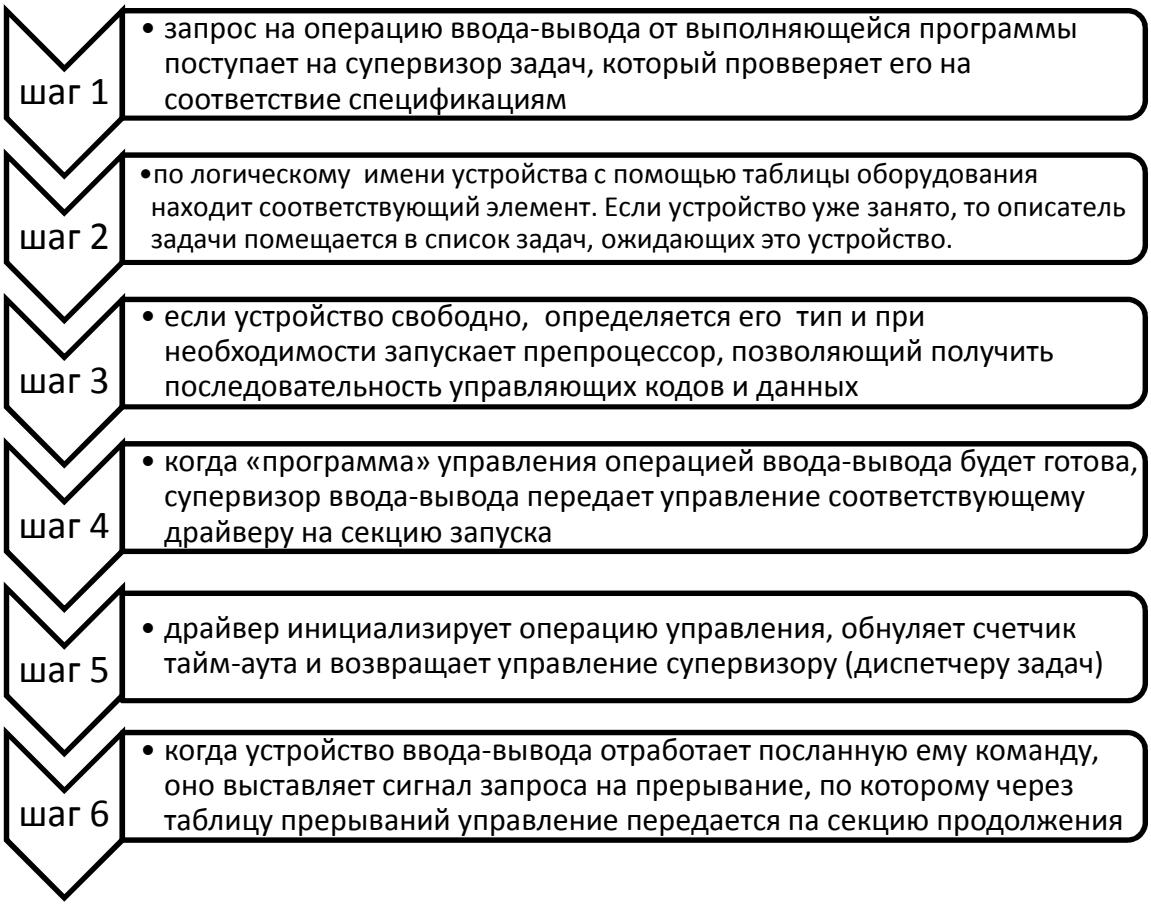
		устройство.
таблица виртуальных логических устройств	установление связи между виртуальными (логическими) устройствами и реальными устройствами	
таблица прерываний	для организации обратной связи между центральной частью и устройствами ввода-вывода	для каждого сигнала запроса на прерывание указывает на элемент таблицы оборудования, который сопоставлен нужному устройству

Управление вводом-выводом

Любые операции по управлению вводом-выводом объявляются привилегированными и могут выполняться только кодом самой операционной системы.

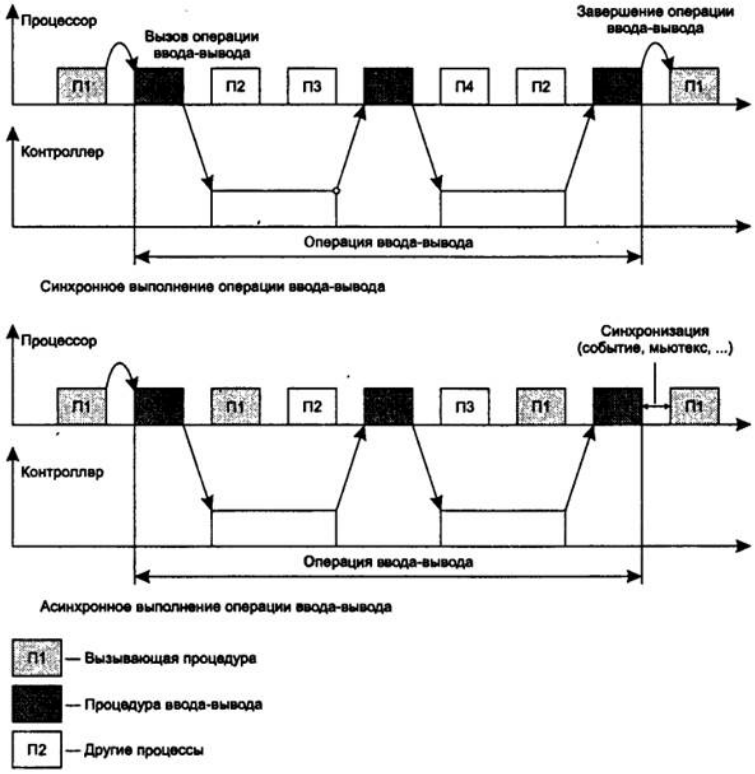
Управление вводом-выводом осуществляется компонентом операционной системы, который часто называют супервизором ввода-вывода.

Шаги работы супервизора ввода-вывода



Синхронный и асинхронный ввод-вывод

Задача, выдавшая запрос на операцию ввода-вывода, переводится супервизором в состояние ожидания завершения заказанной операции. Когда супервизор получает от секции завершения сообщение о том, что операция завершилась, он переводит задачу в состояние готовности к выполнению, и она продолжает выполняться. Эта ситуация соответствует *синхронному вводу-выводу.* Синхронный ввод-вывод является стандартным для большинства операционных систем. Для увеличения скорости выполнения приложений, можно использовать *асинхронный ввод-вывод.*


Процессор
Вызов операции ввода-вывода
Завершение операции ввода-вывода
П1
П2
П3
П4
П2
П1
Контроллер
Операция ввода-вывода
Синхронное выполнение операции ввода-вывода
б
Процессор
Синхронизация (событие, мьютекс, ...)
П1
П1
П2
П3
П1
П1
Контроллер
Операция ввода-вывода
Асинхронное выполнение операции ввода-вывода
П1 — Вызывающая процедура
— Процедура ввода-вывода
П2 — Другие процессы

Общие рекомендации по выполнению работ

Перед началом выполнения задания тщательно изучите теоретические сведения, приведенные в тексте работы. Затем переходите к непосредственному выполнению задания. Запоминайте полученные промежуточные и конечные результаты скин-шотами.

Лабораторная работа 1. Интерфейсы ОС. Операционная система Windows.

1. Интерфейс командной строки Windows

Эффективная профессиональная работа опытного пользователя с операционной системой компьютера немыслима без овладения интерфейсом, обеспечиваемым командной строкой. Этот вид интерфейса является одним из основных применительно к операционным системам Unix и Linux. Преимуществом данного интерфейса служит возможность более “утонченного” управления ресурсами системы, чем с помощью графического интерфейса.

Включение режима командной строки (используется для выполнения вводимых с клавиатуры команд) может быть выполнено двумя путями:

1) Пуск→Выполнить, затем набрать команду “**cmd**” (команда Windows – сокращение от слова commander);

2) Используя рубрики главного меню Пуск → Все программы → Стандартные → Командная строка.

В обоих случаях на экране дисплея должно появиться окно с мигающим курсором – приглашение к вводу команд.

```
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

D:\>
```

Список основных команд.

```
Microsoft Windows [Version 6.1.7601]
(c) Корпорация Майкрософт (Microsoft Corp.), 2009. Все права защищены.

D:\>help
Для получения сведений об определенной команде наберите HELP <имя команды>
ASSOC          Вывод либо изменение сопоставлений по расширениям имен файлов.
ATTRIB         Отображение и изменение атрибутов файлов.
BREAK        Включение и выключение режима обработки комбинации клавиш CTRL+C.
BCDEDIT        Задает свойства в базе данных загрузки для управления начальной
               загрузкой.
CACLS          Отображение и редактирование списков управления доступом (ACL)
               к файлам.
CALL           Вызов одного пакетного файла из другого.
CD             Вывод имени либо смена текущей папки.
CHCP           Вывод либо установка активной кодовой страницы.
CHDIR          Вывод имени либо смена текущей папки.
CHKDSK         Проверка диска и вывод статистики.
CHKNTFS        Отображение или изменение выполнения проверки диска во время
               загрузки.
```

Подробную информацию по каждой команде можно получить, если после команды **help** в качестве параметра набрать имя интересующей команды или имя команды, а затем ключ **/?**.

В сложных сценариях работы, когда используются длинные последовательности вводимых команд, всегда имеется возможность повторить или отредактировать одну (или часть команд) из выполненного перечня. Для этого следует выбрать нужную команду из журнала (истории). Клавиша управления курсором вверх выводит предыдущую команду, вниз – последующую, клавиша вправо позволяет перейти к посимвольному редактированию очередной команды.

Некоторые часто используемые команды управления компьютером

1) **cls –** команда очистки поля экрана. Строка приглашения становится первой строкой, остальное поле готово принимать информацию, выводимую при выполнении следующих команд.

2) **ver –** команда осуществляет проверку и информирование пользователя об установленной, работающей версии программного обеспечения.

3) **date** – команда используется для определения и смены текущей даты. Пользователю сообщается установленная текущая дата и предлагается шаблон *дд-мм-гг* для ввода новой даты в европейском стандарте дат.

4) **time** – команда используется для определения и смены значения реального времени. В каждом компьютере аппаратно-программными средствами таймера производится отсчет времени. Коррекция отсчетов времени осуществляется с большой точностью (с частотой 18,2 имп/с). Пользователю сообщается текущее время с точностью до сотых долей секунды.

5) **prompt** – команда обеспечивает настройку вида приглашения по желанию пользователя. Команда имеем множество параметров, список которых можно получить через помощь (**help**). Стандартным видом приглашения является **prompt pg**, что обозначает вывод в приглашении имени текущего (рабочего) каталога (папки) и знака >.

***Задание 1*.** Включить в строку приглашения:

а) сообщение о текущих значениях даты и времени;

б) текст-приветствие пользователю типа “User! Вас ждут великие дела!”;

в) изменить заголовок окна на текст «Номер гр. Фамилия исполнителя».

2. Команды управления файловой системой (ФС)

Одной из основных подсистем любой ОС является файловая система. Для управления файлами используется целая группа команд, позволяющая их создавать, уничтожать, переименовывать, копировать, редактировать и т.п.

dir – команда вывода на экран дисплея содержимого каталогов (папок). Содержимое каталога (папки) включает данные об имеющихся файлах и дочерних каталогах, их именах, размерах файлов, времени и датах их создания, сводные данные о совокупных размерах файлов и свободной области памяти на диске. Две строки содержимого вместо имен помечены точкой и двумя точками, что обозначает соответственно вход и выход в данном каталоге (папке).

Формат команды **dir** *диск:путь\имя_файла.расширение ключи*.

mkdir (или **md**) – команда предназначается для создания новых каталогов (папок). Формат команды

mkdir (или **md**) *диск:путь\имя_каталога*

Например, команда **md k:\practice** создает в корневом каталоге диска K: дочерний каталог \practice.

Если в команде не указывается диск и путь, то по умолчанию подразумевается, что создается дочерний каталог текущего каталога. При создании новых пользовательских каталогов необходимо придерживаться правила: дерево создаваемых каталогов должно быть “широким, но не глубоким”. В противном случае трудоемкость записи пути к требуемым каталогам и файлам резко возрастает.

chdir (или **cd**) – команда смены каталога.

Формат команды **chdir** *диск:путь*

Команда имеет следующие особенности применения:

- при выполнении команды новым текущим каталогом становится последний из указанных в *пути* каталогов;

- если в команде параметр *диск:путь* не указан, то командный процессор выводит сообщение пути к текущему каталогу. Применение данного варианта оправдано, когда стандартный вид приглашения (**prompt pg**) не установлен;

- если поиск нового каталога требуется начать с корневого каталога, то в качестве первого символа пути используется символ /. В противном случае поиск начинается с текущего каталога;

- для перехода из дочернего каталога в родительский каталог можно использовать сокращенную форму команды **cd..**

- для перехода из каталога нижнего уровня в корневой каталог диска лучше использовать сокращенную форму команды **cd**

***Задание 2*.**

1. Запомнить (записать) полный путь к Вашему текущему каталогу.
2. Создать дочерний каталог для своего текущего каталога.
3. Используя команду **dir**, убедиться в создании такового.
4. Сделать дочерний каталог текущим, т.е. войти в дочерний каталог.
5. Подать команду **dir** и объяснить появившиеся сообщения.
6. Выйти сразу в корневой каталог.
7. Вернуться в исходный каталог.

copy – команда копирования файлов, также используется и для объединения (конкатенации) файлов, создания новых текстовых файлов, передачи параметров на отдельные устройства и т.д. Формат команды

copy *Идиск:путь\имя_файла.расш Пдиск:путь\имя_файла.расш*

Указывается входная спецификация Идиск: путь\имя_файла.расш (И – источник), выходная спецификация Пдиск:путь\имя_файла.расш (П-приемника). В спецификациях источника и приемника возможно применение шаблонов групповых операций. Например,

copy a:*.txt c:\proba копирование с дискеты всех текстовых файлов с именами, начинающимися с test и расширением txt;

copy a:*.* nul копирование всех файлов с дискеты a: на несуществующее псевдоустройство **nul** с целью проверки их “читаемости”. Команда очень полезна, когда копируется очень важная информация для дальнейшего использования.

Для объединения (конкатенации) файлов используется другой формат

copy *И1диск:путь\имя_файла.расш + И2диск:путь\имя_файла.расш +... Пдиск:путь\имя_файла.расш*

Файлы, подлежащие объединению, задаются своими спецификациями И1…, И2…и т.д. между которыми ставится знак +. Результат (объединенный файл) формируется по спецификации приемника П…. Если спецификация приемника отсутствует, то результат формируется по спецификации И1….

Оба варианта команды могут использовать ключи **/a**, **/b**, **/v**. Ключ /a рассматривает обрабатываемые файлы, как текстовые в кодах ASCII, ключ /b – как двоичные. Это следует учитывать, особенно при объединении текстовых файлов. Если объединяются текстовые файлы, то признаки окончаний этих файлов (<Ctrl>+Z или F6) изымаются и остается только последний. При объединении двоичных файлов признаки окончаний файлов не изымаются. Ключ /v включает проверку правильности результата после копирования или объединения.

Создание текстовых файлов можно выполнить следующей командой **copy con** *диск:путь\имя_файла.расш.*

Например, команда **copy con f1.txt** создаст файл **f1.txt** в текущем каталоге. Текст файла будет получен с клавиатуры (консоли, **con**). Окончание набора файла должно заканчиваться нажатием клавиш <Ctrl>+Z или F6 (признак конца файла), а затем <Enter>. Недостатком применения этой команды является то, что редактировать можно только текущую строку файла. После нажатия клавиши <Enter> ранее введенные строки уже не доступны.

Интересна следующая модификация этого варианта применения команды **copy con prn**, которая позволяет реализовать режим пишущей машинки. Текст, набираемый на клавиатуре, поступает на принтер и, если используется матричный принтер, то распечатывается посимвольно. С помощью этой же команды можно производить передачу и установку значений некоторых параметров на отдельные устройства компьютера. При этом передача значений параметров производится в общем потоке данных, следующих в файле на устройство в виде управляющих Esc-последовательностей.

***Задание 3*.**

1. Создайте несколько небольших текстовых файлов, например f1.txt, f2.txt, f3.txt. Запишите в файл f3.txt содержимое текущего каталога.

2. Объедините содержимое файлов f1.txt и f2.txt, f2.txt и f3.txt в новые файлы.

type – команда просмотр содержимого текстовых файлов. Формат команды **type** *диск:путь\имя_файла.расширение*. Если в параметрах команды не задан диск и путь, а указано только имя файла, то подразумевается, что ведется работа с файлами текущего каталога. В параметре команды имена файлов должны быть определены однозначно, шаблоны групповых операций недопустимы.

***Задание 4*.**

1. Создать новый рабочий каталог.

2. Создать различными методами несколько текстовых файлов.

3. Записать в конец одного из файлов текущую дату и время.

4. Используя команду, убедиться в правильности проведенных операций.

12) **del** (или **erase**) – команда удаления одного или нескольких файлов. Формат команды **del (erase)** *диск:путь\имя_файла.расш*

Особенностями выполнения команд служат следующие ситуации:

а) если в команде не указан путь, то подразумевается, что удалению подлежит (подлежат) файл или файлы текущего каталога;

б) в спецификации файла допускается использование групповых операций. Спецификация *.* обозначает, что удалению подлежат все файлы выбранного каталога;

в) если в командах отсутствует спецификация файла, то командный процессор выдает предупреждение “Ошибка в синтаксисе команды”;

г) команды del (erase) не удаляют каталоги (подкаталоги). Они используются только для удаления в них файлов.

13) **rmdir** (или **rd)** – команда удаления каталога. Формат команды **rmdir** (**rd**) *диск:путь*.

Указываемый в команде путь должен завершаться именем удаляемого каталога. Удаляемый каталог должен быть пустым (содержать элементы . и ..). Нельзя удалить корневой каталог. Нельзя удалить текущий каталог, т.е. нельзя удалить каталог, находясь в нем.

3. Основы разработки командных файлов

Командный файл – это группа последовательных команд настройки компьютера на определенный режим. В простейшем случае командный файл может быть представлен в виде определенной последовательности отдельных команд ОС. Разработка командных файлов является мощным средством автоматизации подготовительных работ пользовате-

лей по настройке среды их работы. В Unix-подобных операционных системах вместо термина командные файлы чаще используется термин скрипты.

При разработке командных файлов следует руководствоваться следующими правилами:

1) Вызов на исполнение командного файла осуществляется командой следующего формата:

диск:\полный_путь_к_ком.файлу\имя_ком.файла [p1 p2 ...p10].

Содержимое в квадратных скобках указывает, что может иметь до 10 фактических параметров, замещающих формальные параметры, присутствующих в тексте файла (см. ниже). Команда **Shift** позволяет снять это ограничение.

2) Имя командного файла образуется по обычным правилам, а расширением должно быть только bat (сокращение от слова batch – пачка).

3) Если текущим является каталог (папка), содержащий командный файл, то полный путь к командному файлу можно не указывать.

4) Командный файл выполняется командным процессором строка за строкой.

5) Выполнение командного файла может быть прекращено командами <Ctrl>+<Break> или <Ctrl>+C.

6) Из командного файла можно вызывать другой командный файл командой **Call** (с возвратом) или обычной командой вызова (без возврата).

7) Командный файл может содержать любые внешние и внутренние команды операционной системы, а также специальные внутренние команды.

8) Формальные параметры, включаемые в строки командного файла, имеют вид %0, %1 и т.д. до %9. Фактические значения параметров вводятся в строке вызова командного файла; вводимые параметры подставляются на место формальных параметров %1, %2 и т.д. по порядку. На место формального параметра %0, если он встречается в тексте командного файла, подставляется имя самого командного файла.

9) Для обращения к переменным окружения их имена следует заключать в знаки %, например, %TEXT%.

10) Перед выполнением очередной строки командного файла ее значение выводится на экран. Вывод любой строки командного файла на экран подавляется, если строка начинается с символа @.

Для построения командных файлов широко используются специальные внутренние команды операционной системы **Echo, Goto, For, If, Pause, Rem, Shift**, а также внешние **Find, Sort, Mode, More** и др.

Рассмотрим особенности применения специальных команд.

Команда **Echo**. Служит для отключения “эха” на экране дисплея, т.е. она не выводит лишнюю информацию на монитор (блокирует выдачу на экран последовательностей команд, включенных в командный файл, и текстовых сообщений при выполнении этих команд). Форматы команды:

echo off – запрет вывода на экран;

echo on – разрешение вывода на экран;

echo (без параметров) – запрос состояния эха (on или off);

echo + текстовое сообщение – вывод текстового сообщения на экран.

Echo достаточно часто встречается в командных файлах, но ее простота обманчива. При ее использовании следует помнить:

1) при запуске системы по умолчанию устанавливается режим “echo on”;

2) режим “echo off” действует только до конца командного файла или до очередного переключения режима командой **echo on**;

3) echo влияет только на вывод сообщений командного файла, но не влияет на вывод сообщений из программ пользователей, даже если они используют команды операционной системы;

4) для подавления самой команды **echo off** надо поставить впереди @.

***Задание 5*.**

Для уяснения особенностей работы команды **echo** выполнить следующую последовательность операций:

1) в командной строке выполните команду **echo off**;

2) наберите и выполните, например, команду **dir** (обратите внимание, что в строке приглашения остается только мигающий курсор, а само действие команды не блокируется);

3) поэкспериментируйте с другими командами операционной системы;

4) наберите и выполните команду **echo on** (приглашение восстановится).

***Задание 6*.**

1) Используя команду **copy con**, создайте текстовый файл TEST1.BAT со следующим содержимым:

```
echo off
cls
echo Hello!
echo:
echo Hello again!
```

2) Выполните его в автоматическом режиме и объясните появление фрагментов данных на экране.

3) Заменив первую команду **echo off** на **echo on**. Снова выполните его в автоматическом режиме и объясните появление фрагментов данных на экране.

Команда **Rem** (Remark – примечание). С помощью этой команды имеется возможность в текст командного файла вводить комментарии. Командный процессор полностью игнорирует всю информацию, которая размещается за словом **rem**. Команда очень полезна, когда в командный файл включаются пояснения, описания работы файла или отдельных его команд, тестирования и отладки. Например, появление в тексте командного файла команды **rem echo off** означает, что по умолчанию действует команда **echo on**.

***Задание 7*.**

1) Создайте командный файл TEST2.BAT со следующим содержимым

```
echo off
rem Это первый комментарий
cls
rem Это второй комментарий
echo Hello!
rem Это третий комментарий
echo:
echo Hello again!
rem Это четвертый и последний комментарий
```

2) Выполните этот файл, а затем замените первую команду **echo off** на **echo on** (или замените на **rem echo off**) и снова выполните файл. Сравните и объясните различия в выводимой информации на экране дисплея.

Команда **Pause** используется для приостановки выполнения командного файла. Формат команды: ***Pause*** *сообщение.*

При остановке работы командного файла на экране появляется текст строки сообщения в режиме echo on, а под ним надпись “Press any key to continue” – Нажмите любую клавишу для продолжения (работы файла).

Команду полезно использовать в тех случаях, когда, например, на экран дисплея выводится большое количество информации порциями по страницам, чтобы пользователь мог ее прочитать, осмыслить и перейти к следующему фрагменту. Команда полезна и в случаях, когда необходимо выполнить какие-то вспомогательные действия, например,

….

@
pause Установите дискету с …. в дисковод A:
@ echo off
…..

Кроме того, ее можно использовать и для управления работой командного файла. Если в ответ на команду **pause** нажать <Ctrl>+C, то появляется вопрос “Terminate batch job (Y/N)?”. Выбор **Y** – останавливает выполнение командного файла, а **N** – обеспечивает его продолжение.

В случаях, когда командный файл выполняется с частыми остановками и появление множества фраз “Press any key to continue” нежелательно, строка с командой может выглядеть как **Pause > nul**, то есть вывод переадресуется в несуществующее устройство **nul**.

Команда **Goto** позволяет изменить привычную последовательность выполнения операторов (команд) командного файла. Когда командный процессор встречает строку с оператором **goto**, то он просматривает все строки файла и отыскивает соответствующую метку – строку с двоеточием. Двоеточие может быть и не в первой позиции строки. Идентификатор метки должен иметь до восьми символов. Больше, чем восемь символов, в идентификаторе не воспринимается.

Команда **goto** может использоваться самостоятельно или совместно с операцией if. В следующем фрагменте показано самостоятельное ее использование. Например, командный файл TEST3.BAT имеет следующее содержимое

echo off
cls
goto met
echo эта строка выводиться не будет
echo эта строка тоже выводиться не будет
:met
echo эта строка будет выведена
echo эта строка также будет выведена

***Задание 8*.**

1) Выполните файл TEST3.BAT, обратите внимание, как организован “обход” двух команд **echo** без их выполнения. Организация подобных “карманов” позволяет программистам размещать несколько констант непосредственно в теле программы.

2) Замените строку **goto met** на **rem goto met** и снова запустите этот файл. Объясните различия в результатах.

Команда **IF** – условное выполнение команд, организует разветвление при выполнении командного файла. Формат оператора: **If** *условие команда*.

В качестве *условия* обычно используются:

1) проверка наличия файла. В этом случае в качестве *условия* записывается фраза **exist** *диск:путь\имя_файла.расш*;

2) проверка кода завершения отдельных программ по значению внутренней переменной ОС с именем **ERRORLEVEL**. В этом случае в качестве *условия* записывается фраза “errorlevel *значение*”. *Условие* считается истинным, если код завершения равен или больше параметра *значение.* Командами ОС, устанавливающими коды завершения, являются **backup, diskcomp, diskcopy, format, graftabl, keyb, replace, restore, setver, xcopy**. Значение переменной errorlevel может формироваться и многими вспомогательными (утилитами) и прикладными программами.

3) проверка идентичности двух символьных строк. Строка *условие* при этом записывается в виде: *строка_1==строка_2*.

Предваряя любому из перечисленных условий слово **not**, можно проверять противоположное условие.

Составим командный файл TEST4.BAT, проверяющий наличие в корневом каталоге диска файла автозагрузки (autoexec.bat) и отображающий его содержимое. Обратите внимание на использование команды **goto** совместно с командой **if.**

```
echo off
cls
cd\
if exist autoexec.bat goto m1
echo файла AUTOEXEC.BAT на этом диске нет!
goto end
:m1
type autoexec.bat
:end
```

Другой модификацией этого файла, использующей параметр **not**, может быть файл TEST5.BAT:

```
echo off
cls
cd\
if not exist autoexec.bat goto error
type autoexec.bat
goto end
:error
echo ошибка
:end
```

Видоизменим файл TEST5.BAT таким образом, чтобы можно было отыскивать и просматривать нужный файл в любом каталоге. Учитывая, что файлы могут иметь большие размеры, превышающие емкость одного экрана, обеспечим поэкранный просмотр файлов. Имя нужного файла будем задавать в качестве параметра в строке вызова файла TEST6.BAT. Например,

TEST6.BAT proba.txt

Здесь имя искомого файла **proba.txt** служит фактическим параметром, значение которого должно заменить формальные параметры **%1** внутри командного файла. Напомним, что таких параметров строка вызова может иметь от **%1** до **%9**.

```
echo off
cls
if /%1==/ goto error1
rem if not exist autoexec.bat goto error2
type %1|more
goto end
:error1
echo Вы забыли указать имя искомого файла!
goto end
:error2
echo файла %1 на этом диске нет!
:end
```

Команда **For** применяется для многократного выполнения отдельных команд. Она позволяет обрабатывать целые группы файлов. Команда имеет форматы:

For %%*переменная* **in** (*набор*) **do** *команда* – для строк командных файлов;

For %*переменная* **in** (*набор*) **do** *команда* – для режима командной строки (автономного выполнения команды).

Если в команде употреблено %*переменная*, то к команде добавляются имена файлов из списка. В качестве параметров команды используются:

переменная – однобуквенная переменная, последовательно принимающая значения слов или имен файлов, перечисленных в параметре (*набор*);

(*набор*) – одно или несколько символьных слов или спецификаций файлов. Спецификация файла имеет вид *диск:путь\имя_файла.расш.* Допускаются шаблоны групповых операций. Слова и спецификации файлов разделяются пробелами или запятыми. Максимальная длина строки *набора* – не более 127 символов;

команда – команда ОС, выполняемая для каждого слова или файла из параметра *набор.*

***Задание 9*.** В созданном каталоге сформируйте 3-4 текстовых файла, различающихся размерами и содержанием. Затем в режиме командной строки выполните команду: **For %d in (*.txt) do type %d**. Измените *набор,* вместо шаблона файлов задайте имена файлов перечислением.

***Задание10*.** Создайте командный файл TEST7.BAT со следующим содержимым:

```
echo off
for %%S in (%1) do copy %%S prn
```

Произведите запуск этого файла командой: **TEST7.BAT *.txt**. Поясните смысл включения в команду запуска строки ***.txt**, назначение всех знаков **%** в команде **for**.

Команда **Shift** (сдвиг) позволяет сдвигать строку параметров влево на один параметр. Обычно в командный файл можно передавать до 9 параметров, применение данной команды позволяет снять это ограничение. Смысл использования данной команды лучше всего показать на простом примере. Создадим командный файл TEST8.BAT, который отображает видимые ОС параметры (от %0 до %9) строки запуска файла. Сдвиг каждый раз строки параметров позволяет выводить 10 элементов строки, сдвинутой на одну позицию влево.

```
echo off
:start
if /%0== / goto end
echo %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
shift
goto start
:end
```

Вызов этого файла будем проводить командой

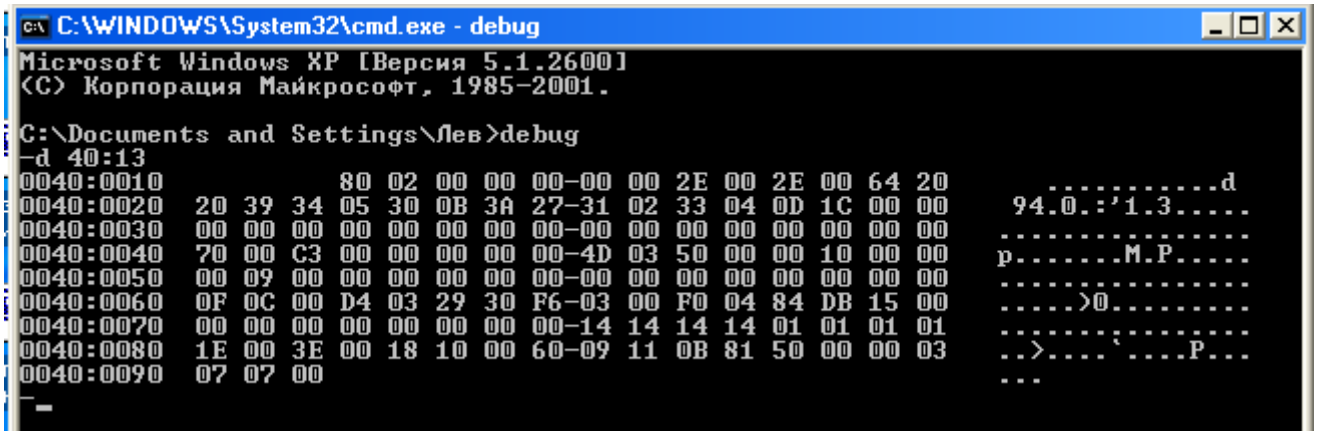
```
>test8.bat a s d f g h j k l z x c v b n m
```

Однобуквенные параметры командной строки можно интерпретировать как имена каких-то файлов, в данном случае несуществующих. Но следует учитывать, что имена файлов, используемые в качестве параметров командной строки не должны содержать расширений.

Лабораторная работа 2. Операционная система Windows. Использование программы Debug

1. Просмотр памяти

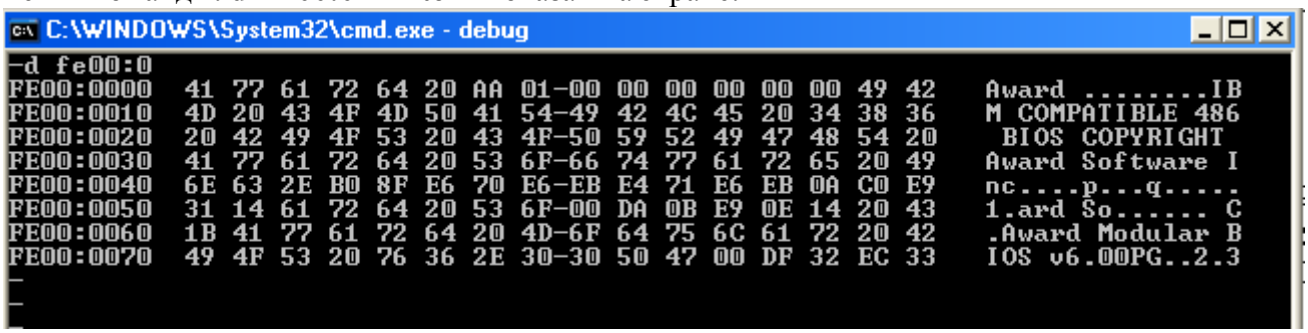
1) Проверка объема основной памяти DOS. После загрузки DOS эта информация находится в 413 и 414 байтах памяти (младший и старший байты). Задаем адрес сегмента: 400 (последний нуль следует отбросить) и смещения: 13. Набираем команду **-d (display) 40:13** и нажимаем Enter. Ввод данных строчными или прописными буквами не имеет значения, поскольку DOS в командной строке воспринимает символы ASCII, независимо от выбранного регистра. На эту команду получаем следующую картину на экране дисплея:



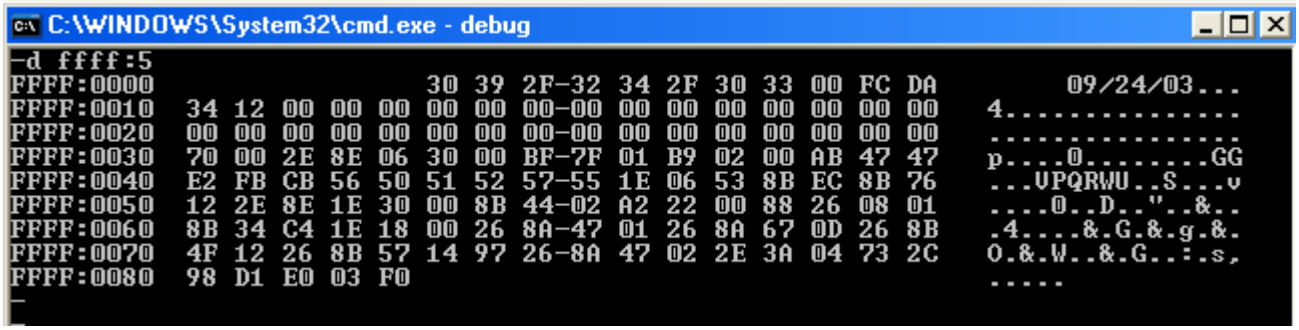
В каждой строке текста содержатся адреса 16-ти байтов, их шестнадцатеричное и символьное представление, если таковое существует в коде ASCII.

В первом и втором байтах (413 и 414) распечатки содержатся цифры 80 и 02, то есть с учетом “вращения” байтов получаем число 0280h Кбайт в шестнадцатеричном представлении или переводя в десятичное представление 2*16*16+8*16+0=640 Кб. Что и должно быть!

2) Проверка “серийного номера и даты копирайта”. Эта информация находится в ПЗУ, начиная с адреса FE000h. Там содержится семизначный номер компьютера, дата копирайта, фирма изготовитель, данная информация является символьной. Результат выполнения команды: **d FE00:0 <Enter>** показан на экране:



3) Проверка даты “прошивки” ROM BIOS (ПЗУ). Информация находится по адресу: FFFF5h, формат даты – американский (mm/dd/yy). По результату команды в верхней строке в символьном представлении читаем интересующую нас дату:



4) Поиск в памяти. Если требуется определить местоположение в памяти какой-либо информации (например, какого-то слова), то можно набрать и выполнить команду **S** (search – поиск), задав адресные границы поиска и ключевое слово. Например, выполнение команды: **s F000:0 L FFFF “IBM”**, в которой **F000:0** – начальный адрес памяти ПЗУ, **FFFF** – конечный адрес, а **IBM** – ключевое слово, позволяет получить следующие результаты:

Ниже выполненной команды следует перечень адресов, в которых содержится искомое ключевое слово. Команду **S** можно использовать, например, для поиска вирусов, если известны их “следы” и т.п.

Таким образом, используя данные команды можно определить модель и “возраст” компьютера, и просмотреть содержимое любой ячейки памяти.

5) Работа часов реального времени. Время, измеряемое компьютером, формируется на основе отсчетов счетчика часов реального времени. Четыре байта этого счетчика располагаются в оперативной памяти, начиная с адреса **0046Ch**.

Значения счетчика времени корректируется по каждому сигналу от таймера с частотой 18,2 имп/с (более точно 18,206481). Тактовая частота системного таймера персонального компьютера составляет 1,19318МГц. Она кратна основной частоте, принятой в телевидении (14,31818 МГц) и составляет 1/12 этой частоты. По результатам изменение значений байтов счетчика наглядно видно, что время “идет” вперед.

Задание. Запишите два показания счетчика через одну минуту. С учетом шестнадцатеричного представления чисел определите разницу этих значений. Переведите полученный результат в десятичную систему счисления, поделите на величину 60*18,2 и убедитесь, что темп изменения отсчетов действительно соответствует темпу изменения реального времени.

2. Выявление связей между ассемблерным кодом программы, ее машинным кодом и содержимым основных регистров при выполнении команд программы

Пусть имеется следующая программа:

Адрес	Ассемблерный код	Машинный код	Содержание
0100	MOV AX,123	B8 23 01	AX:= 123h
0103	ADD AX,0025	05 25 00	AX:= 123h + 25h =148h
0106	MOV BX,AX	89 C3	BX:= AX = 148h
0108	ADD BX,AX	01 C3	BX:= BX + AX=296h
010A	MOV CX,AX	89 C1	CX:= BX = 296h
010C	SUB CX,AX	29 C1	CX:= CX – AX=148h
010E	SUB AX,AX	29 C0	AX = 0
0110	NOP	90	
0111	RETF	CB	

Обратим внимание при выполнении заданий на следующие моменты: 1) как вычисляются адреса команд, 2) как числовая информация размещается в памяти и 3) как должно меняться содержимое регистров AX, BX, CX?

Ввод программы в память ПК. Сначала необходимо задать значение регистра **CS** в диапазоне (**0400…9FF0**), за пределами этого диапазона находятся зоны, контролируемые DOS. Последовательно строку за строкой ввести машинные коды программы. Для этого используется команда **E** (edit – редактирование):

E CS:100 B8 23 01 05 25 00 <Enter>
E CS:106 89 C3 01 C3 89 C1 <Enter>
E CS:10C 29 C1 29 C0 90 CB <Enter>

Далее вызвать команду **R** (Registr) – чтения содержимого регистров, включая и регистр флагов. Это позволит просмотреть исходное состояние всех регистров общего назначения перед выполнением программы. Содержимое регистров **CS** и **IP** определяет

адрес очередной выполняемой команды MOV AX,123, адрес, машинный код и содержание которой приведено после перечня регистров. Если теперь подать команду **T** (трассировка или покомандное выполнение программы с остановками), то можно увидеть последовательное изменение содержания регистров в соответствии с командами программы.

Объясните, как и почему меняется содержимое: 1) регистров после выполнения очередной команды программы; 2) регистра флагов?

Замечание. Если потребуется повторить трассировку указанной программы, то следует выполнить следующее:

1) выполнить **R IP**, для того, чтобы просмотреть содержимое регистра IP;

2) вводим **0100 <Enter>** тем самым регистр IP вновь получает значение 0100, и можно снова повторить трассировку.

3. Пример выполнения программы BIOS

Рассмотрим фрагмент программы BIOS, определяющей объем основной памяти DOS компьютера, не вручную, а по прошитой программе.

Программа работает по прерыванию INT 12h. В ранних версиях DOS это прерывание обрабатывалось обработчиком прерываний по следующей программе:

```
STI
PUSH DS
MOV AX, 0040
MOV DS, AX
MOV AX, [0013]
POP DS
IRET
```

Обратите внимание на начало и конец программы, команда **PUSH DS** обеспечивается сохранение старого значения регистра **DS**, а командой **POP DS** восстанавливается это значение, команда **IRET** возвращается управление прерванной программе. Рассмотрим, какой последовательностью команд под управлением Windows реализуется теперь обработка этого прерывания, а также сам механизм обращения к обработчику прерывания по его вектору.

Лабораторная работа. Операционная система Windows. Мультипрограммирование, задания, процессы, потоки

1. Изучить возможности диспетчера задач (Task Manager)

Диспетчер задач позволяет получить обобщенную информацию об организации вычислительного процесса с детализацией до процессов. Однако он не позволяет отслеживать потоки.

Запуск. Щелкнуть правой кнопкой мыши по панели задач и выбрать строку Диспетчер задач, или нажать клавиши Ctrl+Alt+Del, или нажать последовательно Пуск -> Выполнить -> taskmgr.

На вкладке «Приложения» можно просмотреть запущенные приложения, завершить их работу (кнопка «Снять задачу»), переключиться между приложениями, запустить новое приложение.

Просмотр (мониторинг) процессов. Перейдите на вкладку «Процессы». Таблица процессов включает все процессы, запущенные в собственном адресном пространстве, в том числе все приложения и системные сервисы. Обратить внимание на процесс «Бездействие системы». Если требуется просмотреть 16-разрядные процессы (например, если вы используете 16-разрядный компилятор языка С), то в меню «Параметры» нужно выбрать команду «Отображать 16-разрядные задачи».

Выбор просматриваемых показателей (характеристик). С помощью команды «Выбрать столбцы» меню Вид установить флажки рядом с показателями, которые требуется отображать. Рассмотрите самостоятельно различные показатели.

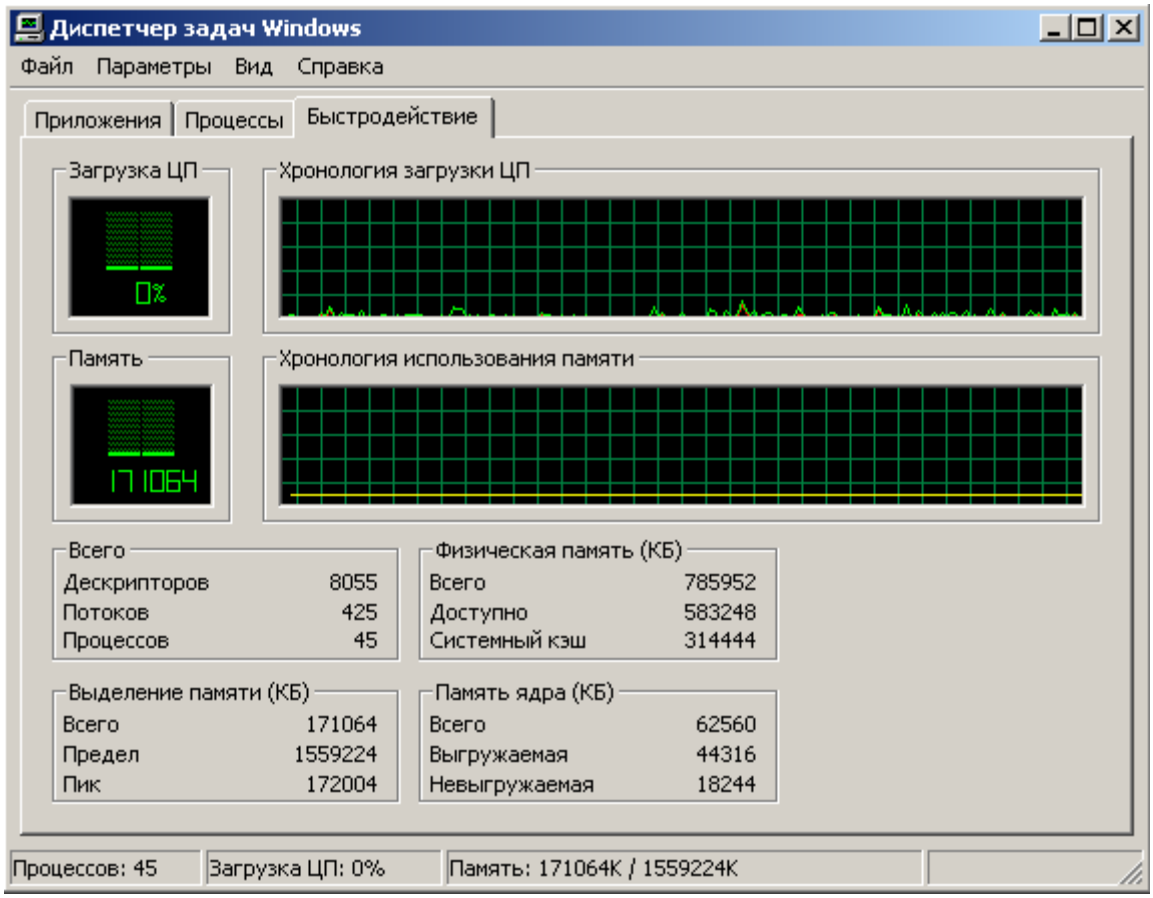
Просмотр процессов приложения. 1) Запустите Open/MS Office. Щелкнув правой клавишей мыши по названию приложения, и в появившемся контекстном меню выбрав строку «Перейти к процессам» произойдет переход на вкладку «Процессы». Можно просмотреть число потоков (2-3) и другие характеристики процесса.

Изменение приоритета процесса. 2) На вкладке «Процессы» щелкнув правой клавишей мыши по названию процесса и выбрать в контекстном меню строку «Приоритет» можно изменить приоритет, выбрав вместо значения «Базовый приоритет» новое значение (обратить внимание на предупреждение).

Изменение скорости обновления данных. 3) Войдя в меню Вид и выбрать команду «Скорость обновления» можно задать требуемую скорость (высокая – каждые полсекунды, обычная – каждую секунду, низкая – каждые 4 секунды, приостановить – обновления нет).

Объясните затраты ОС в связи со скоростью обновления.

Получение обобщенной информации об использовании основных ресурсов компьютера. 1) Перейдите на вкладку «Быстродействие». Верхние два окна показывают интегральную загрузку процессора и хронологию загрузки, а нижние – те же показатели, но по использованию памяти.



2) Для просмотра использования процессора в режиме ядра войдите в меню «Вид» и щелкнуть на строке «Вывод времени ядра».

Задание.

1) Для одной из задач (приложения) определить PID, загрузку ЦП, время ЦП, базовый приоритет процесса, использование памяти, хронологию использования ЦП в режиме ядра, число страничных нарушений, количество операций ввода-вывода, размер вирту-

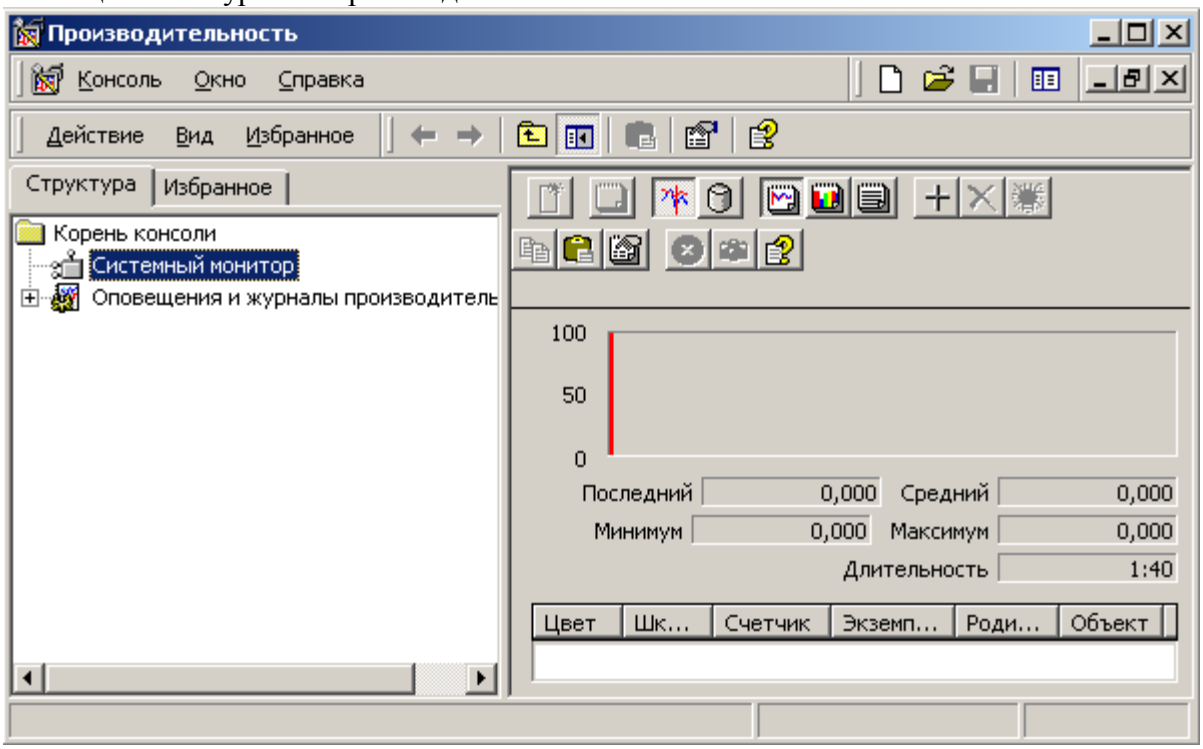
альной памяти, число исполняющихся потоков. Изменив приоритет процесса, проанализируйте, влияет ли это на время выполнения приложения.

2) Монопольно выполнить каждую из 3-х задач, определить время их выполнения. Запустить одновременно три задачи (друг за другом), определить время выполнения пакета. Объяснить полученный результат.

2. Изучить возможности оснастки «Производительность»

Оснастка «Производительность» позволяет анализировать вычислительные процессы с детализацией до потоков.

Запуск. Пуск -> Программы -> Администрирование -> Системный монитор. Откроется окно «Производительность», содержащее две оснастки: «Системный монитор» и «Оповещения и журналы производительности».



Оснастки являются инструментом системного администратора. Доступ к ним можно получить, запустив Microsoft Management Consol (mmc). Для этого нужно выполнить следующие действия: Пуск -> Выполнить -> Открыть mmc -> OK. Откроется окно Консоль1, в котором можно выбрать нужную оснастку.

Системный монитор. Позволяет анализировать вычислительный процесс, используя различные счетчики. Объектами исследования являются: процессор, кэш, задание, процесс, поток, физический диск, файл подкачки, очереди сервера, протоколы и др.

Просмотр и выбор объектов мониторинга и настройка счетчиков.

1) Открыть оснастку «Производительность». На панели результатов (правая панель) щелкнуть правой клавишей мыши и выбрать в контекстном меню строку Добавить счетчики или щелкнуть на кнопке Добавить (значок +) на панели инструментов.

2) В появившемся окне «Добавить счетчики» выбрать объект мониторинга, например процессор, а затем выбрать нужные счетчики из списка «Выбрать счетчики из списка», например, % времени прерываний, нажимая кнопку «Добавить». Для потока можно определить: число контекстных переключений в секунду; состояние потока (для построения графа состояний и переходов); текущий приоритет (для анализа его изменения); базовый приоритет; % работы в привилегированном режиме и др.

Нажав кнопку «Объяснение», можно получить информацию о счетчике. При выборе нескольких однотипных объектов, например потоков, нужно их указать в правом поле «Выбрать вхождения из списка».

Настройка внешнего вида. Просмотр информации производительности возможен в виде графика, гистограммы и отчета. Для настройки внешнего вида окна нужно щелкнуть на графике правой кнопкой мыши и выбрать команду свойства.

На вкладке «Общие» можно задать вид информации (график, гистограмма, отчет), отображаемые элементы (легенда, строка значений, панель инструментов), данные отчета и гистограммы (максимальные, минимальные и т. д), период обновления данных и др. На вкладке «Источник» задается источник данных. На вкладке «Данные» можно для каждого счетчика задать цвет, ширину линии, масштаб и др. На вкладке «График» можно задать заголовок, вертикальную и горизонтальную сетку, диапазон значений вертикальной шкалы. На вкладках «Цвета и шрифты» можно изменить набор цветов и шрифт.
Режимы «График» и «Гистограмма» не всегда удобны для отображения результатов анализа, например, при большом количестве счетчиков, меняющих свое значение в разных диапазонах величин. Режим «Отчет» позволяет наблюдать реальные значения счетчиков, так как не использует масштабирующих множителей. В этом режиме доступна только одна опция – изменение интервала опроса.
Задание. **Исследовать свои приложения. Определить характеристики процессов: % загрузки процессора (в пользовательском и привилегированном режиме), % времени прерываний, количество прерываний, базовый приоритет, обращения к диску, время выполнения процесса.**

3. Изучить возможности представления информации в оснастке «Журналы и оповещения производительности», которая содержит три компонента: Журналы счетчиков, Журналы трассировки и Оповещения. Данные, созданные при помощи оснастки можно просматривать как в процессе сбора, так и после его окончания.

Создание журнала счетчиков. Файл журнала счетчиков состоит из данных для каждого указанного счетчика на указанном временном интервале. Для создания журнала необходимо выполнить следующие действия:

1) запустить оснастку «Производительность»;

2)дважды щелкнуть на значке «Оповещения и журналы производительности»;

3) выбрать значок «Журналы счетчиков», щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт «Новые параметры журнала»;

4) в открывшемся окне ввести имя журнала и нажать кнопку OK;

5) в новом окне на вкладке «Общие» добавить нужные счетчики и установить интервал съема данных;

6) на вкладке «Файлы журналов» можно выбрать размещение журнала, имя файла, добавить комментарий, указать тип журнала и ограничить его объем. Возможны следующие варианты:

текстовый файл – CVS (данные сохраняются с использованием запятой в качестве разделителя);

текстовый файл – TSV (данные сохраняются с использованием табуляции в качестве разделителя);

двоичный файл для регистрации прерывающейся информации;

двоичный циклический файл для регистрации данных с перезаписью;

7) на вкладке «Расписание» выбрать режим запуска и остановки журнала (вручную или по времени). Для запуска команды после закрытия журнала установить флажок Выполнить команду и указать путь к исполняемому файлу;

8) после установки всех значений нажать кнопки Применить и OK.

Задание. Исследовать свои приложения с помощью Журналов счетчиков, выбрав счетчики: % загруженности работы процессора в привилегированном и пользовательском режимах, % времени прерываний, % использования выделенной памяти, частота обращений к диску, скорость обмена с диском.

1) Запустить журнал (частота съема данных 10 сек., файл типа CVS).

2) Запустить исследуемую программу.

3) Через 2-3 мин. Остановить журнал.

4) Просмотреть Результаты, открыв файл в табличном процессоре (для удобства просмотра нужно, используя меню Формат, установить Перенос по словам для заголовков и растянуть ширину ячеек с числовыми данными). Объяснить полученные результаты.

5) Исследовать программу еще раз, указав тип журнала – двоичный (по результатам постройте диаграмму).

Просмотр собранной информации в консоли Производительность.

1) дважды щелкнуть по значку Системный монитор;

2) щелкнуть правой клавишей мыши в правом поле и выбрать в контекстном меню строку Свойства;

3) перейти на вкладку Источник;

4) щелкнуть на кнопке Файл журнала и указать его размещение, используя кнопку Обзор;

5) нажать кнопку Диапазон времени и выбрать диапазон представления результатов, передвигая левую и правую планки;

6) нажать кнопку OK;

7) добавить счетчики, выбрав их из журнала (необязательно сразу все, можно просматривать отдельно каждый счетчик или несколько счетчиков);

8) просмотреть полученные диаграммы. Объяснить полученные результаты.

Создание журнала трассировки. В отличие от журналов счетчиков, журналы трассировки находятся в ожидании определенных событий. Для интерпретации содержимого журнала трассировки необходимо использовать *специальный анализатор*.

Для создания журнала трассировки необходимо выполнить следующие действия:

1) запустить оснастку Производительность;

2) щелкнуть на значке Журналы трассировки;

3) щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт Новые параметры журнала;

4) в открывшемся окне ввести произвольное имя журнала и нажать кнопку OK;

5) по умолчанию файл журнала создается в папке PerfLogs в корневом каталоге и к имени журнала присоединяется серийный номер;

6) на вкладке Общие указать путь и имя созданного журнала (по умолчанию оно уже есть);

7) на этой же вкладке выбрать События, протоколируемые системным поставщиком или указать другого поставщика;

8) на вкладке Файлы журналов выбрать тип журнала:

- файл циклической трассировки (журнал с перезаписью событий, расширение etl);
- файл последовательной трассировки (данные записываются пока журнал не достигнет предельного размера, расширение etl);

9) на этой же вкладке выбрать и размер файла;

10) на вкладке Дополнительно можно указать размер буфера журнала;

11) на вкладке Расписание выбрать режим запуска и остановки журнала (вручную или по времени).

Задание. Создать журнал трассировки для исследования своего приложения.

Оповещения. С помощью этого компонента можно установит оповещения для выбранных счетчиков. При превышении или снижении относительно заданного значения выбранными счетчиками оснастка посредством сервиса Messenger оповещает пользователя.

Для создания оповещений необходимо выполнить следующие действия:

1) щелкнуть на значке Оповещения;

2) щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт Новые параметры оповещений;

3) в открывшемся окне ввести произвольное имя оповещения и нажать кнопку OK;

4) на вкладке Общие можно задать комментарий оповещения и выбрать нужные счетчики;

5) в поле Оповещать, когда значение выбрать предельные значения для счетчиков;

6) в поле Снимать показания каждые выбрать период опроса счетчиков;

7) на вкладке Действие можно выбрать действие, которое будет происходить при запуске оповещения, например, Послать сетевое сообщение и указать имя компьютера;

8) на вкладке Расписание выбрать режим запуска и остановки наблюдения.

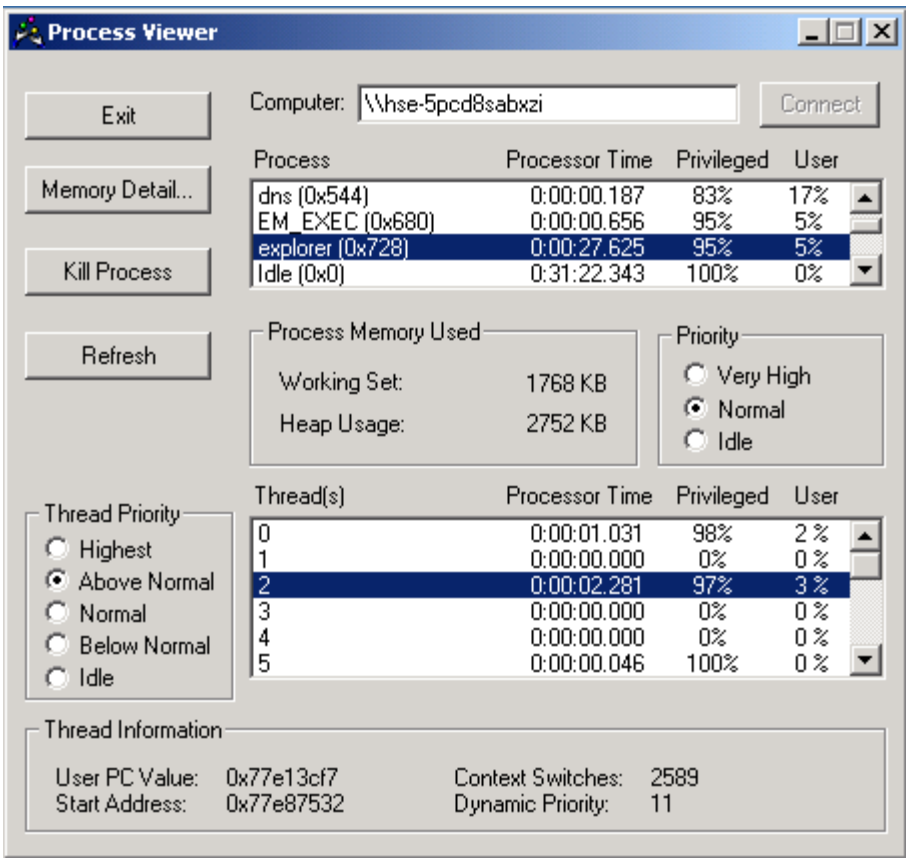
Задание. Создать Оповещения по выбранным счетчикам для своего приложения. Объяснить результаты.

Лабораторная работа 4. Операционная система Windows. Исследование заданий, процессов и потоков

1. Изучить возможности программы Process Viewer

Назначение программы. Программа позволяет увидеть процессы и потоки простым одноступенчатым способом, как на локальном, так и на удаленном компьютере сети.

В верхней части окна перечисляются все процессы, выполняемые на выбранном компьютере. Статистические данные включают количество процессорного времени и количество времени, проведенного в режиме пользователя и в режиме ядра. Под списком процессов приводятся статистические данные для выделенного процесса, включающие количество используемой памяти и приоритет процесса.



В нижней части окна перечисляются потоки выделенного процесса. Здесь также выводятся статистические данные по использованию процессора, но уже потоками. Слева от перечня потоков даются текущие значения их приоритетов. Поток получает тоже значение приоритета, как и связанный с ним процесс, однако это базовый приоритет, который во время выполнения потока может меняться операционной системой. Динамический приоритет выделенного потока и другая информация о нем помещается в нижней части окна.

Щелчок на кнопке «Memory Details..» выводит окно с информацией об использовании памяти процессом. Поле адресного пространства (User Address Space) содержит наименование просматриваемого адресного пространства. Значение «Total Commit» указывает на память, используемую всем процессом.

Если щелкнуть по стрелке комбинированного окна, то можно увидеть перечень файлов .dll и .exe, используемых данным приложением. Выбрав один из этих элементов, можно увидеть, какая память используется данной частью приложения.

Задание. Запустить программу и просмотреть с ее помощью текущие характеристики процессов и потоков, в том числе свои задачи.

2. Изучить возможности программы Microsoft Spy++

Назначение утилиты. Программа обладает широкими возможностями, мы будем использовать ее для просмотра процессов, потоков и ресурсов, в том числе потоков пользовательского интерфейса (UI-потоки).

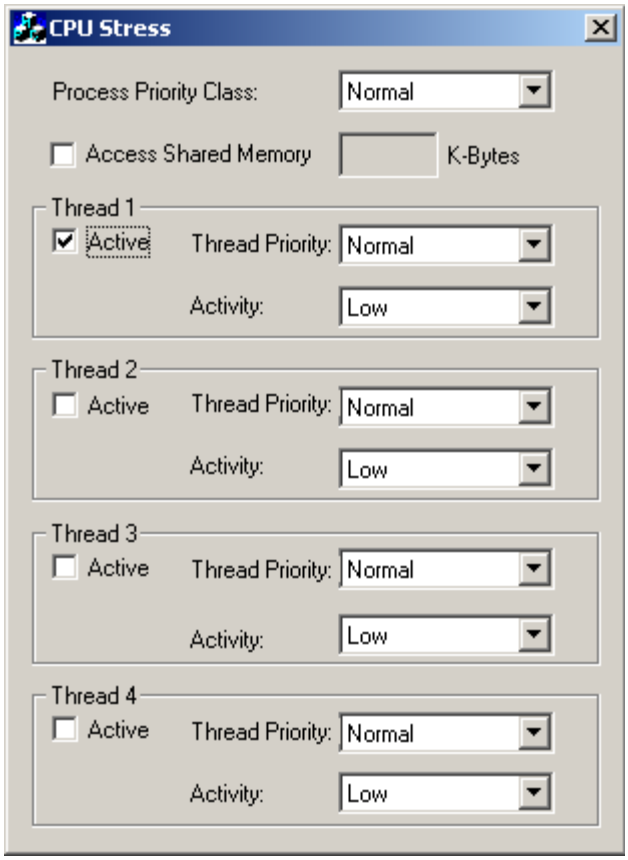
Использование утилиты. Запустите программу в режиме Process (меню Spy -> Processes). Найдите процесс Explorer и щелкнуть по значку + слева от него. На экране высветятся потоки процесса. Около потоков UI расположен значок +, у рабочих потоков он отсутствует.

Щелкнув по значку + потока UI, можно увидеть объекты, поддерживаемые этим потоком. Просмотреть свойства любого из этих объектов можно, щелкнув по нему правой клавишей мыши и выбрав из контекстного меню строку Свойства.

На вкладке Process можно прочитать идентификаторы процесса и потока, чтобы наблюдать за его производительностью.

3. Изучить возможности программы CPU Stress

Назначение утилиты. Программа CPU Stress нагружает процессор, чтобы проконтролировать его работу в заданных условиях. Как видно, программа рассчитана на запуск от одного до четырех потоков.

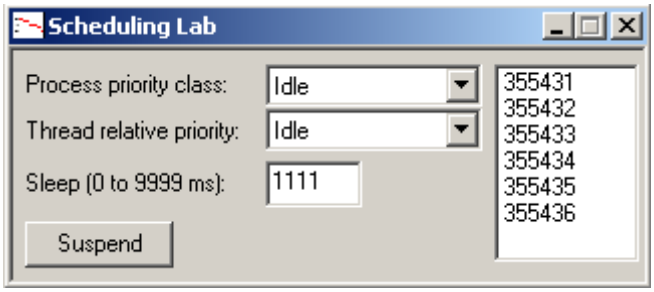


Кроме задания количества потоков, утилита позволяет менять приоритет потоков процесса и уровень их активности. Программа начинает работать сразу после запуска. Ее можно использовать для определения возможностей компьютера и их деградации во времени.

Задание. Используя возможности оснастки Производительность, получить диаграммы, характеризующие использование процессора при его нагрузке различным количеством потоков, меняя их активность и уровни приоритета. Создать журнал счетчиков, провести запись в формате CVS, передать файл в Excel и представить результаты в виде таблицы и графика.

4. Изучить возможности программы Scheduling Lab

Назначение программы. Программа позволяет проводить эксперименты с классами приоритетов процессов и относительными приоритетами потоков и исследовать их влияние на общую производительность системы.



Изначально первичный поток работает очень активно, и степень использования процессора доходит до 100%. Все действия потока сводятся к увеличению на 1 исходного значения и выводу текущего результата в правое окно списка.

Задание. Для анализа влияния изменения приоритета потока нужно запустить несколько экземпляров программы (3-4) и понаблюдать за нагрузкой процессора, используя оснастку Производительность. В начале теста процессор будет загружен на 100%, и все экземпляры программы получают примерно равные кванты процессорного времени. Если поднять класс приоритета одному из экземпляров программы, то большую долю процессорного времени начнет получать именно этот экземпляр, а аналогичные показатели для других экземпляров упадут. Однако они никогда не опустятся до нуля, поскольку действует механизм динамического повышения приоритета (убедиться в этом, построив диаграммы изменения приоритетов процессов).

Лабораторная работа 5. Операционная система Windows. Алгоритмы планирования процессов

1. Изучить возможности программы AlgOfComp по моделированию различных алгоритмов планирования процессов (потоков)

Назначение и возможности программы. Программа позволяет моделировать следующие алгоритмы распределения времени центрального процессора очереди выполняемых процессов: RR (Round Robin); FCFS (First Com –First Served); SJF (Short Job First) в двух вариантах. Первый – вытесняющий алгоритм, т.е. с прерыванием процесса по истечении кванта времени, второй – невытесняющий алгоритм, т.е. с выполнением короткой задачи до ее завершения.

В верхней части окна выбирается для исследования один из возможных алгоритмов. Среднее поле программы предназначено для ввода исходных данных. Здесь задается количество обрабатываемых процессов, причем для каждого процесса задаются время перехода в состояние “Готовность” (например по причине ожидания ввода-вывода) и время выполнения в условных единицах.

После ввода исходных данных по всем процессам и задания величины кванта времени процессора в таких же условных единицах следует нажать кнопку «**Произвести расчет**». В нижней части окна представляются полученные результаты. По каждому процессу даются среднее время ожидания и среднее время обслуживания (выполнения). Полученные результаты можно просмотреть в виде диаграммы, нажав кнопку «**Диаграмма**», а также можно сохранить в файле, нажав кнопку «**Сохранить**».

Файл можно просмотреть в программе Блокнот. В правой части диаграммы даются обобщенные данные по результатам выполнения процессов.

Задание. Провести исследование различных алгоритмов планирования процессов, используя программу AlgOfComp. В качестве исходных данных сформировать 3-4 про-

цессов со временем перехода в состояние готовность в диапазоне 1-5 усл. ед. и временем исполнения в диапазоне от 20 до 50 усл. ед. Исследования провести для различных значений квантов времени процессора 1, 2, 3, 4, 5. По полученным результатам построить графики в табличном процессоре. Провести анализ эффективности того или иного алгоритма в зависимости от величины кванта времени процессора и характеристик процессов.

2. Программа-пример Queue (иллюстрация использования мьютекса и семафора)

Назначение и возможности программы. Программа управляет очередью обрабатываемых элементов данных, используя мьютекс и семафор.

При инициализации программа создает четыре клиентских и два серверных потока. Каждый клиентский поток засыпает на определенный период времени, а затем помещает в очередь элемент данных. Когда в очередь ставится новый элемент, содержимое списка «**Client threads**» обновляется. Каждый элемент данных состоит из номера клиентского потока и порядкового номера запроса, выданного этим потоком. Первая запись в левом окне сообщает, что клиентский поток 1 поставил в очередь пятый запрос.

Серверные потоки ничего не делают, пока в очереди не появится хотя бы один элемент данных. Как только он появляется, для его обработки пробуждается один из серверных потоков. Их работа отображается в правом окне. Первая запись говорит о том, что нулевой серверный поток обрабатывает первый запрос от клиентского потока с номером 0. В приведенном примере серверные потоки не успевают обрабатывать клиентские запросы и очередь, в конечном счете, заполняется до максимума (10 элементов). Это происходит к тому моменту, когда клиентский поток 3 выдает свой пятый запрос.

Исходный код программы приведен в файле Module. Жирным черным и красным цветом выделены комментарии и строки, на которые нужно обратить внимание. Эти строки связаны с работой и созданием потоков, семафора и мьютекса. Очередью управляет класс CQueue. Метод *Append* пытается добавить элемент в очередь, но сначала он должен убедиться, что вызывающему потоку разрешен монопольный доступ к очереди. Для этого метод *Append* вызывает *WaitForSingleObject*, передавая описатель объекта-мьютекса, *m_bmtxQ*. Если функция возвращает WAIT_OBJECT_0, то значит поток получил монопольный доступ к очереди.

Далее метод *Append* пытается увеличить число элементов очереди, вызвав функцию *ReleaseSemaphore* и передав ей счетчик числа освобождений, равный 1. Если вызов *ReleaseSemaphore* проходит успешно, в очереди есть место и в нее помещается элемент. По окончании этой операции *Append* вызывает *ReleaseMutex*, чтобы и другие потоки могли получить доступ к очереди.

Для выборки элемента из очереди серверный поток вызывает метод *Remove* сначала этот метод должен убедиться, что вызывающий поток получил монопольный доступ к очереди и что в ней есть хотя бы один элемент. Поэтому метод *Remove* предварительно обращается к *WaitMultipleObjects*, передавая ей описания мьютекса и семафора. После освобождения этих объектов серверный поток может пробудиться.

Объект семафор отслеживает, сколько элементов находится в очереди. Это значение увеличивается, когда метод *Append* вызывает *ReleaseSemaphore* после добавления нового элемента в очередь и уменьшается при вызове *WaitMultipleObjects* из метода *Remove*.

Задание. Запустить программу Queue. Для более удобного просмотра ее работы, предварительно запустить программу CPU Stress, присвоив ей высший приоритет. В файле Module просмотреть выделенные жирным шрифтом участки, которые связаны с синхронизацией потоков. Объяснить работу программы.

Лабораторная работа. Операционная система Windows. Управление памятью

1. Общая информация об использовании памяти

Диспетчер задач Windows позволяет просматривать общее использование памяти на вкладке Быстродействие. Здесь отображается информация в трех разделах: «Выделение памяти», «Физическая память» и «Память ядра».

В первом разделе содержится три статистических параметра виртуальной памяти: 1) «Всего» – это общий объем виртуальной памяти, используемой как приложениями, так и операционной системой; 2) «Предел» – это объем доступной виртуальной памяти; «Пик» – наибольший объем памяти, использованный в течение сессии с момента последней загрузки.

В разделе «Физическая память» содержатся параметры, несущие информацию о текущем состоянии физической памяти машины. Эта статистика не имеет никакого отношения к файлу подкачки, следовательно, может являться хорошим индикатором ситуаций, когда его увеличение не даст эффекта. 1) «Всего» – это объем памяти, обнаруженный операционной системой на компьютере. 2) «Доступно» отражает память, доступную для использования процессами. Эта величина не включает память, доступную приложениям за счет файла подкачки. Каждое приложение требует определенный объем физической памяти и не может использовать только ресурсы файла подкачки. 3) «Системный кэш» сообщает объем, доступный кэш-памяти системы. Это объем физической памяти, оставленный операционной системой после удовлетворения своих потребностей.

В разделе «Память ядра» отображается информация о потребностях компонентов операционной системы, обладающих наивысшим приоритетом. Эти компоненты обычно работают с сервисом низкого уровня, типа прямого доступа к жесткому диску. Параметры «Память ядра» отображают потребности ключевых служб операционной системы: 1) «Всего» – это объем виртуальной памяти, необходимой операционной системе; 2) «Выгружаемая» несет информацию об общем объеме памяти, использованной системой за счет файла подкачки; 3) «Невыгружаемая» – объем физической памяти, потребляемой операционной системой. Необходимо помнить, что эти параметры относятся лишь к привилегированным службам, а не ко всему сервису системы в целом. Многие компоненты ОС работают как приложения. В большинстве случаев параметры «Память ядра» должны оставаться без изменений, если не меняется что-либо в ядре операционной системы (например, устанавливается новое устройство в компьютер). Глобальные изменения в этом разделе обычно являются сигналом возможного возникновения проблем.

С помощью Диспетчера задач можно также узнать объемы памяти, используемые процессами. Для этого нужно перейти на вкладку «Процессы», которая показывает список исполняемых процессов и занимаемую ими память, в том числе физическую память, пиковое (максимальное использование памяти) и виртуальную память.

Однако конкретное размещение процесса в виртуальной памяти с помощью Диспетчера задач узнать невозможно, нельзя также увидеть свободные, занятые страницы и блоки памяти, их размер и атрибуты защиты.

Информация, которую способен вывести диспетчер задач, не является полной. В ряде случаев ее достаточно для оптимизации системы, но есть несколько ограничений, характерных для Диспетчера задач:

1. Список процессов не полон. В окне Диспетчера задач представлены только процессы, зарегистрированные в Windows. В частности в этот список не включаются драйверы устройств и некоторые системные службы.

2. Требования к памяти отражают текущее состояние процесса. В списке отражены объемы памяти, занимаемые приложениями в текущий момент времени, а не их максимальные значения.

3. Отсутствуют статистические данные. Поскольку в Диспетчере задач не выводятся временные характеристики, а только мгновенная картина потребления памяти, нет возможности отследить ее изменение.

Утилита TaskList предоставляет более обширную информацию по сравнению с Диспетчером задач, но пользоваться ей сложнее. Запускается утилита из окна командной строки.

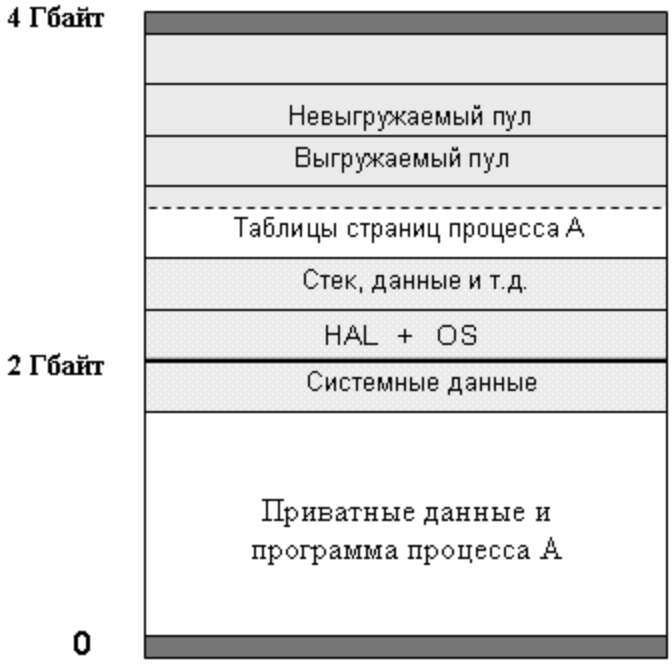
Вызов утилиты с аргументами позволит получить более полезную информацию. Например, параметр /M (модуль) позволит отобразить модули (обычно DLL), задействованные приложением. Параметр /FI обеспечит фильтрацию информации, выводимой утилитой, чтобы можно было видеть только интересующие записи и т.д. Получить информацию о параметрах утилиты TaskList можно обычным образом в окне командной строки.

Операционные системы Windows в Служебных программах содержат программу Сведения о системе, с помощью которой можно получить сведения об основных характеристиках организации памяти в компьютере.

В частности, здесь можно узнать полный объем установленной в компьютере физической памяти, общий объем виртуальной памяти и доступной (свободной) в данный момент времени виртуальной памяти, размещение и объем файла подкачки. Щелкнув по кнопке «Ресурсы аппаратуры», а затем «Память», можно получить сведения об использовании физической памяти аппаратными компонентами компьютера.

2. Архитектура памяти в Windows

В операционной системе W2K реализована сложная система виртуальной памяти. Предусмотрено множество функций Win32 для использования виртуальной памяти, часть исполняющей системы (менеджер памяти) плюс шесть выделенных потоков ядра для управления памятью. Каждый пользовательский процесс имеет собственное виртуальное адресное пространство размером в 4 Гбайт (адрес имеет 32 двоичных разряда). Конфигурация адресного пространства приведена на рисунке.



Нижние 2 Гбайт за вычетом примерно 256 Кбайт (системные данные – указатели и таймеры, используются совместно в режиме “только чтение”) доступны для программы и данных процесса. Верхние 2 Гбайт защищенным образом отображаются на память ядра операционной системы. Страницы виртуального адресного пространства имеют фиксированный размер (4 Кбайт для процессора Pentium) и загружаются по требованию. Белым цветом изображена область приватных данных пользовательского процесса. Затененные области представляют память, которая совместно используется всеми процессами.

Нижние и верхние 64 Кбайт каждого виртуального адресного пространства в обычном состоянии не отображаются на физическую память. Это делается для облегчения перехвата программных ошибок (выявления недействительных указателей, имеющих значения 0 или -1).

Верхние 2 Гбайт виртуального адресного пространства предназначены для ОС, включая код ядра, драйверы устройств, кэш-буферы ввода-вывода, данные, выгружаемые и невыгружаемые пулы (используемые для объектов, создаваемых ОС) и т.д. Эта область используется совместно всеми процессами, кроме таблиц страниц, которые являются индивидуальными для каждого процесса. Верхняя часть адресного пространства запрещена для записи в режиме пользователя, и по большей части и в режиме чтения.

Причина, по которой потокам доступна эта область, заключается в том, что когда поток обращается к системному вызову, он переключается в режим ядра, но остается все тем же потоком. Если сделать всю операционную систему и все ее структуры данных (как и весь пользовательский процесс) видимыми в адресном пространстве потока, когда он переключается в режим ядра, то отпадает необходимость в изменении карты памяти или выгрузке кэша при входе в ядро. Все, что нужно сделать, - это переключиться на стек режима ядра. Платой за более быстрые системные вызовы при данном подходе является уменьшение приватного адресного пространства для каждого процесса.

Адресное пространство, выделенное процессу в момент создания, практически свободно все (не зарезервировано). Поэтому чтобы воспользоваться какой-нибудь его частью, нужно выделить в нем определенные области, используя функцию Win32 API *VirtualAlloc*. Выделенные области всегда начинаются с 64-килобайтных страниц (предполагается, что это сведет к минимуму проблемы переноса системы на компьютеры такой архитектуры с большим размером страниц). Количество выделенного адресного пространства может быть меньше, чем 64 Кбайт, но оно должно состоять из целого числа страниц.

Для использования зарезервированного региона (области) адресного пространства ему нужно выделить физическую память и спроецировать ее на этот регион. Такая операция называется *передачей физической памяти*. Делается это с помощью той же функции *VirtualAlloc*. Передавая физическую память, нет необходимости отводить ее целому региону. Например, зарезервировав регион размером 64 Кбайт, можно передать физическую память только его второй и пятой страницам. Если физическая память, переданная зарезервированному региону, больше не нужна, ее освобождают вызовом функции *VirtualFree*.

Рассматривая физическую память современных ОС, нужно помнить, что они "умеют" имитировать память за счет дискового пространства, создавая страничный файл (paging file), который (с точки зрения прикладной программы) просто «увеличивает» объем памяти программы. Следовательно, физическую память следует рассматривать как данные, хранимые в дисковом файле со страничной структурой. Поэтому, когда приложение передает физическую память какому-нибудь региону адресного пространства, она на самом деле выделяется из файла, размещенного на жестком диске.

Таким образом, размер страничного файла – главный фактор, определяющий количество физической памяти, доступной приложению. Реальный объем оперативной памяти имеет меньшее значение. Однако, надо помнить о том, что процессор проецирует виртуальный адрес на физический только тогда, когда данные находятся в оперативной памяти.

Чем чаще системе приходится копировать страницы памяти в страничный файл и обратно, тем больше нагрузка на жесткий диск и тем медленнее работает ОС. Получается так, что система будет тратить все свое время на подкачку страниц вместо выполнения программы. Поэтому, увеличив объем оперативной памяти, можно снизить частоту обращения к диску и тем самым увеличить общую производительность системы.

Не следует думать, что страничный файл сильно увеличивается при одновременном выполнении нескольких программ за счет того, что система при каждом запуске приложения резервирует регионы адресного пространства, передает им физическую память, а затем копирует код и данные из файла программы (расположенного на диске) в физическую память, переданную из страничного файла. ОС действует не так, иначе бы на загрузку и подготовку программы к запуску уходило бы слишком много времени. На самом деле при запуске приложения система открывает исполняемый файл и определяет объем кода и данных. Затем резервирует регион адресного пространства и помечает, что физическая память, связанная с этим регионом, – сам exe-файл, т.е. вместо выделения какого-то пространства из страничного файла система использует *образ* exe-файла как зарезервированный регион адресного пространства программы. Благодаря этому приложение загружается очень быстро, а размер страничного файла удается существенно уменьшить.

Образ исполняемого файла (т.е. EXE- или DLL-файл), размещенный на диске и применяемый как физическая память для того или иного региона адресного пространства, называется *проецируемым в память файлом* (memory-mapped file). При загрузке EXE или DLL файлов система автоматически резервирует регион адресного пространства и проецирует на него образ файла. Помимо этого система позволяет проецировать на регион адресного пространства и файлы данных.

.

Задание.

1) Используя рассмотренные средства операционной системы, определите объем установленной физической памяти, объем виртуальной памяти, величину файла подкачки и его размещение в компьютере.

2) Определите, какие области физической памяти использует системная плата.

3) Запустить программу VMMap и просмотреть карту виртуальной памяти процесса на своем компьютере. Обратить внимание на свободные участки памяти на "концах" адресного пространства. Объясните их назначение.

3. Исследование виртуальной памяти

Программа SystemInfo. Назначение программы заключается в выводе системной информации о процессоре, их количестве и виртуальной памяти. Для этого программа вызывает функцию *GetSystemInfo*.

По полученной информации определить основные характеристики процессора компьютера, число процессоров в компьютере, активные процессоры компьютера, дискретность (granularity) выделения памяти процессу, размер страницы и размер виртуального адресного пространства процесса (по значениям минимального и максимального адреса приложения).

Программа VMStat в отличие от программы SysInfo, дает динамику изменения основных данных о физической и виртуальной памяти, обновляемых каждую секунду. Это делает программу пригодной для мониторинга памяти в системе. Для этого программа вызывает функцию *GlobalMemoryStatus*.

Элемент Memory load позволяет оценить, насколько занята подсистема управления памятью. Это число может быть любым в диапазоне от 0 до 100 (на практике от значения этого элемента толку немного). Элемент TotalPhys отражает общий объем физической (оперативной) памяти в байтах. Однако реально память компьютера несколько больше (проверить экспериментально на своем компьютере). Причина, по которой *GlobalMemoryStatus* не сообщает полный объем памяти заключается в том, что система при загрузке резервирует небольшой участок оперативной памяти, недоступный даже ядру. Этот участок никогда не сбрасывается на диск.

Элемент AvailPhys сообщает число байтов свободной физической памяти. Следующий элемент TotalPageFile дает максимальное количество байтов, которое может содержаться в страничном файле (файлах) на жестком диске (дисках). Свободное число байтов в страничном файле, которое может быть передано любому процессу, показывает элемент AvailPageFile.

Элемент TotalVirtual отражает общее количество байтов, отведенных под закрытое адресное пространство процесса. Значение 2 147 352 576 ровно на 128 Кбайт меньше 2 Гбайт. Два раздела недоступного адресного пространства – от 0x00000000 до 0x0000FFFF и от 0x7FFF0000 до 0x7FFFFFFF – как раз и составляют эту разницу в 128 Кбайт.

Последний элемент AvailVirtual – единственный элемент структуры, специфичный для конкретного процесса, вызывающего *GlobalMemoryStatus* (остальные элементы относятся исключительно к самой системе и не зависят от того, какой именно процесс вызывает эту функцию). При подсчете значения AvailVirtual функция суммирует размеры всех свободных регионов в адресном пространстве вызывающего процесса. В данном случае его значение говорит о том, что в распоряжении программы VMStat имеется 2 135 785 472 байтов свободного адресного пространства. Вычтя из значения TotalVirtual величину AvailVirtual, получим 11 567 104 байтов – такой объем памяти VMStat зарезервировала в своем виртуальном адресном пространстве.

Задание. Используя программу SystemInfo, определите объем виртуальной памяти, доступной процессу. Сравните эти данные с результатом программы VMStat. Совпадают ли эти значения? Если нет, то почему?

4. Использование виртуальной памяти

Операционная система Windows реализует три механизма работы с памятью:

1) виртуальную память – для операций с большими массивами объектов или структур;

2) проецируемые в память файлы – для операций с большими потоками данных (обычно из файлов) и для совместного использования данных несколькими процессами на одном компьютере;

3) кучи – для работы с множеством малых объектов.

Функции, работающие с виртуальной памятью, позволяют напрямую резервировать регион адресного пространства, передавать ему физическую память из страничного файла и присваивать любые допустимые атрибуты защиты.

Программа VMAlloc демонстрирует применение механизма виртуальной памяти для управления массивом структур.

Изначально для массива не резервируется никакого региона, все адресное пространство свободно, что отображается на карте памяти (Memory map) белым цветом. При нажатии кнопки Reserve region (50,2KB structures) программа VMAlloc вызовет функцию *VirtualAlloc* для резервирования региона, что сразу отразится на карте памяти, которая окрасится серым цветом.

Активными становятся все кнопки и теперь можно ввести индекс и нажать кнопку Use. Например, введем индексы 5, 15 и 41. При этом по адресу, где должен располагаться указанный элемент массива, передается физическая память. Это отображается в окне программы черным цветом на карте памяти.

Аналогично можно занять следующие элементы зарезервированной памяти. Любой элемент массива, помеченный как занятый, можно освободить щелчком по кнопке Clear. Однако это не приведет к возврату физической памяти, переданной под элемент массива. Дело в том, что каждая страница содержит несколько структур и освобождение одной структуры не влечет за собой освобождения других. Если бы память была возвращена, то пропали бы и данные, содержащиеся в остальных структурах.

Освобождение структуры приводит к тому, что ее элемент sInUse принимает значение False. Это нужно для того, чтобы функция сбора мусора могла вернуть не используемую больше физическую память. Делается это с помощью кнопки Garbage collect (Сборка мусора). Чтобы посмотреть, как это работает, очистим элемент массива с индексом 41. Карта памяти пока не изменится. Если щелкнуть теперь по кнопке Сборка мусора, то карта памяти обновится.

5. Проецируемые в память файлы

Как и виртуальная память, проецируемые файлы позволяют резервировать регион адресного пространства и передавать ему физическую память. Различие между этими механизмами заключается в том, что в последнем случае физическая память не выделяется из страничного файла, а берется из файла, уже находящегося на диске. Как только файл спроецирован в память, к нему можно обращаться так, будто он целиком в нее загружен.

Проецируемые файлы применяются в следующих случаях:

1) загрузка и выполнение EXE- и DLL-файлов. Это позволяет существенно экономить как на размере страничного файла, так и на времени, необходимом для подготовки приложения к выполнению;

2) организация доступа к файлу данных, размещенному на диске. При этом можно исключить операции файлового ввода-вывода и буферизацию его содержимого;

3) разделение данных между несколькими процессами, выполняемыми на одной машине.

Рассмотрим процесс загрузки и выполнения EXE- и DLL-файлов. При выполнении функции *CreateProcess* (создать процесс) операционная система действует следующим образом:

1) отыскивается EXE-файл, указанный в вызове функции и, если он не найден, новый процесс не создается, а функция возвращает значение FALSE;

2) если файл найден, создается новый объект ядра “процесс”;

3) создается адресное пространство нового процесса;

4) резервируется такой регион адресного пространства, чтобы в него поместился данный EXE-файл. Желательное расположение этого региона указывается внутри самого EXE-файла. По умолчанию базовый адрес EXE-файла – 0x00400000. При создании испол-

няемого файла приложения базовый адрес может быть изменен параметром компоновщика /BASE;

5) Система отмечает, что физическая память, связанная с зарезервированным регионом, - EXE-файл на диске, а не страничный файл.

Спроецировав EXE-файл на адресное пространство процесса, система обращается к разделу EXE-файла, содержащему список DLL, которые обеспечивают программе необходимые функции. После этого система, вызывая функцию *LoadLibrary*, поочередно загружает указанные DLL-модули (а при необходимости и дополнительные). Каждый раз, когда для загрузки DLL вызывается *LoadLibrary*, операционная система выполняет действия, аналогичные описанным выше в пп. 4 и 5.

Если система по какой-либо причине не свяжет EXE-файл с необходимыми ему DLL, на экране появится соответствующее сообщение, а адресное пространство процесса и объект “процесс” будут освобождены. При этом функция *CreateProcess* вернет значение FALSE.

После увязки EXE- и DLL-файлов с адресным пространством процесса начинает исполняться стартовый код EXE-файла. Подкачку страниц, буферизацию и кэширование выполняет система. Например, если код в EXE-файле переходит к команде, не загруженной в оперативную память, возникает ошибка. Обнаружив ее, система перекачивает нужную страницу кода из образа файла на страницу оперативной памяти. Затем отображает страницу оперативной памяти на должный участок адресного пространства процесса, позволяя потоку продолжить выполнение кода. Все эти операции скрыты от приложения и периодически повторяются при каждой попытке процесса обратиться к коду или данным, отсутствующим в оперативной памяти.

ОС позволяет проецировать на адресное пространство процесса и файл данных. Для этого нужно выполнить три операции:

1) создать или открыть объект ядра “файл”, идентифицирующий дисковый файл, который предполагается использовать как проецируемый в память, для этого используется функция *CreateFile*;

2) создать объект ядра “проекция файла” с помощью функции *CreateFileMapping*, чтобы сообщить системе размер файла и способ доступа к нему;

3) указать системе через функцию *MapViewOfFile*, как спроецировать в адресное пространство процесса объект “проекция файла” – целиком или частично.

Закончив работу с проецируемым в память файлом, следует выполнить тоже три операции:

1) сообщить системе об отмене проецирования на адресное пространство процесса объекта ядра “проекция файла”. Для этого предусмотрена функция *UnmapViewOfFile*;

2) закрыть объект “проекция файла”;

3) закрыть объект файл. Последние два действия осуществляются с помощью функции *CloseHandle*.

Остановимся теперь на возможности разделения данных между несколькими процессами, выполняемыми на одной машине. Создавать файл на диске и хранить там данные только с этой целью неудобно. Поэтому в Windows предусмотрена возможность проецирования файлов непосредственно на физическую память из страничного файла, а не из специально создаваемого дискового файла. Этот способ проще стандартного, основанного на создании дискового файла, проецируемого в память.

Создав объект “проекция файла” и спроецировав его представление на адресное пространство своего процесса, его можно использовать так же, как и любой другой регион памяти. Для того чтобы данные стали доступны другим процессам, нужно вызвать функцию *CreateFileMapping* и передать в параметре *pszName* строку с нулевым символом в конце. Тогда посторонние процессы, если им понадобится доступ к этому же файлу, смогут вызвать *CreateFileMapping* или OpenFileMapping и передать ей то же имя.

Когда необходимость в доступе к объекту “проекция файла” отпадет, процесс должен вызвать функцию CloseHandle. Как только все описатели объекта будут закрыты, система освободит память, переданную из страничного файла.

Программа MMFShare иллюстрирует, как происходит обмен данными между двумя и более процессами с помощью файлов, проецируемых в память. Для проведения эксперимента нужно запустить минимум две копии программы. Каждый экземпляр программы создаст свое диалоговое окно.

Чтобы переслать данные из одной копии программы в другую, нужно набрать какой-нибудь текст в поле Data, а затем щелкнуть кнопку Create mapping of Data. Программа вызовет функцию *CreateFileMapping*, чтобы создать объект “проекция файла” размером 4 Кбайт и присвоить ему имя *MMFSharedData* (ресурсы выделяются объекту из страничного файла). Увидев, что объект с таким именем существует, программа выдаст сообщение, что не может создать объект. А если такого объекта нет, спроецирует представление файла на адресное пространство процесса и скопирует данные из поля Data в проецируемый файл.

Далее программа прекратит проецировать представление файла, отключит кнопку Create mapping of Data и активизирует кнопку Close Mapping of Data. На этот момент проецируемый в память файл с именем *MMFSharedDATA* будет просто где-то находиться в системе и никакие процессы пока не проецируют представление на данные, содержащиеся в файле.

Если теперь перейти в другую копию программы и там щелкнуть кнопку Open mapping and get Data, то программа попытается найти объект “проекция файла” через функцию *OpenFileMapping*. Если ей не удастся найти объект с таким именем, она выдаст соответствующее сообщение. В ином случае она спроецирует представление объекта на адресное пространство своего процесса и скопирует данные из проецируемого файла в поле Data. Таким образом, данные пересланы из одного процесса в другой.

Кнопка Close Mapping Of Data служит для закрытия объекта “проекция файла”, что приводит к освобождению физической памяти, занимаемой им в страничном файле. Если же объект “проекция файла” не существует, никакой другой экземпляр программы MMFShare не сможет открыть этот объект и получить от него данные. Кроме того, если один экземпляр программы создал объект “проекция файла”, то остальным повторить его создание и тем самым перезаписать данные, содержащиеся в файле, уже не удастся.

Задание.

1) Изучите возможности использования виртуальной памяти для операций с большими массивами объектов или структур на примере программы VMAlloc.

2) Изучите возможности применения проецируемых файлов, достоинства и недостатки этой технологии Windows.

3) Запустите два и более экземпляров программы MMFShare и проведите эксперимент по обмену данными между процессами.

6. Изменение размера файла подкачки

Файл подкачки – это область жесткого диска, используемая Windows для хранения данных оперативной памяти. Он создает иллюзию, что система располагает большим объемом оперативной памяти, чем это есть на самом деле. Единой стратегии работы с файлом подкачки не существует. Многое определяется назначением и настройкой компьютера.

По умолчанию Windows удаляет файл подкачки после каждого сеанса работы и создает его в процессе загрузки операционной системы. Размер файла постоянно меняется по мере выполнения приложений и контролируется операционной системой. Обычно используется единственный файл подкачки, расположенный на том же диске, что и операционная система. Такой подход не является лучшим, и более того, практически всегда плох. В этом случае возникает несколько проблем.

1) Память может неожиданно оказаться исчерпанной из-за того, что приложение создало на жестком диске файл большого объема или операционная система без предупреждения увеличила потребности файла подкачки. Большой файл подкачки приводит к дефициту дискового пространства и к увеличению непроизводительных затрат на организацию страничного обмена.

2) Файл подкачки фрагментируется, что приводит не только к медленному считыванию жесткого диска, но и к дополнительным перемещениям считывающей головки диска, а в итоге – к существенному снижению производительности.

3) Файл подкачки фрагментируется сам по себе и очень быстро, причем так, что одна и та же область памяти может оказаться в разных местах жесткого диска. В этом случае даже отдельные приложения не могут получить доступ к памяти без нескольких обращений к диску.

4) Производительность системы падает и в том случае, если ОС установлена не на самом быстром из жестких дисков, имеющихся в компьютере.

Наличие двух жестких дисков может дать значительное преимущество при настройке файла подкачки. Для максимально эффективного использования файла подкачки нужно так его настроить, чтобы он располагался на жестком диске в виде достаточно протяженных фрагментов (это уменьшает количество перемещений считывающей головки, радикально влияющих на производительность). Кроме того, файл подкачки необходимо периодически удалять, чтобы избежать его фрагментации.

Для установки размера файла подкачки нужно выполнить следующую последовательность действий. Щелкнуть правой клавишей мыши на значке Мой компьютер и выбрать в контекстном меню строку Свойства. На экране появится окно Свойства системы.

Кроме того, следует задать режим использования памяти. Для пользовательского компьютера – оптимизировать работу программ, для сервера – системного кэша.

Определение размера файла подкачки до сих пор вызывает многочисленные дискуссии. Основное правило заключается в том, что при небольшом объеме оперативной памяти файл подкачки должен быть достаточно большим. При большом объеме оперативной памяти (512 Мбайт и более) файл подкачки можно уменьшить. Существует возможность вообще ликвидировать файл подкачки, выполнив определенную настройку реестра. Однако в этом случае объем оперативной памяти должен быть достаточно большим, лишь немногие системы обладают подобными ресурсами.

По рекомендации, приведенной в ряде источников, можно установить исходный размер файла подкачки, равный размеру физической памяти, а максимальный размер не более двух размеров физической памяти.

Задание.

1) Определите объем оперативной памяти компьютера и рекомендуемый объем файла подкачки.

2) Как изложено выше, проведите дефрагментацию жесткого диска, на который предполагается поместить файл подкачки, установите его желаемое значение и перезагрузите компьютер.

3) Оцените полученный эффект, полученный в результате изменения объема и размещения файла подкачки.

Лабораторная работа. Операционная система Windows. Файловая безопасность

1. Назначение разрешений для файлов

Устанавливая пользователям определенные разрешения для файлов и каталогов (папок), администраторы системы могут защищать конфиденциальную информацию от несанкционированного доступа. Каждый пользователь имеет определенный набор разрешений на доступ к конкретному объекту файловой системы. Администратор может назна-

чить себя владельцем любого объекта файловой системы (обратная передача владения невозможна).

Разрешения пользователя на доступ к объектам файловой системы работают по принципу аддитивности. Это значит, что действующие разрешения в отношении конкретного файла или каталога образуются из всех прямых и косвенных разрешений, назначенных пользователю для данного объекта с помощью логической функции ИЛИ.

Для назначения пользователю или группе разрешения на доступ к некоторому файлу нужно выполнить следующие действия:

1) Щелкнуть по файлу правой кнопкой мыши, выбрать в контекстном меню команду Свойства и в открывшемся окне перейти на вкладку Безопасность.

2) В верхней части окна (Группы и пользователи) отображен список пользователей и групп, которым уже предоставлены разрешения для данного файла. Для добавления или удаления пользователя нужно нажать кнопку Добавить или Удалить.

3) В новом окне ввести имя нужного объекта (в нашем случае пользователя), проверить это имя, нажав кнопку Проверить имена, нажать кнопку добавить, а затем ОК, чтобы вернуться на вкладку Безопасность.

4) Теперь можно назначить или запретить стандартные разрешения для файлов. Если нужно назначить разрешения более детально по видам возможных действий, то нужно нажать кнопку Дополнительно, после чего в новом окне нажать кнопку Изменить и в появившемся окне выбрать нужные разрешения. Дважды нажать кнопку ОК (возвращаясь по окнам), а затем Применить и ОК.

2. Назначение разрешений для папок

Последовательность действий в этом случае практически не отличается от назначения разрешений для файлов. Для назначения разрешения на доступ к каталогу (папке) пользователю или группе нужно:

1) Выбрать каталог и нажать правую кнопку мыши. В контекстном меню выбрать команду Свойства, после чего в появившемся окне перейти на вкладку Безопасность.

2) Как и в случае разрешений для файлов, предоставление разрешений пользователям и группам выполняется с помощью кнопок Добавить и Удалить. Список стандартных разрешений в группе Разрешения для каталога несколько отличается от набора разрешений для файла. Кроме того, нужно помнить, что разрешения для каталогов распространяются на находящиеся в них файлы.

3) В группе Разрешения можно назначить или запретить стандартные разрешения для каталогов, аналогичные разрешениям для файлов. Чтобы задать особые разрешения нужно нажать кнопку Дополнительно.

4) В новом окне можно установить два варианта наследования разрешений. По умолчанию каталоги наследуют разрешения пользователя на родительский каталог. Если наследование разрешений необходимо запретить, следует снять флажок Наследовать от родительского объекта применимые к дочерним объектам разрешения, добавляя их к явно заданным в этом окне.

5) После снятия значка появляется окно, которое позволяет выбрать желаемый вид наследования.

6) Чтобы выполнить более тонкую настройку доступа к папке, нужно нажать кнопку Изменить и в появившемся окне установить требуемые разрешения.

3. Передача права владения

Пользователь может назначить себя владельцем какого-либо объекта файловой системы, если у него есть необходимые права. Для передачи владения или просмотра текущего владельца файла (папки), нужно открыть окно Свойства, перейти на вкладку Безопасность и нажать кнопку Дополнительно.

В появившемся окне перейти на вкладку Владелец. Текущий владелец виден в поле Текущий владелец этого элемента. В списке Изменить владельца на перечислены пользователи, имеющие право получения владения данным объектом. Выбрать нужного пользователя и нажать кнопку Применить и затем ОК.

4. Точки соединения NTFS

Точки соединения (аналог монтирования в UNIX) позволяют отображать целевую папку (диск) в пустую папку, находящуюся в пространстве имен файловой системы NTFS 5.0 локального компьютера. Целевой папкой может служить любой допустимый путь Windows 2000. Точки соединений (поддерживаются только в NTFS 5.0) прозрачны для приложений, это означает, что приложение или пользователь, осуществляющий доступ к локальной папке NTFS, автоматически перенаправляется к другой папке.

Точки соединения NTFS отличаются от точек соединения распределенной файловой системы DFS. Последние отображают общий ресурс сети, управляемый DFS. Таким ресурсом может быть любой допустимый общий ресурс сети. Однако оба средства служат для создания общего пространства имен хранения информации. Рассмотрите самостоятельно сравнение точек соединения DFS и NTFS и их свойства.

Для работы с точками соединения на уровне томов можно использовать стандартные средства системы – утилиту Mountvol.exe и оснастку Управление дисками. Для монтирования папок нужна утилита Linkd (из Windows 2000 Resource Kit).

С помощью утилиты Mountvol можно выполнить следующие действия:

1) отобразить корневую папку локального тома в некоторую целевую папку NTFS, т.е. подключить или монтировать том;

2) вывести на экран информацию о целевой папке точки соединения NTFS, использованной при подключении тома;

3) просмотреть список доступных для использования томов файловой системы;

4) уничтожить точки подключения томов.

Параметры утилиты Mountvol можно получить, введя в командной строке ее имя.

Лабораторная работа Основы работы в ОС Linux

Linux – многозадачная, многопользовательская сетевая операционная система, поддерживающая стандарты открытых систем и протоколы сети Интернет. Все компоненты системы, включая исходные тексты, распространяются с лицензией на свободное копирование и установку для неограниченного числа пользователей.

В создании системы принимают участие тысячи разработчиков по всему миру. Авторские права на каждую из программ сохраняются за ее создателями, сами программы свободно распространяются на основе Публичной лицензии GNU, предусматривающей свободное копирование и модификацию программ с обязательным указанием авторов первичных кодов и без права на коммерческое использование.

Преимущества Linux:

- легальное и бесплатное получение современной ОС;
- высокое быстродействие;
- надежная и устойчивая работа;
- защита от вирусов;
- предъявление очень скромных требований к ресурсам компьютера и, в то же время, очень эффективное использование возможностей, предоставляемых современными (в том числе многопроцессорными) системами;
- эффективная реализация режима истинной многозадачности;
- простое интегрирование компьютера в локальную сеть;
- выполнение представленных в виде загрузочных модулей прикладных программ, написанных для других ОС;

Работа с консолью

Консолью называется совокупность основных устройств ввода информации в компьютер (клавиатура и мышь) и вывода информации (монитор). Linux работает с несколькими так называемыми виртуальными консолями, из которых в каждый момент времени только одна может быть связана с реальной (физической) консолью (то есть, является активной). В дальнейшем слово ``консоль'' будет обозначать именно виртуальную консоль.

Консоли нумеруются целыми положительными числами. Их общее количество может изменяться в зависимости от настроек ОС и достигать нескольких десятков, хотя в стандартной настройке оно не превышает десяти. Несколько первых консолей – текстовые, далее идут графические (в стандартной настройке – одна).

При работе в графической консоли д**ля того чтобы сделать активной другую консоль с некоторым номером n, требуется нажать клавиши [Ctrl]+[Alt]+[Fn]**, то есть, например [Ctrl]+[Alt]+[F2], при переходе в консоль с номером 2.

Если в результате запуска корректно настроенной системы или при переходе из другой консоли активной становится текстовая консоль, следует ввести имя пользователя, которое вводится в позицию текстового курсора после приглашения localhost login: (здесь слово localhost означает имя компьютера в локальной сети по умолчанию; это имя может быть другим).

В следующей строке будет выведено приглашение Password:

При наборе пароля символы на экране не появляются, и курсор не перемещается.

При неверном вводе будет предложено повторить всю процедуру идентификации пользователя. В случае успешного прохождения идентификации появится приглашение командного процессора вида:

[user_name@localhost dir_name]$

(user_name – имя пользователя, а dir_name – название текущего каталога).
Дальнейшая работа состоит в запуске различных программ в режиме командной строки.

Общая структура файловой системы

Файловая система Linux основана на модели иерархического дерева каталогов. Однако в Linux отсутствует понятие логического устройства (диска), все каталоги являются подкаталогами единого дерева и начинаются с так называемого корневого каталога. Корневой каталог системы обозначается символом /, подкаталог корневого каталога с именем каталог1 обозначается /каталог1, подкаталог этого каталога /каталог1/каталог2, а файл, находящийся в каталоге /каталог1 обозначается /каталог1/файл1 (никакой разницы в обозначении файлов и каталогов не существует).

В именах файлов и каталогов могут встречаться практически любые символы (причем прописные и строчные буквы различаются), однако далеко не все программы могут работать с именами, в которых используется, например, символ звездочка (*). Необходимо проявлять особую осторожность и при работе с файлами, содержащими в именах символы с кодами, превышающими 127 (например, буквы национальных алфавитов).

Имена физических устройств компьютера выглядят как имена файлов в подкаталоге первого уровня /dev (и действительно являются файлами особого вида). Разделы жесткого диска с интерфейсом IDE (EIDE) имеют имена вида /dev/hdXY, где X это одна из букв a, b, c, d, обозначающие соответственно с 1 по 4 физический диск (от Primary Master до Secondary Slave), а Y число, обозначающее номер раздела на диске (разделы нумеруются в том порядке, в котором они перечислены в таблице разделов диска). Например, единственный раздел второго (Slave) диска, присоединенного к первичному (Primary) контроллеру, обозначается /dev/hdb1.

Присоединение других файловых систем (монтирование устройств)

Для того чтобы сделать доступными файловые системы, размещенные на физических устройствах, требуется совместить корневые каталоги этих устройств с некоторыми подкаталогами файловой системы. Эта операция называется монтированием устройства. Жесткие диски обычно монтируются при запуске системы (например, основной раздел установки Linux, монтируется в корневой каталог /), однако их можно монтировать и отдельно.

Другие устройства, например, дисководы для гибких дисков A: (устройство /dev/fd0) и B: (/dev/fd1) и дисковод CD-ROM (/dev/cdrom), монтируются по мере необходимости.

По окончании работы с устройствами они размонтируются, и размещенные на них файлы становятся недоступными. Для жестких дисков это действие обычно выполняется автоматически при отключении системы. Иначе обстоит дело с гибкими дисками и CD-ROM: соответствующие устройства необходимо размонтировать перед тем как удалять из компьютера носители.

Нельзя удалить компакт-диск из неразмонтированного привода (устройство не реагирует на кнопку извлечения).

Команда монтирования устройства имеет вид:

mount -t тип_файловой_системы устройство каталог

Параметр устройство задает физическое устройство в вышеописанном формате, параметр каталог точку монтирования (каталог общего дерева каталогов, с которым будет совмещен корневой каталог устройства, каталог должен существовать и быть пуст), параметр тип_файловой_системы ключевое слово, обозначающее стандарт, в котором записываются на устройстве файлы и каталоги. Чаще всего Вам могут встретиться следующие типы файловых систем:

ext2
файловая система, используемая в разделах жесткого диска GNU/Linux;
iso9660
файловая система, используемая на большинстве CD-ROM;
vfat
файловая система, используемая (с вариациями) в MS-DOS и Windows 9x для жестких дисков и дискет (включает в себя системы, в терминологии Windows называющиеся FAT и FAT32).

Таким образом, чтобы смонтировать привод CD-ROM в каталог /MyCD, требуется ввести команду

mount -t iso9660 /dev/cdrom /MyCD

Для двух наиболее стандартных ситуаций можно применять упрощенные форматы команд монтирования.

Для того чтобы смонтировать дисковод A: на каталог /mnt/floppy с автоматическим определением типа файловой системы дискете, введите

mount /dev/fd0

Для того чтобы смонтировать дисковод CD-ROM на каталог /mnt/cdrom с автоматическим определением типа файловой системы на диске, введите

mount /dev/cdrom

Для того чтобы размонтировать устройство, введите команду

umount устройство

например, umount /dev/cdrom

Домашний каталог

Каждый пользователь системы имеет свой личный каталог, который называется начальным каталогом или домашним. При входе в систему после ввода имени и пароля вы окажетесь в вашем личном каталоге. Имя начального каталога пользователя хранится в системной переменной ``$HOME".

Каталог может содержать и другие каталоги (подкаталоги). Поэтому естественным способом представления организации каталогов и файлов является дерево каталогов. В процессе обхода этого дерева, начиная от корня, можно найти любой нужный файл.

Каталог является специальным файлом, который содержит имена файлов, размещенных в этом каталоге, а также имена подкаталогов и ссылки на них. По отношению к подкаталогам, текущий каталог является родительским (parent directory). Для обозначения каталогов используются следующие системные имена:

. (точка) - текущий каталог

.. (две точки) - родительский каталог, расположенный на один уровень ближе к корню.

Структура каталогов Linux

Только что установленная система Linux имеет дерево каталогов следующей структуры:

/
корневой каталог.
/bin
основные системные программы.
/boot
загрузочные файлы ядра ОС.
/dev
описания устройств компьютера.
/etc
конфигурационные файлы системы и подкаталоги с конфигурационными файлами прикладных программ.
/home
подкаталоги (домашние каталоги) пользователей.
/lib
динамические библиотеки.
/lost+found
информация об удаленных файлах, при некоторых условиях помогающая восстановить данные.
/mnt
подкаталоги стандартные точки монтирования сменных устройств, таких как дисковод CD-ROM.
/root
домашний каталог суперпользователя.
/sbin
системные программы.
/usr
прикладные программы и библиотеки.
/var
рабочие каталоги программ.

Следует помещать свои документы и другие файлы только в свой домашний каталог. В нем можно создавать любые нужные подкаталоги, руководствуясь соображениями удобства представления информации. Не следует вносить изменения в другие каталоги, за исключением случаев установки новых прикладных программ (в этом случае надо следовать инструкции к дистрибутиву устанавливаемой программы) или редактирования конфигурационных файлов.

Многопользовательские возможности

Многопользовательские возможности системы Linux имеют принципиальное значение: любые действия пользователя возможны только после идентификации пользователя с какой-либо из регистрационных записей. Linux предоставляет пользователю возможности, определяемые его правами на работу с файлами и каталогами. Считается, что каждый файл принадлежит определенному пользователю и определенной группе пользователей. С каждым файлом также связана таблица прав, указывающая, какие действия (чтение, запись, исполнение программы, открытие каталога и некоторые другие) может предпринимать владелец файла, член группы, владеющей файлом, и произвольный пользователь.

Кроме регистрационных записей обычных пользователей существует также регистрационная запись суперпользователя с зарезервированным именем root. Суперпользователь может выполнять любые допустимые действия с любым файлом, независимо от того, кто является его владельцем и какова таблица прав этого файла. Кроме того, суперпользователь может изменять информацию о владельце любого файла и его таблицу прав. А также суперпользователь обладает значительно более широкими правами, не связанными с файловыми операциями.

Создание групп, состоящих из пользователей, совместно работающих над определенными проектами, и присвоение каталогам, содержащим файлы этих проектов, соответствующего атрибута группы-владельца, в сочетании с настройкой таблицы прав, определяющих, какие именно операции могут выполнять члены этой группы, дает мощный инструмент для совместной работы над защищенными от постороннего вмешательства данными.

Добавление пользователя

Для того чтобы добавить регистрационную запись обычного пользователя, требуется сделать следующее.

Перейдите в текстовую консоль или режим эмуляции терминала и войдите в систему как суперпользователь. Введите в командной строке команду

adduser имя_пользователя

где имя_пользователя должно состоять из латинских букв и цифр и начинаться с буквы. В системе не должно существовать регистрационной записи с таким же именем. В случае успешного завершения операции создания новой регистрационной записи никаких сообщений не выдается. Создание регистрационный записи происходит в режиме, когда вход пользователя невозможен, поэтому для вновь созданного пользователя необходимо установить пароль.

Команда adduser создает регистрационную запись для нового пользователя (для чего вносит изменения в ряд конфигурационных файлов системы) и создает для него так называемый домашний каталог

/home/имя_пользователя

содержащий некоторые необходимые файлы и подкаталоги. Этот каталог и файлы в нем принадлежат пользователю, и он имеет полный набор прав для работы с этими файлами и подкаталогами, а также для всех вновь создаваемых им в этом каталоге объектов.

Установка или смена пароля

Для введения пароля нового пользователя или изменения пароля существующего введите в командной строке команду

passwd имя_пользователя

В ответ на приглашение

New UNIX password:

введите пароль (допускаются любые символы, прописные и строчные буквы различаются). На экране не отображаются никакие символы, и курсор по мере набора символов не перемещается.

Если введенный пароль слишком простой (например, короче шести символов), об этом будет выдано предупреждение, начинающееся со слов BAD PASSWORD (далее следует описание недостатков пароля).

После ввода пароля будет предложено подтвердить введенный пароль. В ответ на приглашение

Retype new UNIX password:

введите пароль еще раз.

В случае успеха (если дважды был введен одинаковый пароль) выдается сообщение

passwd: all authentification tokens updated successfully

и программа завершает работу. Если пароли, введенные в первом и во втором случае, не совпадают, выдается сообщение

Sorry, passwords do not match

и Вам вновь предлагается ввести пароль.

Удаление пользователя

Иногда необходимо удалить регистрационную запись пользователя (например, зарегистрированного по ошибке). Для того чтобы удалить регистрационную запись пользователя имя_пользователя, необходимо работать в текстовой консоли с правами суперпользователя. Введите в командной строке команду

userdel имя_пользователя

Для того чтобы удалить регистрационную запись, а также принадлежащие пользователю файлы, введите в командной строке команду

userdel -r имя_пользователя

Эта команда удаляет каталог /home/имя_пользователя и все его содержимое. Файлы данного пользователя, расположенные в других местах, не удаляются автоматически, их необходимо находить и удалять вручную.

Удаление файлов, принадлежавших ранее удаленному пользователю, а также любой другой доступ к ним (например, просмотр) возможен только с правами суперпользователя.

Удаление пользователя невозможно, если он в данный момент зарегистрирован в системе или работает какой-либо процесс, запущенный от его имени.

Командный процессор

При работе пользователя в текстовой консоли или терминале первичный диалог осуществляет системная программа, называемая командным процессором. В простейшем случае эта программа выдает приглашение и ожидает ввода команды. Команда представляет собой имя исполняемого файла (двоичного или текстового, так называемого скрипта, написанного на одном из специальных командных языков) или имя внутренней команды самого процессора. По окончании работы эти программы, как правило, возвращают управление командному процессору, однако могут также возвращать управление, не завершая работы (при этом командный процессор выдает приглашение, а ранее запущенная программа продолжает работу в фоновом режиме).

Linux может использовать разные командные процессоры, в том числе несколько одновременно, однако Вам, вероятнее всего, встретится наиболее распространенный из них Bash. Shell - интерпретатор командного языка

Запуск команды в фоновом режиме (символ &)

Некоторые команды shell занимают много времени при выполнении. Эти команды можно запустить в фоновом режиме с использованием &, освобождая тем самым терминал для других задач. Общий формат для запуска команд в фоновом режиме следующий:

command &

Примечание. Интерактивные команды shell (например, read) нельзя запускать в фоновом режиме.

Последовательное выполнение команд (символ ;)

В одной командной строке вы можете указать несколько команд. Эти команды должны быть разделены символом ; (точка с запятой) или символом & (амперсанд):

```
command1; command2; command3
```

Отмена специального значения (метасимвол ``слэш'')

Символ \(``слэш'') позволяет вам отменить специальное значение следующего за ним символа. Например, у вас есть файл trail, который содержит следующий текст:

```
The all * game
was held in Summit.
```

Чтобы найти символ звездочка (*) в файле, воспользуйтесь командой grep:

```
$ grep \* trail
The all * game
$
```

команда grep найдет символ * в тексте и отобразит строку, в которой она появилась. Без символа \, символ звездочка будет интерпретироваться shell как метасимвол.

Отмена специального значения (метасимвол кавычки)

Отменить специальное значение символа вы также можете с помощью метасимвола кавычки. Одиночные кавычки ('...') отменяют специальное значение всех символов за исключением самих одиночных кавычек. Двойные кавычки (``...'') отменяют специальное значение всех символов, за исключением символов двойные кавычки, $ и `(слабое ударение). Использование кавычек удобно для цифровых специальных символов.

Например, ваш файл trail содержит строку:

```
He really wondered why? Why???
```

Чтобы найти строку, содержащую три вопросительных знака, воспользуйтесь командой grep:

```
$ grep '???' trail
He really wondered why? Why???
$
```

Перенаправление ввода и вывода

В системе Linux некоторые команды ожидают ввод только с клавиатуры (стандартный ввод) и большинство команд отображают свой вывод на экране терминала (стандартный вывод). Однако система Linux позволяет вам перенаправлять ввод и вывод в файлы и программы, т.е. вы можете сказать shell:

взять ввод из файла, а не с клавиатуры;

послать вывод в файл, а не на терминал;

использовать программу как исходные данные для другой программы.

Чтобы перенаправить ввод, укажите в командной строке после знака "меньше чем" (<) имя файла:

```
command < имя\_файла
```

Чтобы перенаправить вывод, укажите в командной строке после знака "больше чем" (>) имя файла:

```
command > имя\_файла
```

Примечание. Если вы перенаправите вывод в уже существующий файл, то вывод вашей команды заменит содержимое существующего файла.

Перед тем, как перенаправить вывод команды в конкретный файл убедитесь, что этот файл не существует. shell не предупреждает, что выполняет перезапись существующего файла.

Добавить вывод в существующий файл

Чтобы добавить вывод в существующий файл и не разрушить его, вы можете воспользоваться символом »:

command >> имя_файла

Команда cat печатает содержимое файлов, имена которых являются ее аргументами, в стандартный вывод.

Комбинирование фонового режима и перенаправления вывода

Когда команда запущена в фоновом режиме, то вывод ее печатается на экране терминала. И если вы используете терминал в то же время для выполнения других задач, то вывод фоновой задачи будет прерывать вашу работу. Однако, если перенаправить вывод в файл, то вы сможете спокойно работать.

Получить состояние запущенного процесса

Команда ps дает вам состояние всех процессов, запущенных на данный момент. Например, вы можете использовать команду ps, чтобы просмотреть состояние всех процессов, которые запущены в фоновом режиме, применив символ &.

В следующем подпункте обсуждается вопрос, как применить номер PID (идентификатор процесса), чтобы остановить выполнение команды. PID является уникальным номером, который система Linux назначает каждому активному процессу.

Обратите внимание, что система распечатала номер PID для команды grep так же, как и для всех других запущенных процессов: для самой команды ps и команды bash, которая была запущена во время вашей регистрации.

Можно приостановить и вновь запустить программу, если в вашей системе предусмотрена функция управления заданиями. Команда jobs выдает список текущих фоновых процессов, запущенных или приостановленных. Команда jobs дополнительно к PID распечатывает идентификатор задания (JID) и имя задания. Чтобы вновь запустить приостановленное задание, либо возобновить фоновый процесс в оперативном режиме, вам необходимо знать JID. JID распечатывается на экране каждый раз, когда вы вводите команду запуска или останова процесса. Если вы введете:

jobs

то на экране появится следующая информация:

[JID] - Stopped (signal) <имя задания>

или

[JID] + Running <имя задания>

Завершение активных процессов

Команда kill завершает активные процессы в фоновом режиме и команда stop приостанавливает временно процессы. Общий формат этих команд:

kill PID

или

stop JID

Обратите внимание, что вы не можете завершать фоновые задания нажатием клавиш BREAK или DEL.

После того как система выдаст ответ на запрос, на экране появится подсказка $, означающая, что процесс завершен. Если система не найдет указанный PID, то появится сообщение об ошибке:

No such process

Чтобы приостановить оперативный процесс (если активна функция управления заданиями), введите ^Z. На экране появится следующее сообщение:

<JID> Stopped(user) <имя задания>

Запуск остановленного задания

Если функция управления заданиями активна, то вы можете вновь запустить приостановленный процесс. Чтобы вновь запустить процесс, остановленный командой stop, вы сначала должны определить JID с помощью команды jobs. Затем вы можете использовать JID со следующими командами:

fg <JID>

- возобновляет приостановленное задание или переводит задание из фонового режима в оперативный;

bg <JID>

- вновь запускает приостановленное задание в фоновом режиме.

. Если вы хотите, чтобы после вашего выхода из системы процесс в фоновом режиме продолжал выполняться, то вам необходимо использовать команду nohup. Команда nohup имеет следующий формат:

nohup command &

Задания:

1. Переключитесь на новую консоль, и войдите пользователем user, пароль user.
2. Смонтируйте и размонтируйте привод CD-ROM, и дисковод в каталог /CD и /Floppy соответственно
3. Перейдите в каталог /dev (cd) и выведите на экран список содержащихся в нем файлов (ls), затем содержимое любого файла на экран (cat). При необходимости используйте команды (less и more)
4. Создайте пользователя, установите ему пароль, удалите пользователя.
5. Перейдите в папку /home и найдите в файле Lab1 строку с символом *
6. Найдите в файле Lab2 строку содержащую «???»
7. Содержимое файла Lab2 добавляем после последней строки в файл Lab1 (cat, перенаправление вывода)
8. Найти все появления слова "test" в файле schedule. Запустите команду grep в фоновом режиме и перенаправьте вывод в файл testfile. Затем выведите на экран содержимое testfile
9. Запустите команду grep (поиск во всех файлах в вашем текущем каталоге строки "word" перенаправив вывод в файл) в фоновом режиме, и покажите как она отображается в списке процессов
10. Завершите выполнение процесса grep принудительно.
11. Повторите задание 9 с использованием nohup

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Самостоятельная работа по дисциплине включает изучение учебной литературы, подготовка к выполнению лабораторных заданий и отчетов по их выполнению, а также подготовку к тестированию по теоретическим разделам и непосредственно к зачету.

Задания к лабораторным работам выдаются заранее, как правило, на первом занятие текущего семестра, и для их успешного их выполнения необходимо предварительное освоение теоретического материала. Для этого наряду с конспектами можно воспользоваться учебно-методическим обеспечением для самостоятельной работы, указанным в рабочей программе.

Для подготовки к выполнению лабораторных работ и повторения, усвоения (изучения пропущенного) теоретического материала студентам рекомендуется самостоятельно организовать по месту проживания дополнительное рабочее место, оборудованное персональным компьютером, подключённым к сети Интернет, и установленным программным обеспечением, необходимым для разработки программ и указанном в рабочей программе.

В отчете по выполнению заданий должны содержаться следующие сведения: формулировка задания, ход его выполнения, скриншоты.

Итоговый контроль проводится на основании перечней вопросов, представленных в рабочей программе. Подготовка к итоговому заключается в изучении и тщательной проработке студентом конспектов лекций и лабораторных занятий в соответствии с перечнем вопросов, представленном в рабочей программе дисциплины.

Ответы на поставленные вопросы студент дает после предварительной подготовки. Преподаватель имеет право задать дополнительные вопросы, если ответ дан неполный или затруднительно однозначно оценить ответ.