МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ по дисциплине
«Операционные системы»
для студентов специальности 10.05.01 «Компьютерная безопасность»
(специализация «Информационно-аналитическая и техническая экспертиза компьютерных систем»)

Ставрополь
2026

Содержание

Лабораторная работа №1. Интерпретатор командной строки ОС MS Window (4ч)

Лабораторная работа №2. Интерпретатор командной строки ОС MS Windows (4ч)

Лабораторная работа №3. Оболочка командной строки Windows PowerShell (4ч)

Лабораторная работа № 4. Исследование и мониторинг производительности ос windows с помощью системного монитора (4ч).

Лабораторная работа № 5. Симметричная мультипроцессорная обработка (4ч).

Лабораторная работа № 6. Файловые системы операционной системы LINUX (4ч).

Лабораторная работа 7. Контроль использования ресурсов ОС Linux (4ч)

Лабораторная работа № 8 Обработка строк (работа с текстовыми данными) (4ч)

Лабораторная работа №1. Интерпретатор командной строки ОС MS Windows (4 часа)

Часть 1. Внешние и внутренние команды

Цель работы – знакомство с возможностями интерпретатора командной строки и командами MS Windows

1.1. Эволюция инструментов для автоматизации работы в ОС Microsoft Windows

В настоящее время графический интерфейс Windows стал настолько привычным, что многие пользователи и начинающие администраторы даже не задумываются об альтернативных способах управления данной ОС, связанных с командной строкой (command line) и различными сценариями (scripts), о тех преимуществах, которые дают эти инструменты с точки зрения автоматизации работы. Подобная ситуация обусловлена тем, что исторически командная строка всегда была слабым местом операционной системы Windows (по сравнению с Unix-системами).

При этом, однако, неправильно было бы думать, что командная строка или сценарии нужны только администраторам. Ведь ежедневные рутинные задачи пользователей (связанные, например, с копированием или архивированием файлов, подключением или отключением сетевых ресурсов и т.п.), которые обычно выполняются с помощью графического интерфейса проводника Windows, можно полностью самостоятельно автоматизировать, написав нехитрый командный файл, состоящий всего из нескольких строчек! Более того, для человека, не знающего основные команды Windows и такие базовые возможности ОС, как перенаправление ввода/вывода и конвейеризация команд, некоторые простейшие задачи могут показаться нетривиальными. Попробуйте, например, пользуясь только графическими средствами, сформировать файл, содержащий имена файлов из всех подкаталогов какого-либо каталога! А ведь для этого достаточно выполнить единственную команду DIR (с определенными ключами) и перенаправить вывод этой команды в нужный текстовый файл.

Каким же нам хотелось бы видеть инструмент для автоматизации работы в ОС? Желательно, чтобы было реализовано следующее:

– работа в разных версиях ОС без установки какого-либо дополнительного программного обеспечения;

– интеграция с командной строкой (непосредственное выполнение вводимых с клавиатуры команд);

– согласованный и непротиворечивый синтаксис команд и утилит;

– наличие подробной встроенной справки по командам с примерами использования.

В ОС Windows дело обстоит сложнее. На сегодняшний день одного "идеального" средства автоматизации, удовлетворяющего сразу всем перечисленным выше требованиям, в Windows нет; в последних версиях ОС поддерживаются несколько стандартных инструментов автоматизации, сильно отличающихся друг от друга: оболочка командной строки cmd.exe, среда выполнения сценариев Windows Script Host и оболочка Microsoft PowerShell. Поэтому администратору или пользователю Windows приходится выбирать, каким именно подходом воспользоваться для решения определенной задачи, а для этого желательно иметь четкое представление о сильных и слабых сторонах данных средств автоматизации. Рассмотрим достоинства и недостатки каждого из них.

1.1.1. Оболочка (интерпретатор) командной строки command.com/cmd.exe

Во всех версиях ОС Windows поддерживается интерактивная оболочка командной строки (command shell) и определенный набор утилит командной строки (количество и состав этих утилит зависит от версии ОС). Механизм работы оболочек командной строки в разных системах одинаков: в ответ на приглашение ("подсказку", prompt), выдаваемое находящейся в ожидании оболочкой, пользователь вводит некоторую команду (функциональность этой

команды может быть реализована либо самой оболочкой, либо определенной внешней утилитой), оболочка выполняет ее, при необходимости выводя на экран какую-либо информацию, после чего снова выводит приглашение и ожидает ввода следующей команды.

Оболочка представляет собой построчный интерпретатор простого языка сентенциального (директивного) программирования, в качестве операторов которого могут использоваться исполняемые программы.

Наряду с интерактивным режимом работы оболочки, как правило, поддерживают и пакетный режим, в котором система последовательно выполняет команды, записанные в текстовом файле-сценарии. Оболочка Windows не является исключением, с точки зрения программирования язык командных файлов Windows может быть охарактеризован следующим образом:

– реализация сентенциальной (директивной) парадигмы программирования;
– выполнение в режиме построчной интерпретации;
– наличие управляющих конструкций;
– поддержка нескольких видов циклов (в том числе специальных циклов для обработки текстовых файлов);
– наличие оператора присваивания (установки значения переменной);
– возможность использования внешних программ (команд) операционной системы в качестве операторов и обработки их кодов возврата;
– наличие нетипизированных переменных, которые декларируются первым упоминанием (значения переменных могут интерпретироваться как числа и использоваться в выражениях целочисленной арифметики).

Начиная с версии Windows NT, оболочка командной строки представляется интерпретатором Cmd.exe.

Итак, учитывая сказанное выше, можно сделать вывод: оболочка командной строки cmd.exe и командные файлы – наиболее универсальные и простые в изучении средства автоматизации работы в Windows, доступные во всех версиях операционной системы.

1.1.2. Поддержка языков сценариев. Сервер сценариев Windows Script Host

Следующим шагом в развитии средств и технологий автоматизации в ОС Windows стало появление сервера сценариев Windows Script Host (WSH). Этот инструмент разработан для всех версий Windows и позволяет непосредственно в ОС выполнять сценарии на полноценных языках сценариев (по умолчанию, VBScript и JScript), которые до этого были доступны только внутри HTML-страниц и работали в контексте безопасности веб-браузера (в силу этого подобные сценарии, например, могли не иметь доступа к файловой системе локального компьютера).

По сравнению с командными файлами интерпретатора cmd.exe сценарии WSH имеют несколько **преимуществ**.

Во-первых, VBScript и JScript – это полноценные алгоритмические языки, имеющие встроенные функции и методы для обработки символьных строк, выполнения математических операций, обработки исключительных ситуаций и т.д.; кроме того, для написания сценариев WSH может использоваться любой другой язык сценариев (например, широко распространенный в Unix-системах Perl), для которого установлен соответствующий модуль поддержки.

Во-вторых, WSH поддерживает несколько собственных объектов, свойства и методы которых позволяют решать некоторые часто возникающие повседневные задачи администратора операционной системы: работа с сетевыми ресурсами, переменными среды, системным реестром, ярлыками и специальными папками Windows, запуск и управление работой других приложений.

В-третьих, из сценариев WSH можно обращаться к службам любых приложений-серверов автоматизации (например, программ из пакета MS Office), которые регистрируют в ОС свои объекты.

Наконец, сценарии WSH позволяют работать с объектами информационной модели Windows Management Instrumentation (WMI), обеспечивающей программный интерфейс управления всеми компонентами операционной модели, а также с объектами службы каталогов Active Directory Service Interface.

Следует также отметить, что технология WSH поддерживается в Windows уже довольно давно, в Интернете (в том числе на сайте Microsoft) можно найти множество готовых сценариев.

1.1.3. Командная оболочка Microsoft PowerShell

С одной стороны функциональности и гибкости языка оболочки cmd.exe явно недостаточно, а с другой стороны сценарии WSH, работающие с объектными моделями ADSI и WMI, слишком сложны для пользователей среднего уровня и начинающих администраторов.

Перед разработчиками новой оболочки, получившей название Windows PowerShell, стояли следующие основные цели:

- применение командной строки в качестве основного интерфейса администрирования;
- реализация модели ObjectFlow (элементом обмена информации является объект);
- переработка существующих команд, утилит и оболочки;
- интеграция командной строки, объектов COM, WMI и .NET;
- работа с произвольными источниками данных в командной строке по принципу файловой системы.

Самая важная идея, заложенная в PowerShell, состоит в том, что в командной строке вывод результатов команды представляет собой не текст (в смысле последовательности символов), а объект (данные вместе со свойственными им методами). В силу этого работать в PowerShell становится проще, чем в традиционных оболочках, так как не нужно выполнять никаких манипуляций по выделению нужной информации из символьного потока.

Отметим, что PowerShell одновременно является и оболочкой командной строки (пользователь работает в интерактивном режиме) и средой выполнения сценариев, которые пишутся на специальном языке PowerShell.

В целом, оболочка PowerShell намного удобнее и мощнее своих предшественников (cmd.exe и WSH), а основным недостатком, сдерживающим распространение нового инструмента, является тот факт, что PowerShell работает не во всех версиях ОС Windows. Оболочкой можно пользоваться только на версиях не ниже Windows XP Service Pack 2 с установленным пакетом .NET Framework 2.0.

1.2 Оболочка командной строки Windows. Интерпретатор Cmd.exe

Рассматриваются внутренние команды, поддерживаемые интерпретатором Cmd.exe, и наиболее часто используемые внешние команды (утилиты командной строки). Описываются механизмы перенаправления ввода/вывода, конвейеризации и условного выполнения команд. Даются примеры команд для работы с файловой системой

В ОС Windows, как и в других ОС, интерактивные (набираемые с клавиатуры и сразу же выполняемые) команды выполняются с помощью так называемого командного интерпретатора, иначе называемого командным процессором или оболочкой командной строки (command shell). Начиная с версии Windows NT, в операционной системе реализован интерпретатор команд Cmd.exe, обладающий гораздо более широкими возможностями.

1.2.1 Запуск оболочки

В Windows NT/2000/XP (и выше) файл Cmd.exe, как и другие исполняемые файлы, соответствующие внешним командам ОС, находятся в каталоге %SystemRoot%\SYSTEM32 (значением переменной среды %SystemRoot% является системный каталог Windows, обычно

C:\Windows или C:\WinNT). Для запуска командного интерпретатора (открытия нового сеанса командной строки) можно выбрать пункт Выполнить… (Run) в меню Пуск (Start), ввести имя файла Cmd.exe и нажать кнопку OK. В результате откроется новое окно (см. рис. 1), в котором можно запускать команды и видеть результат их работы.

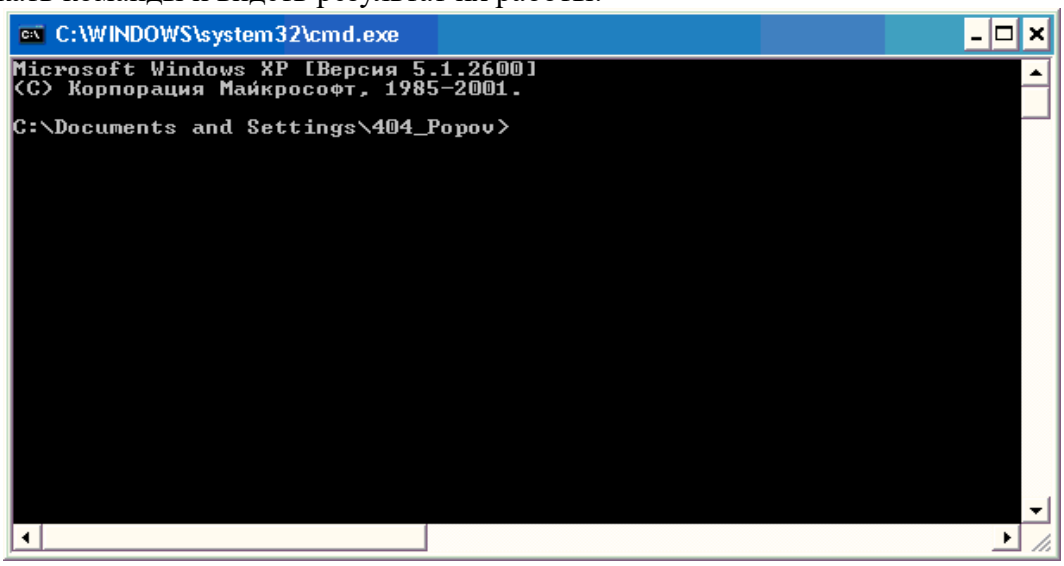


Рис. 1 - Командное окно интерпретатора Cmd.exe в Windows

1.2.2 Настройка параметров командного окна интерпретатора

У утилиты командной строки, которая поставляется в виде стандартного приложения ОС Windows, имеется свой набор опций и параметров настройки. Один из способов просмотра этих опций – использование пункта Свойства управляющего меню окна (нажать правой кнопкой мыши на заголовок окна). В окне свойств (см. рис. 2) будут доступны четыре вкладки с опциями: общие, шрифт, расположение и цвета.

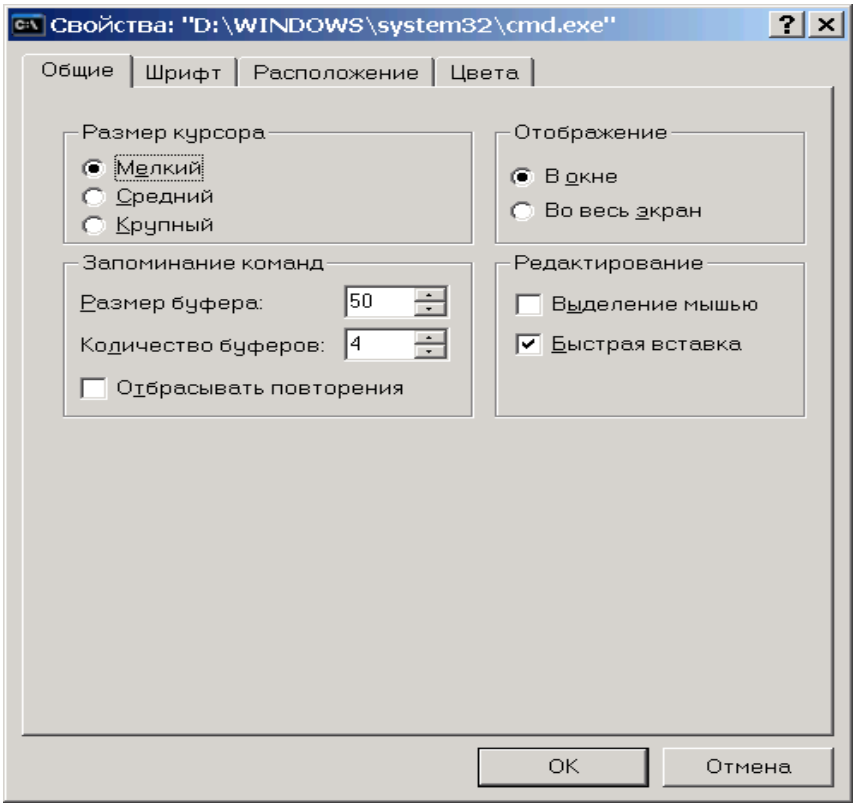


Рис. 2 – окно настройки свойств интерпретатора

1.2.3. Внутренние и внешние команды. Структура команд

Некоторые команды распознаются и выполняются непосредственно самим командным интерпретатором – такие команды называются **внутренними** (например, COPY или DIR). Другие команды ОС представляют собой отдельные программы, расположенные по умолчанию в том же каталоге, что и Cmd.exe, которые Windows загружает и выполняет аналогично другим программам. Такие команды называются **внешними** (например, MORE или XCOPY).

Рассмотрим структуру самой командной строки и принцип работы с ней. Для того, чтобы выполнить команду, после приглашения командной строки (например, C:\>) следует ввести имя этой команды (регистр не важен), ее параметры и ключи (если они необходимы) и нажать клавишу <Enter>. Например:

C:\>COPY C:\myfile.txt A:\ /V

Имя команды здесь – COPY, параметры – C:\myfile.txt и A:\, а ключом является /V. Отметим, что в некоторых командах ключи могут начинаться не с символа /, а с символа – (минус), например, -V.

Многие команды Windows имеют большое количество дополнительных параметров и ключей, запомнить которые зачастую бывает трудно. Большинство команд снабжено встроенной справкой, в которой кратко описываются назначение и синтаксис данной команды. Получить доступ к такой справке можно путем ввода команды с ключом /?. Например, если выполнить команду ATTRIB /?, то в окне MS-DOS мы увидим следующий текст:

```
Отображение и изменение атрибутов файлов.
ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] [[диск:][путь]имя_файла] [/S]
  +   Установка атрибута.
  -   Снятие атрибута.
  R   Атрибут "Только чтение".
  A   Атрибут "Архивный".
  S   Атрибут "Системный".
  H   Атрибут "Скрытый".
  /S  Обработка файлов во всех вложенных папках указанного пути.
```

Для некоторых команд текст встроенной справки может быть довольно большим и не умещаться на одном экране. В этом случае помощь можно выводить последовательно по одному экрану с помощью команды MORE и символа конвейеризации |, например:

XCOPY /? | MORE

В этом случае после заполнения очередного экрана вывод помощи будет прерываться до нажатия любой клавиши. Кроме того, используя символы перенаправления вывода > и >>, можно текст, выводимый на экран, направить в текстовый файл для дальнейшего просмотра. Например, для вывода текста справки к команде XCOPY в текстовый файл xcopy.txt, используется следующая команда:

XCOPY /? > XCOPY.TXT

Замечание. Вместо имени файла можно указывать обозначения устройств компьютера. В Windows поддерживаются следующие **имена устройств**: PRN (принтер), CON (терминал: при вводе это клавиатура, при выводе - монитор), NUL (пустое устройство, все операции ввода/вывода для него игнорируются).

1.2.4. Перенаправление ввода/вывода и конвейеризация (композиция) команд

С помощью переназначения устройств ввода/вывода одна программа может направить свой вывод на вход другой или перехватить вывод другой программы, используя его в качестве своих входных данных. Таким образом, имеется возможность передавать информацию от процесса к процессу при минимальных программных издержках. Практически это означает, что для программ, которые используют стандартные входные и выходные устройства, ОС позволяет:

– выводить сообщения программ не на экран (стандартный выходной поток), а в файл или на принтер (перенаправление вывода);

– читать входные данные не с клавиатуры (стандартный входной поток), а из заранее подготовленного файла (перенаправление ввода);

– передавать сообщения, выводимые одной программой, в качестве входных данных для другой программы (конвейеризация или композиция команд).

Из командной строки эти возможности реализуются следующим образом. Для того, чтобы перенаправить текстовые сообщения, выводимые какой-либо командой, в текстовый файл, нужно использовать конструкцию

команда > имя_файла

Если при этом заданный для вывода файл уже существовал, то он перезаписывается, если не существовал – создается. Можно также не создавать файл заново, а *дописывать* информацию, выводимую командой, в конец существующего файла. Для этого команда перенаправления вывода должна быть задана так:

команда >> имя_файла

С помощью символа < можно прочитать входные данные для заданной команды не с клавиатуры, а из определенного (заранее подготовленного) файла:

команда < имя_файла

Приведем несколько примеров перенаправления ввода/вывода.

1. Вывод встроенной справки для команды COPY в файл copy.txt:

COPY /? > copy.txt

2. Добавление текста справки для команды XCOPY в файл copy.txt:

XCOPY /? >> copy.txt

3. Вывод текущей даты в файл date.txt (DATE /T – это команда для просмотра и изменения системной даты, T ключ для получения только даты без запроса нового значения):

DATE /T > date.txt

Если при выполнении определенной команды возникает ошибка, то сообщение об этом по умолчанию выводится на экран. В случае необходимости сообщения об ошибках (стандартный поток ошибок) можно перенаправить в текстовый файл с помощью конструкции

команда 2> имя_файла

В этом случае стандартный вывод будет производиться на экран. Также имеется возможность информационные сообщения и сообщения об ошибках выводить в один и тот же файл. Делается это следующим образом:

команда > имя_файла 2>&1

Например, в приведенной ниже команде стандартный выходной поток и стандартный поток ошибок перенаправляются в файл copy.txt:

XCOPY A:\1.txt C: > copy.txt 2>&1

Наконец, с помощью конструкции

команда1 | команда2

можно использовать сообщения, выводимые первой командой, в качестве входных данных для второй команды (конвейер команд).

Используя механизмы перенаправления ввода/вывода и конвейеризации, можно из командной строки посылать информацию на различные устройства и автоматизировать ответы на запросы, выдаваемые командами или программами, использующими стандартный ввод. Для решения таких задач служит команда

ECHO [сообщение]

которая выводит сообщение на экран. Пример использования этой команды.

1. Удаление всех файлов в текущем каталоге без предупреждения (автоматический положительный ответ на запрос об удалении):

ECHO y | DEL *.*

1.2.5. Команды MORE и SORT

Одной из наиболее часто использующихся команд, для работы с которой применяется перенаправление ввода/вывода и конвейеризация, является MORE. Эта команда считывает стандартный ввод из конвейера или перенаправленного файла и выводит информацию частями, размер каждой из которых не больше размера экрана. Используется MORE обычно для просмотра длинных файлов. Возможны три варианта синтаксиса этой команды:

MORE [диск:][путь]имя_файла

MORE < [диск:][путь]имя_файла

имя_команды | MORE

Параметр [диск:][путь]имя_файла определяет расположение и имя файла с просматриваемыми на экране данными. Параметр имя_команды задает команду, вывод которой отображается на экране (например, DIR или команда TYPE, использующаяся для вывода содержимого текстового файла на экран). Приведем два примера.

Для поэкранного просмотра текстового файла news.txt возможны следующие варианты команд:

MORE news.txt

MORE < news.txt

TYPE news.txt | MORE

Другой распространенной командой, использующей перенаправление ввода/вывода и конвейеризацию, является SORT. Эта команда работает как фильтр: она считывает символы в заданном столбце, упорядочивает их в возрастающем или убывающем порядке и выводит отсортированную информацию в файл, на экран или другое устройство. Возможны два варианта синтаксиса этой команды:

SORT [/R] [/+n] [[диск1:][путь1]файл1] [> [диск2:][путь2]файл2]

или

[команда |] SORT [/R] [/+n] [> [диск2:][путь2]файл2]

В первом случае параметр [диск1:][путь1]файл1 определяет имя файла, который нужно отсортировать. Во втором случае будут отсортированы выходные данные указанной команды. Если параметры файл1 или команда не заданы, то SORT будет считывать данные с устройства стандартного ввода.

Параметр [диск2:][путь2]файл2 задает файл, в который будет направляется сортированный вывод; если этот параметр не задан, то вывод будет направлен на устройство стандартного вывода.

По умолчанию сортировка выполняется в порядке возрастания. Ключ /R позволяет изменить порядок сортировки на обратный (от Z к A и затем от 9 до 0). Например, для поэкранного просмотра отсортированного в обратном порядке файла price.txt, нужно задать следующую команду:

SORT /R < price.txt |MORE

Ключ /+n задает сортировку в файле по символам n-го столбца. Например, /+10 означает, что сортировка должна осуществляться, начиная с 10-й позиции в каждой строке. По умолчанию файл сортируется по первому столбцу.

1.2.6. Условное выполнение и группировка команд

В командной строке Windows можно использовать специальные символы, которые позволяют вводить несколько команд одновременно и управлять работой команд в зависимости от результатов их выполнения. С помощью таких символов условной обработки можно

содержание небольшого пакетного файла записать в одной строке и выполнить полученную составную команду.

Используя символ амперсанда &, можно разделить несколько утилит в одной командной строке, при этом они будут выполняться друг за другом. Например, если набрать команду DIR & PAUSE & COPY /? и нажать клавишу <Enter>, то вначале на экран будет выведено содержимое текущего каталога, а после нажатия любой клавиши – встроенная справка команды COPY.

Условная обработка команд в Windows осуществляется с помощью символов && и || следующим образом. Двойной амперсанд && запускает команду, стоящую за ним в командной строке, только в том случае, если команда, стоящая перед амперсандами была выполнена успешно. Например, если в корневом каталоге диска C: есть файл plan.txt, то выполнение строки TYPE C:\plan.txt && DIR приведет к выводу на экран этого файла и содержимого текущего каталога. Если же файл C:\plan.txt не существует, то команда DIR выполняться не будет.

Два символа || осуществляют в командной строке обратное действие, т.е. запускают команду, стоящую за этими символами, только в том случае, если команда, идущая перед ними, не была успешно выполнена. Таким образом, если в предыдущем примере файл C:\plan.txt будет отсутствовать, то в результате выполнения строки TYPE C:\plan.txt || DIR на экран выведется содержимое текущего каталога.

Отметим, что условная обработка действует только на ближайшую команду, то есть в строке

TYPE C:\plan.txt && DIR & COPY /?

команда COPY /? запустится в любом случае, независимо от результата выполнения команды TYPE C:\plan.txt.

Несколько утилит можно сгруппировать в командной строке с помощью *круглых скобок*. Рассмотрим, например, две строки:

TYPE C:\plan.txt && DIR & COPY /?

TYPE C:\plan.txt && (DIR & COPY /?)

В первой из них символ условной обработки && действует только на команду DIR, во второй – одновременно на две команды: DIR и COPY.

1.3. Команды для работы с файловой системой

Рассмотрим некоторые наиболее часто используемые команды для работы с файловой системой. Отметим сначала несколько особенностей определения путей к файлам в Windows.

1.3.1. Пути к объектам файловой системы

Файловая система логически имеет древовидную структуру и имена файлов задаются в формате [диск:][путь\]имя_файла, то есть обязательным параметром является только имя файла. При этом, если путь начинается с символа "\", то маршрут вычисляется от корневого каталога, иначе – от текущего каталога. Например, имя C:123.txt задает файл 123.txt в текущем каталоге на диске C:, имя C:\123.txt – файл 123.txt в корневом каталоге на диске C:, имя ABC\123.txt – файл 123.txt в подкаталоге ABC текущего каталога.

Существуют особые обозначения для текущего каталога и родительского каталогов. Текущий каталог обозначается символом . (точка), его родительский каталог – символами .. (две точки). Например, если текущим каталогом является C:\WINDOWS, то путь к файлу autoexec.bat в корневом каталоге диска C: может быть записан в виде ..\autoexec.bat.

В именах файлов (но не дисков или каталогов) можно применять так называемые **групповые символы** или шаблоны: ? (вопросительный знак) и * (звездочка). Символ * в имени файла означает произвольное количество любых допустимых символов, символ ? – один произвольный символ или его отсутствие. Скажем, под шаблон text??1.txt подходят, например,

имена text121.txt и text11.txt, под шаблон text*.txt – имена text.txt, textab12.txt, а под шаблон text.* – все файлы с именем text и произвольным расширением.

Для того, чтобы использовать длинные имена файлов при работе с командной строкой, их нужно заключать в двойные кавычки. Например, чтобы запустить файл с именем 'Мое приложение.exe' из каталога 'Мои документы', нужно в командной строке набрать "C:\Мои документы\Мое приложение.exe" и нажать клавишу <Enter>.

1.3.2. Команда CD

Текущий каталог можно изменить с помощью команды CD [диск:][путь\]

Путь к требуемому каталогу указывается с учетом приведенных выше замечаний. Например, команда CD \ выполняет переход в корневой каталог текущего диска. Если запустить команду CD без параметров, то на экран будут выведены имена текущего диска и каталога.

1.3.3. Команда COPY

Одной из наиболее часто повторяющихся задач при работе на компьютере является копирование и перемещение файлов из одного места в другое. Для копирования одного или нескольких файлов используется команда COPY.

Синтаксис этой команды:

COPY [/A|/B] источник [/A|/B] [+ источник [/A|/B] [+ ...]]
[результат [/A|/B]] [/V][/Y|/–Y]

Краткое описание параметров и ключей команды COPY приведено в табл. 1.

Таблица 1. Параметры и ключи команды COPY

Параметр	**Описание**
источник	Имя копируемого файла или файлов
/A	Файл является текстовым файлом ASCII, то есть конец файла обозначается символом с кодом ASCII 26 (<Ctrl>+<Z>)
/B	Файл является двоичным. Этот ключ указывает на то, что интерпретатор команд должен при копировании считывать из источника число байт, заданное размером в каталоге копируемого файла
результат	Каталог для размещения результата копирования и/или имя создаваемого файла
/V	Проверка правильности копирования путем сравнения файлов после копирования
/Y	Отключение режима запроса подтверждения на замену файлов
/-Y	Включение режима запроса подтверждения на замену файлов

Примеры использования команды COPY.

1. Копирование файла abc.txt из текущего каталога в каталог D:\PROGRAM под тем же именем:

COPY abc.txt D:\PROGRAM

2. Копирование файла abc.txt из текущего каталога в каталог D:\PROGRAM под новым именем def.txt:

COPY abc.txt D:\PROGRAM\def.txt

3. Копирование всех файлов с расширением txt с диска A: в каталог 'Мои документы' на диске C:

COPY A:*.txt "C:\Мои документы"

Если не задать в команде целевой файл, то команда COPY создаст копию файла-источника с тем же именем, датой и временем создания, что и исходный файл, и поместит новую копию в текущий каталог на текущем диске. Например, для того, чтобы скопировать все файлы из корневого каталога диска A: в текущий каталог, достаточно выполнить такую команду:

COPY A:*.*

Пример 1 создания нового текстового файла и записи в него информации без использования текстового редактора.

Для решения задачи необходимо ввести команду COPY CON my.txt, которая будет копировать то, что набирается на клавиатуре в файл my.txt (если этот файл существовал, то он перезапишется, иначе – создастся). Для завершения ввода необходимо ввести символ конца файла, то есть нажать клавиши <Ctrl>+<Z>.

Команда COPY может также объединять (склеивать) несколько файлов в один. Для этого необходимо указать единственный результирующий файл и несколько исходных. Это достигается путем использования групповых знаков (? и *) или формата файл1 + файл2 + файл3. Например, для объединения файлов 1.txt и 2.txt в файл 3.txt можно задать следующую команду:

COPY 1.txt+2.txt 3.txt

Объединение всех файлов с расширением dat из текущего каталога в один файл all.dat может быть произведено так:

COPY /B *.dat all.dat

Ключ /B здесь используется для предотвращения усечения соединяемых файлов, так как при комбинировании файлов команда COPY по умолчанию считает файлами текстовыми.

Если имя целевого файла совпадает с именем одного из копируемых файлов (кроме первого), то исходное содержимое целевого файла теряется. Если имя целевого файла опущено, то в его качестве используется первый файл из списка. Например, команда COPY 1.txt+2.txt добавит к содержимому файла 1.txt содержимое файла 2.txt. Командой COPY можно воспользоваться и для присвоения какому-либо файлу **текущей даты и времени** без модификации его содержимого. Для этого нужно ввести команду

COPY /B 1.txt +,,

Здесь запятые указывают на пропуск параметра приемника, что и приводит к требуемому результату.

Команда COPY имеет и свои недостатки. Например, с ее помощью нельзя копировать скрытые и системные файлы, файлы нулевой длины, файлы из подкаталогов. Кроме того, если при копировании группы файлов COPY встретит файл, который в данный момент нельзя скопировать (например, он занят другим приложением), то процесс копирования полностью прервется, и остальные файлы не будут скопированы.

1.3.4. Команда XCOPY

Указанные при описании команды COPY проблемы можно решить с помощью команды XCOPY, которая предоставляет намного больше возможностей при копировании. Необходимо отметить, правда, что XCOPY может работать только с файлами и каталогами, но не с **устройствами**. Синтаксис команды:

XCOPY источник [результат] [ключи]

Команда XCOPY имеет множество ключей, далее приведены лишь некоторых из них. Ключ /D[:[дата]] позволяет копировать только файлы, измененные не ранее указанной даты. Если параметр дата не указан, то копирование будет производиться только если источник новее результата. Например, команда

XCOPY "C:\Мои документы*.*" "D:\BACKUP\Мои документы" /D

скопирует в каталог 'D:\BACKUP\Мои документы' только те файлы из каталога 'C:\Мои документы', которые были изменены со времени последнего подобного копирования или которых вообще не было в 'D:\BACKUP\Мои документы'.

Ключ /S позволяет копировать все непустые подкаталоги в каталоге-источнике. С помощью же ключа /E можно копировать вообще все подкаталоги, включая и пустые.

Если указан ключ /C, то копирование будет продолжаться даже в случае возникновения ошибок. Это бывает очень полезным при операциях копирования, производимых над группами файлов, например, при резервном копировании данных.

Ключ /I важен для случая, когда копируются несколько файлов, а файл назначения отсутствует. При задании этого ключа команда XCOPY считает, что файл назначения должен быть каталогом. Например, если задать ключ /I в команде копирования всех файлов с расширением txt из текущего каталога в несуществующий еще подкаталог TEXT,

```
XCOPY  *.txt TEXT /I
```

то подкаталог TEXT будет создан без дополнительных запросов.

Ключи /Q, /F и /L отвечают за режим отображения при копировании. При задании ключа /Q имена файлов при копировании не отображаются, ключа /F – отображаются полные пути источника и результата. Ключ /L обозначает, что отображаются только файлы, которые должны быть скопированы (при этом само копирование не производится).

С помощью ключа /H можно копировать скрытые и системные файлы, а с помощью ключа /R – заменять файлы с атрибутом "Только для чтения". Например, для копирования всех файлов из корневого каталога диска C: (включая системные и скрытые) в каталог SYS на диске D:, нужно ввести следующую команду:

```
XCOPY C:\*.* D:\SYS /H
```

Ключ /T позволяет применять XCOPY для копирования только структуры каталогов источника, без дублирования находящихся в этих каталогах файлов, причем пустые каталоги и подкаталоги не включаются. Для того, чтобы все же включить пустые каталоги и подкаталоги, нужно использовать комбинацию ключей /T /E.

Используя XCOPY можно при копировании обновлять только уже существующие файлы (новые файлы при этом не записываются). Для этого применяется ключ /U. Например, если в каталоге C:\2 находились файлы a.txt и b.txt, а в каталоге C:\1 – файлы a.txt, b.txt, c.txt и d.txt, то после выполнения команды

```
XCOPY C:\1 C:\2 /U
```

в каталоге C:\2 по-прежнему останутся лишь два файла a.txt и b.txt, содержимое которых будет заменено содержимым соответствующих файлов из каталога C:\1. Если с помощью XCOPY копировался файл с атрибутом "Только для чтения", то по умолчанию у файла-копии этот атрибут снимется. Для того, чтобы копировать не только данные, но и полностью атрибуты файла, необходимо использовать ключ /K.

Ключи /Y и /-Y определяют, нужно ли запрашивать подтверждение перед заменой файлов при копировании. /Y означает, что такой запрос нужен, /-Y – не нужен.

1.3.5. Команда DIR

Команда

```
DIR [диск:][путь][имя_файла] [ключи]
```

используется для вывода информации о содержимом дисков и каталогов. Параметр [диск:][путь] задает диск и каталог, содержимое которого нужно вывести на экран. Параметр [имя_файла] задает файл или группу файлов, которые нужно включить в список. Например, команда

```
DIR C:\*.bat
```

выведет на экран все файлы с расширением bat в корневом каталоге диска C:. Если задать эту команду без параметров, то выводится метка диска и его серийный номер, имена (в коротком и длинном вариантах) файлов и подкаталогов, находящихся в текущем каталоге, а также дата и время их последней модификации. После этого выводится число файлов в

каталоге, общий объем (в байтах), занимаемый файлами, и объем свободного пространства на диске. Например:

```
Том в устройстве C имеет метку PHYS1_PART2
 Серийный номер тома: 366D-6107
 Содержимое папки C:\aditor
.            <ПАПКА>      25.01.00  17:15 .
..           <ПАПКА>      25.01.00  17:15 ..
HILITE   DAT        1 082  18.09.98  18:55 hilite.dat
TEMPLT01 DAT           48  07.08.98   1:00 templt01.dat
TTABLE   DAT          357  07.08.98   1:00 ttable.dat
ADITOR   EXE      461 312  01.12.99  23:13 aditor.exe
README   TXT        3 974  25.01.00  17:26 readme.txt
ADITOR   HLP       24 594  08.10.98  23:12 aditor.hlp
ТЕКСТО~1 TXT            0  11.03.01   9:02 Текстовый файл.txt
       11 файлов       533 647 байт
        2 папок     143 261 696 байт свободно
```

С помощью ключей команды DIR можно задать различные режимы расположения, фильтрации и сортировки. Например, при использовании ключа /W перечень файлов выводится в широком формате с максимально возможным числом имен файлов или каталогов на каждой строке. Например:

```
Том в устройстве C имеет метку PHYS1_PART2
 Серийный номер тома: 366D-6107
 Содержимое папки C:\aditor
[.]           [..]        TEMPLT02.DAT  UNINST1.000     HILITE.DAT
TEMPLT01.DAT    UNINST0.000     TTABLE.DAT    ADITOR.EXE      README.TXT
ADITOR.HLP        ТЕКСТО~1.TXT
       11 файлов       533 647 байт
        2 папок     143 257 600 байт свободно
```

С помощью ключа /A[[:]атрибуты] можно вывести имена только тех каталогов и файлов, которые имеют заданные атрибуты (R – "Только чтение", A – "Архивный", S – "Системный", H – "Скрытый", префикс "–" имеет значение НЕ). Если ключ /A используется более чем с одним значением атрибута, будут выведены имена только тех файлов, у которых все атрибуты совпадают с заданными. Например, для вывода имен всех файлов в корневом каталоге диска C:, которые одновременно являются скрытыми и системными, можно задать команду

```
DIR C:\ /A:HS
```

а для вывода всех файлов, кроме скрытых — команду

```
DIR C:\ /A:-H
```

Отметим здесь, что атрибуту каталога соответствует буква D, и для того, чтобы, например, вывести список всех каталогов диска C:, нужно задать команду

```
DIR C: /A:D
```

Ключ /O[[:]сортировка] задает порядок сортировки содержимого каталога при выводе его командой DIR. Если этот ключ опущен, DIR печатает имена файлов и каталогов в том порядке, в котором они содержатся в каталоге. Если ключ /O задан, а параметр сортировка не указан, то DIR выводит имена в алфавитном порядке. В параметре сортировка можно использовать следующие значения: N – по имени (алфавитная), S – по размеру (начиная с меньших), E – по расширению (алфавитная), D – по дате (начиная с более старых), A — по дате загрузки (начиная с более старых), G – начать список с каталогов. Префикс "–" означает обратный порядок. Если задается более одного значения порядка сортировки, файлы сортируются по первому критерию, затем по второму и т.д.

Ключ /S означает вывод списка файлов из заданного каталога и его подкаталогов.

Ключ /B перечисляет только названия каталогов и имена файлов (в длинном формате) по одному на строку, включая расширение. При этом выводится только основная информация, без итоговой. Например:

templt02.dat
UNINST1.000
hilite.dat
templt01.dat
UNINST0.000
ttable.dat
aditor.exe
readme.txt
aditor.hlp
Текстовый файл.txt

1.3.6. Команды MKDIR и RMDIR

Для создания нового каталога и удаления уже существующего пустого каталога используются команды MKDIR [диск:]путь и RMDIR [диск:]путь [ключи] соответственно (или их короткие аналоги MD и RD). Например:

```
MKDIR "C:\Примеры"
RMDIR "C:\Примеры"
```

Команда MKDIR не может быть выполнена, если каталог или файл с заданным именем уже существует. Команда RMDIR не будет выполнена, если удаляемый каталог не пустой.

1.3.7. Команда DEL

Удалить один или несколько файлов можно с помощью команды

DEL [диск:][путь]имя_файла [ключи]

Для удаления сразу нескольких файлов используются групповые знаки ? и *. Ключ /S позволяет удалить указанные файлы из всех подкаталогов, ключ /F – принудительно удалить файлы, доступные только для чтения, ключ /A[[:]атрибуты] – отбирать файлы для удаления по атрибутам (аналогично ключу /A[[:]атрибуты] в команде DIR).

1.3.8. Команда REN

Переименовать файлы и каталоги можно с помощью команды RENAME (REN). Синтаксис этой команды имеет следующий вид:

REN [диск:][путь][каталог1|файл1] [каталог2|файл2]

Здесь параметр каталог1|файл1 определяет название каталога/файла, которое нужно изменить, а каталог2|файл2 задает новое название каталога/файла. В любом параметре команды REN можно использовать групповые символы ? и *. При этом представленные шаблонами символы в параметре файл2 будут идентичны соответствующим символам в параметре файл1. Например, чтобы изменить у всех файлов с расширением txt в текущей директории расширение на doc, нужно ввести такую команду:

REN *.txt *.doc

Если файл с именем файл2 уже существует, то команда REN прекратит выполнение, и произойдет вывод сообщения, что файл уже существует или занят. Кроме того, в команде REN нельзя указать другой диск или каталог для создания результирующих каталога и файла. Для этой цели нужно использовать команду MOVE, предназначенную для переименования и перемещения файлов и каталогов.

1.3.9. Команда MOVE

Синтаксис команды для перемещения одного или более файлов имеет вид:

MOVE [/Y|/–Y] [диск:][путь]имя_файла1[,...] результирующий_файл

Синтаксис команды для переименования папки имеет вид:

MOVE [/Y|/–Y] [диск:][путь]каталог1 каталог2

Здесь параметр результирующий_файл задает новое размещение файла и может включать имя диска, двоеточие, имя каталога, либо их сочетание. Если перемещается только один файл, допускается указать его новое имя. Это позволяет сразу переместить и переименовать файл. Например,

MOVE "C:\Мои документы\список.txt" D:\list.txt

Если указан ключ /-Y, то при создании каталогов и замене файлов будет выдаваться запрос на подтверждение. Ключ /Y отменяет выдачу такого запроса.

2. МЕТОДИКА ВЫПОЛНЕНИЯ

1. Ознакомиться с теоретическими сведениями.
2. Запустить интерпретатор командной строки
3. Увеличить размер окна интерпретатора и задать цвет фона и цвет шрифта (рекомендуется синий фон и белый шрифт).
4. Создать список фамилий студентов группы, используя пример 1. Отсортировать список в алфавитном порядке и сохранить его в новом файле.

Замечание 1. При создании текстового файла интерпретатор командной строки использует кодировку **кириллица (DOS)**. Поэтому рекомендуется переназначать вывод в файл с расширением .**txt,** а для просмотра содержимого файла использовать Internet Explorer, указав вид кодировки кириллица (DOS). Пример вывода содержимого текстового файла приведен на рис. 3.

Замечание 2. Интерпретатор хранит историю введенных команд в буфере (размером 50 строк). Для просмотра содержимого буфера используйте клавиши клавиатуры СТРЕЛКА ВВЕРХ и СТРЕЛКА ВНИЗ. Полученную команду можно отредактировать и выполнить снова.

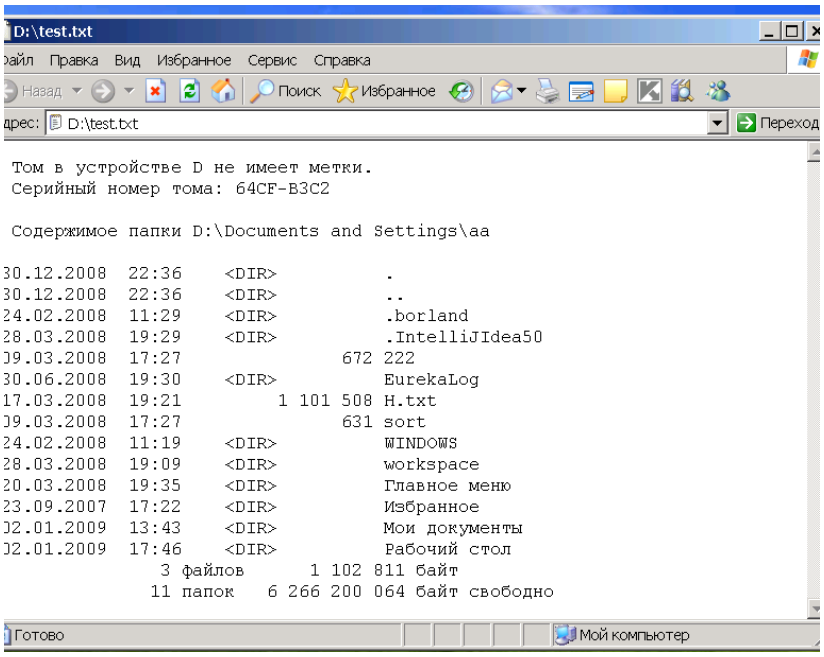


Рис.3 – вывод содержимого текстового файла, полученного с помощью команды DIR, в IE в кодировке кириллица (DOS)

5. Создать текстовый файл, содержащий справочные сведения по командам DIR, COPY и XCOPY.
6. Вывести содержимое указанного в табл.3 каталога по указанному формату на экран и в файл.

Таблица 3. Варианты заданий

Номера вариантов	Имя каталога	Что выводить	Сортировать по	Атрибуты файлов и каталогов
1	%Windows%	Только файлы	По размеру	Системный
2	%Windows%	Файлы и подкаталоги	По дате	Скрытый
3	%Windows%	Только подкаталоги	Именам	Только чтение
4	%Windows% и все подкаталоги	Только файлы bmp	По размеру	Только чтение
5	%Windows% и все подкаталоги	Только файлы jpg	Именам	Любые
6	%Windows%	Только файлы	По дате	Только чтение
7	%Windows%	Только подкаталоги	Именам	Системный
8	%Windows%	Только файлы jpg	По размеру	Любые

7. Скопировать все имеющиеся в каталоге Windows растровые графические файлы в каталог WinGrafika на диске C:. Если диск C: недоступен, использовать любой другой доступный диск.

8. Скопировать все имеющиеся в каталоге Windows **исполняемые файлы** в каталог WinEx на диске C:. Если диск C: недоступен, использовать любой другой доступный диск.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Достоинства и недостатки интерфейса командной строки.
2. Инструменты командной строки для автоматизации работы в ОС Microsoft Windows.
3. Настраиваемые свойства интерпретатора.
4. Различие между внутренними и внешними командами. Примеры внешних и внутренних команд.
5. Структура команды интерпретатора.
6. Получение информации о конкретной команде.
7. Групповые символы (шаблоны) и их использование.
8. Перенаправление ввода/вывода и конвейеризация команд.
9. Условное выполнение и группировка команд.
10. Назначение символов &, &&, || и () .
11. Команды для работы с файловой системой – названия и возможности.
12. Достоинства и недостатки команд COPY и XCOPY.
13. Назначение команды ECHO и примеры ее использования.
14. Команда DIR и ее возможности.
15. В какой кодировке интерпретатор выводит информацию и как получить читаемую твердую копию?

Лабораторная работа №2. Интерпретатор командной строки ОС MS Windows (4 часа)

Часть 2. Язык интерпретатора и командные файлы

Цель работы – знакомство с языком интерпретатора командной строки ОС MS Windows и командными файлами

1.1 Язык интерпретатора Cmd.exe. Командные файлы

Язык оболочки командной строки (shell language) в Windows реализован в виде командных (или пакетных) файлов. **Командный файл** в Windows – это обычный текстовый файл с расширением bat или cmd, в котором записаны допустимые команды ОС (как внешние, так и внутренние), а также некоторые дополнительные инструкции и ключевые слова, придающие командным файлам некоторое сходство с программами, написанными на языке программирования. Например, если записать в файл deltmp.bat следующие команды:

```
C:\
CD %TEMP%
DEL /F *.tmp
```

и запустить его на выполнение (аналогично исполняемым файлам с расширением com или exe), то мы удалим все файлы во временной директории Windows. Таким образом, исполнение командного файла приводит к тому же результату, что и последовательный ввод записанных в нем команд. При этом не проводится никакой предварительной компиляции или проверки синтаксиса кода; если встречается строка с ошибочной командой, то она игнорируется. Очевидно, что если приходится часто выполнять одни и те же действия, то использование командных файлов может сэкономить много времени.

1.1.1. Вывод сообщений и дублирование команд

По умолчанию команды пакетного файла перед исполнением выводятся на экран, что выглядит не очень эстетично. С помощью команды ECHO OFF можно отключить дублирование команд, идущих после нее (сама команда ECHO OFF при этом все же дублируется). Например,

```
REM Следующие две команды будут дублироваться на экране …
:: эта строка – такой же комментарий, как и предыдущая
DIR C:\
ECHO OFF
:: А остальные уже не будут
DIR D:\
```

Для восстановления режима дублирования используется команда ECHO ON. Кроме этого, можно отключить дублирование любой отдельной строки в командном файле, написав в начале этой строки символ @, например:

```
ECHO ON
:: Команда DIR C:\ дублируется на экране
DIR C:\
:: А команда DIR D:\ — нет
@DIR D:\
```

Таким образом, если поставить в самое начало файла команду

```
@ECHO OFF
```

то это решит все проблемы с дублированием команд.

В пакетном файле можно выводить на экран строки с сообщениями. Делается это с помощью команды

```
ECHO сообщение
```

Например,

```
@ECHO OFF
ECHO Привет!
```

Команда ECHO. (точка должна следовать непосредственно за словом "ECHO") выводит на экран **пустую строку**. Например,

```
@ECHO OFF
ECHO Привет!
ECHO.

ECHO Пока!
```

Часто бывает удобно для просмотра сообщений, выводимых из пакетного файла, предварительно полностью очистить экран командой CLS.

Используя механизм *перенаправления ввода/вывода* (символы > и >>), можно направить сообщения, выводимые командой ECHO, в определенный текстовый файл. Например:

```
@ECHO OFF
ECHO Привет! > hi.txt
ECHO Пока! >> hi.txt
```

С помощью такого метода можно, скажем, заполнять файлы-протоколы с отчетом о произведенных действиях. Например:

```
@ECHO OFF
REM Попытка копирования
XCOPY C:\PROGRAMS D:\PROGRAMS /s
:: Добавление сообщения в файл report.txt в случае удачного завершения копирования
IF NOT ERRORLEVEL 1 ECHO Успешное копирование >> report.txt
```

1.1.2 Использование параметров командной строки

При запуске пакетных файлов в командной строке можно указывать произвольное число параметров, значения которых можно использовать внутри файла. Это позволяет, например, применять один и тот же командный файл для выполнения команд с различными параметрами.

Для доступа из командного файла к параметрам командной строки применяются символы %0, %1, …, %9 или %*. При этом вместо %0 подставляется имя выполняемого пакетного файла, вместо %1, %2, …, %9 — значения первых девяти параметров командной строки соответственно, а вместо %* — все аргументы. Если в командной строке при вызове пакетного файла задано меньше девяти параметров, то "лишние" переменные из %1 – %9 замещаются пустыми строками. Рассмотрим следующий пример. Пусть имеется командный файл copier.bat следующего содержания:

```
@ECHO OFF
CLS
ECHO Файл %0 копирует каталог %1 в %2
XCOPY %1 %2 /S
```

Если запустить его из командной строки с двумя параметрами, например

```
copier.bat C:\Programs D:\Backup
```

то на экран выведется сообщение

```
Файл copier.bat копирует каталог C:\Programs в D:\Backup
```

и произойдет копирование каталога C:\Programs со всеми его подкаталогами в D:\Backup.

При необходимости можно использовать более девяти параметров командной строки. Это достигается с помощью команды SHIFT, которая изменяет значения замещаемых параметров с %0 по %9, копируя каждый параметр в предыдущий, то есть значение %1 копируется в %0, значение %2 – в %1 и т.д. Замещаемому параметру %9 присваивается значение параметра, следующего в командной строке за старым значением %9. Если же такой параметр не задан, то новое значение %9 — пустая строка.

Пример 1. Пусть командный файл my.bat вызван из командной строки следующим образом:

my.bat p1 p2 p3

Тогда %0=my.bat, %1=p1, %2=p2, %3=p3, параметры %4 – %9 являются пустыми строками. После выполнения команды SHIFT значения замещаемых параметров изменятся следующим образом: %0=p1, %1=p2, %2=p3, параметры %3 – %9 – пустые строки.

При включении расширенной обработки команд SHIFT поддерживает ключ /n, задающий начало сдвига параметров с номера n, где n может быть числом от 0 до 9.

Например, в следующей команде SHIFT /2

параметр %2 заменяется на %3, %3 на %4 и т.д., а параметры %0 и %1 остаются без изменений.

Команда, обратная SHIFT (обратный сдвиг), отсутствует. После выполнения SHIFT уже нельзя восстановить параметр (%0), который был первым перед сдвигом. Если в командной строке задано больше десяти параметров, то команду SHIFT можно использовать несколько раз.

В командных файлах имеются некоторые возможности синтаксического анализа заменяемых параметров. Для параметра с номером n (%n) допустимы синтаксические конструкции (операторы), представленные в табл. 1.

Таблица 1. Операторы для заменяемых параметров

Операторы	Описание
%~Fn	Переменная %n расширяется до полного имени файла
%~Dn	Из переменной %n выделяется только имя диска
%~Pn	Из переменной %n выделяется только путь к файлу
%~Nn	Из переменной %n выделяется только имя файла
%~Xn	Из переменной %n выделяется расширение имени файла
%~Tn	Возвращается дата и время создания (модификации) файла
%~Zn	Возвращается размер файла в байтах
%~$PATH:n	Проводится поиск по каталогам, заданным в переменной среды PATH, и переменная %n заменяется на полное имя первого найденного файла. Если переменная PATH не определена или в результате поиска не найден ни один файл, эта конструкция заменяется на пустую строку. Естественно, здесь переменную PATH можно заменить на любое другое допустимое значение

Данные синтаксические конструкции можно объединять друг с другом, например:

%~DPn — из переменной %n выделяется имя диска и путь,

%~NXn — из переменной %n выделяется имя файла и расширение.

Пример 2. Пусть мы находимся в каталоге C:\TEXT и запускаем пакетный файл с параметром Рассказ.doc (%1=Рассказ.doc). Размер файла 2150 байт, дата создания 12.12.2009, время -12:55. Тогда применение операторов, описанных в табл. 1, к параметру %1 даст следующие результаты:

%~F1=C:\TEXT\Рассказ.doc
%~D1=C:
%~P1=\TEXT\
%~N1=Рассказ
%~X1=.doc
%~DP1=C:\TEXT\
%~NX1=Рассказ.doc
%~T1=12.12.2009 12:55
%~Z1=2150

1.1.3. Работа с переменными среды

Внутри командных файлов можно использовать так называемые **переменными среды** (или переменными окружения), каждая из которых хранится в оперативной памяти, имеет свое уникальное имя, а ее значением является **строка**. Стандартные переменные среды автоматически инициализируются в процессе загрузки операционной системы. Такими переменными являются:

WINDIR, которая определяет расположение каталога Windows,

TEMP, которая определяет путь к каталогу для хранения временных файлов Windows

PATH, в которой хранится системный путь (путь поиска), то есть список каталогов, в которых система должна искать выполняемые файлы или файлы совместного доступа (например, динамические библиотеки).

Кроме того, в командных файлах с помощью команды SET можно объявлять собственные переменные среды.

1.1.3.1. Получение значения переменной

Для получения значения определенной переменной среды нужно заключить имя этой переменной в символы %. Например:

```
@ECHO OFF
CLS
:: Создание переменной MyVar
SET MyVar=Привет
:: Изменение переменной
SET MyVar=%MyVar%
ECHO Значение переменной MyVar: %MyVar%
:: Удаление переменной MyVar
SET MyVar=
ECHO Значение переменной WinDir: %WinDir%
```

При запуске такого командного файла на экран выведется строка

Значение переменной MyVar: Привет!

Значение переменной WinDir: C:\WINDOWS

1.1.4. Преобразования переменных как строк

С переменными среды в командных файлах можно производить некоторые манипуляции. *Во-первых*, над ними можно производить операцию конкатенации (соединения). Для этого нужно в команде SET просто написать рядом значения соединяемых переменных. Например,

```
SET A=Раз
SET B=Два
SET C=%A%%B%
```

После выполнения в файле этих команд значением переменной C будет являться строка 'РазДва'. Не следует для конкатенации использовать знак +, так как он будет воспринят просто в качестве символа. Например, после запуска файл следующего содержания

```
SET A=Раз
SET B=Два
SET C=A+B
ECHO Переменная C=%C%
SET D=%A%+%B%
ECHO Переменная D=%D%
```

на экран выведутся две строки:

Переменная C=A+B

Переменная D=Раз+Два

Во-вторых, из переменной среды можно **выделять подстроки** с помощью конструкции %имя_переменной:~n1,n2%, где число n1 определяет смещение (количество пропускаемых

символов) от начала (если n1 положительно) или от конца (если n1 отрицательно) соответствующей переменной среды, а число n2 – количество выделяемых символов (если n2 положительно) или количество последних символов в переменной, которые не войдут в выделяемую подстроку (если n2 отрицательно). Если указан только один отрицательный параметр -n, то будут извлечены последние n символов. Например, если в переменной хранится строка "21.12.2017" (символьное представление текущей даты при определенных региональных настройках), то после выполнения следующих команд

```
SET dd1=%DATE:~0,2%
SET dd2=%DATE:~0,-8%
SET mm=%DATE:~-7,2%
SET yyyy=%DATE:~-4%
```

новые переменные будут иметь такие значения: %dd1%=21, %dd2%=21, %mm%=12, %yyyy%=2017.

В-третьих, можно выполнять процедуру замены подстрок с помощью конструкции %имя_переменной:s1=s2% (в результате будет возвращена строка, в которой каждое вхождение подстроки s1 в соответствующей переменной среды заменено на s2). Например, после выполнения команд

```
SET a=123456
SET b=%a:23=99%
```

в переменной b будет храниться строка "199456". Если параметр s2 не указан, то подстрока s1 будет удалена из выводимой строки, т.е. после выполнения команды

```
SET a=123456
SET b=%a:23=%
```

в переменной b будет храниться строка "1456".

1.1.5. Операции с переменными как с числами

При включенной расширенной обработке команд имеется **возможность рассматривать значения переменных среды как числа** и производить с ними арифметические вычисления (используются ТОЛЬКО целые числа). Для этого используется команда SET с ключом /A. Ниже приведен пример пакетного файла add.bat, складывающего два числа, заданных в качестве параметров командной строки, и выводящего полученную сумму на экран:

```
@ECHO OFF
:: В переменной M будет храниться сумма
SET /A M=%1+%2
ECHO Сумма %1 и %2 равна %M%
:: Удалим переменную M
SET M=
```

В команде SET с ключом /A могут использоваться операции – (вычитание), * (умножение), / (деление нацело), % (остаток от деления). При использовании знака % в качестве знака операции в **командных файлах** он должен быть записан ДВАЖДЫ.

Рекомендуется при инициализации числовых переменных использовать ключ /A

```
SET /A col=0
```

1.1.6 Ввод значения переменной с клавиатуры

Ввод значения переменной при выполнении командного файла выполняется командой SET с ключом /P. Например, для ввода значения переменной M следует использовать команду

```
SET /P  M=[введите M]
```

Текст подсказки [введите M] будет выведен на экран.

1.1.7. Локальные изменения переменных

Все изменения, производимые с помощью команды SET над переменными среды в командном файле, сохраняются и после завершения работы этого файла, но действуют только внутри текущего командного окна. Также имеется возможность локализовать изменения переменных среды внутри пакетного файла, то есть автоматически восстанавливать значения всех переменных в том виде, в каком они были **до начала запуска** этого файла. Для этого используются две команды: SETLOCAL и ENDLOCAL. Команда SETLOCAL определяет начало области локальных установок переменных среды. Другими словами, изменения среды, внесенные после выполнения SETLOCAL, будут являться локальными относительно текущего пакетного файла. Каждая команда SETLOCAL должна иметь соответствующую команду ENDLOCAL для восстановления прежних значений переменных среды. Изменения среды, внесенные после выполнения команды ENDLOCAL, уже не являются локальными относительно текущего пакетного файла; их прежние значения не будут восстановлены по завершении выполнения этого файла.

1.1.8. Связывание времени выполнения для переменных

При работе с составными выражениями (группы команд, заключенных в круглые скобки) нужно учитывать, что переменные среды в командных файлах используются в режиме раннего связывания. С точки зрения логики выполнения командного файла это может привести к ошибкам. Например, рассмотрим командный файл 1.bat со следующим содержимым:

```
SET a=1
ECHO a=%a%
SET a=2
ECHO a=%a%
```

и командный файл 2.bat:

```
SET a=1
ECHO a=%a%
(SET a=2
ECHO a=%a% )
```

Казалось бы, результат выполнения этих двух файлов должен быть одинаковым: на экран выведутся две строки: "a=1" и "a=2". На самом же деле таким образом сработает только файл 1.bat, а файл 2.bat два раза выведет строку "a=1".

Данную ошибку можно обойти, если для получения значения переменной вместо знаков процента (%) использовать восклицательный знак (!) и предварительно включить режим связывания времени выполнения командой SETLOCAL ENABLEDELAYEDEXPANSION. Таким образом, для корректной работы файл 2.bat должен иметь следующий вид:

```
SETLOCAL ENABLEDELAYEDEXPANSION
SET a=1
ECHO a=%a%
(SET a=2
ECHO a=!a!)
```

ВНИМАНИЕ! Приведенный материал необходим для правильной работы команды цикла FOR и будет использован в командных файлах!

1.1.9. Приостановка выполнения командных файлов

Для того, чтобы вручную **прервать выполнение** запущенного bat-файла, нужно нажать клавиши <Ctrl>+<C> или <Ctrl>+<Break>. Однако часто бывает необходимо программно приостановить выполнение командного файла в определенной строке с выдачей запроса на нажатие любой клавиши. Это делается с помощью команды PAUSE. Перед запуском этой

команды полезно с помощью команды ECHO информировать пользователя о действиях, которые он должен произвести. Например:

```
ECHO Вставьте дискету в дисковод A: и нажмите любую клавишу
PAUSE
```

Команду PAUSE обязательно нужно использовать при выполнении потенциально опасных действий (удаление файлов, форматирование дисков и т.п.). Например,

```
ECHO Сейчас будут удалены все файлы в C:\Мои документы!
ECHO Для отмены нажмите Ctrl-C
PAUSE
DEL "C:\Мои документы\*.*"
```

1.1.10. Вызов внешних командных файлов

Из одного командного файла можно вызвать другой, просто указав его имя. Например:

```
@ECHO OFF
CLS
REM Вывод списка log-файлов
DIR C:\*.log
:: Передача выполнения файлу f.bat
f.bat
COPY A:\*.* C:\
PAUSE
```

Однако в этом случае после выполнения вызванного файла управление в вызывающий файл не передается, то есть в приведенном примере команда

```
COPY A:\*.* C:\
```

(и все следующие за ней команды) никогда не будет выполнена.

Для того, чтобы вызвать внешний командный файл с последующим возвратом в первоначальный файл, нужно использовать специальную команду CALL файл

Например:

```
@ECHO OFF
CLS
:: Вывод списка log-файлов
DIR C:\*.log
:: Передача выполнения файлу f.bat
CALL f.bat
COPY A:\*.* C:\
PAUSE
```

В этом случае после завершения работы файла f.bat управление вернется в первоначальный файл на строку, следующую за командой CALL (в нашем примере это команда COPY A:*.* C:\).

1.1.11. Операторы перехода GOTO и вызова CALL

Командный файл может содержать метки и команды GOTO перехода к этим меткам. Любая строка, начинающаяся с двоеточия :, воспринимается при обработке командного файла как метка. Имя метки задается набором символов, следующих за двоеточием до первого пробела или конца строки.

Пример 3. Пусть имеется командный файл следующего содержания:

```
@ECHO OFF
COPY %1 %2
GOTO Label1
```

ECHO Эта строка никогда не выполнится
:Label1
:: Продолжение выполнения
DIR %2
После того, как в этом файле мы доходим до команды
GOTO Label1
его выполнение продолжается со строки
:: Продолжение выполнения

В команде перехода внутри файла GOTO можно задавать в качестве метки перехода строку :EOF, которая передает управление в конец текущего пакетного файла (это позволяет легко выйти из пакетного файла без определения каких-либо меток в самом его конце).

Для перехода к метке внутри текущего командного файла кроме команды GOTO можно использовать и рассмотренную выше команду CALL:

CALL :метка аргументы

При вызове такой команды создается новый контекст текущего пакетного файла с заданными аргументами, и управление передается на инструкцию, расположенную сразу после метки. Для выхода из такого пакетного файла необходимо два раза достичь его конца. Первый выход возвращает управление на инструкцию, расположенную сразу после строки CALL, а второй выход завершает выполнение пакетного файла. Например, если запустить с параметром "Копия-1" командный файл следующего содержания:

@ECHO OFF
ECHO %1
CALL :2 Копия-2
:2
ECHO %1
то на экран выведутся три строки:
Копия-1
Копия-2
Копия-1

Таким образом, подобное использование команды CALL похоже на вызов подпрограмм в языках высокого уровня.

1.1.12. Оператор проверки условия IF

С помощью команды IF … ELSE (ключевое слово ELSE может отсутствовать) в пакетных файлах можно выполнять обработку условий нескольких типов. При этом если заданное после IF условие принимает истинное значение, система выполняет следующую за условием команду (или несколько команд, заключенных в круглые скобки), в противном случае выполняется команда (или несколько команд в скобках), следующие за ключевым словом ELSE.

1.1.12.1. Проверка значения переменной

Первый тип условия используется обычно для проверки значения переменной. Для этого применяются два варианта синтаксиса команды IF:

IF [NOT] строка1==строка2 команда1 [ELSE команда2]
(квадратные скобки указывают на необязательность заключенных в них параметров) или
IF [/I] [NOT] строка1 оператор_сравнения строка2 команда

Рассмотрим сначала первый вариант. Условие строка1==строка2 (здесь необходимо писать именно два знака равенства – как и в программах на C/C++) считается истинным при точном совпадении обеих строк. Параметр NOT указывает на то, что заданная команда выполняется лишь в том случае, когда сравниваемые строки не совпадают.

Для *группировки команд* могут использоваться круглые скобки. Иногда использование круглых скобок необходимо для правильной работы команды if…else – например для вывода на экран наибольшего из двух параметров, с которыми запущен командный файл, следует использовать оператор

if %1 GTR %2 (echo %1) else (echo %2)

Строки могут быть литеральными или представлять собой значения переменных (например, %1 или %TEMP%). Кавычки для литеральных строк **не требуются**. Например,

IF %1==%2 ECHO Параметры совпадают!

IF %1==windows ECHO значение первого параметра - windows

Отметим, что при сравнении строк, заданных переменными, следует проявлять определенную осторожность. Дело в том, что значение переменной может оказаться пустой строкой, и тогда может возникнуть ситуация, при которой выполнение командного файла аварийно завершится. Например, если вы не определили с помощью команды SET переменную MyVar, а в файле имеется условный оператор типа

IF %MyVar%==C:\ ECHO Ура!!!

то в процессе выполнения вместо %MyVar% подставится пустая строка и возникнет синтаксическая ошибка. Такая же ситуация может возникнуть, если одна из сравниваемых строк является значением параметра командной строки, так как этот параметр может быть не указан при запуске командного файла. Поэтому при сравнении строк нужно приписывать к ним в начале какой-нибудь символ, например:

IF -%MyVar%==-C:\ ECHO Ура!!!

С помощью команд IF и SHIFT можно в цикле обрабатывать все параметры командной строки файла, даже не зная заранее их количества. Например, следующий командный файл (назовем его primer.bat) выводит на экран имя запускаемого файла и все параметры командной строки:

```
@ECHO OFF
ECHO Выполняется файл: %0
ECHO.
ECHO Файл запущен со следующими параметрами...
:: Начало цикла
:BegLoop
IF -%1==- GOTO ExitLoop
ECHO %1
:: Сдвиг параметров
SHIFT
:: Переход на начало цикла
GOTO BegLoop
:ExitLoop
:: Выход из цикла
ECHO.
ECHO Все.
```

Если запустить primer.bat с четырьмя параметрами:

```
  primer.bat A B C D
```

то в результате выполнения на экран выведется следующая информация:

```
Выполняется файл: primer.bat

Файл запущен со следующими параметрами:
A
B
C
D
```

Все.
Рассмотрим теперь оператор IF в следующем виде:
IF [/I] строка1 оператор_сравнения строка2 команда
Синтаксис и значение операторов_сравнения представлены в табл. 2.

Таблица 2. Операторы сравнения в IF

Оператор	Значение
EQL	Равно
NEQ	Не равно
LSS	Меньше
LEQ	Меньше или равно
GTR	Больше
GEQ	Больше или равно

Пример 4 использования операторов сравнения:

```
@ECHO OFF
CLS
IF -%1 EQL –Вася ECHO Привет, Вася!
IF -%1 NEQ –Вася ECHO Привет, но Вы не Вася!
```

Ключ /I, если он указан, задает сравнение текстовых строк **без учета регистра**. Ключ /I можно также использовать и в форме строка1==строка2 команды IF. Например, условие

```
IF /I DOS==dos …
```

будет истинным.

1.1.12.2. Проверка существования заданного файла

Второй способ использования команды IF – это проверка существования заданного файла. Синтаксис для этого случая имеет вид:

```
IF [NOT] EXIST файл команда1 [ELSE команда2]
```

Условие считается истинным, если указанный файл существует. Кавычки для имени файла не требуются. Приведем пример командного файла, в котором с помощью такого варианта команды IF проверяется наличие файла, указанного в качестве параметра командной строки.

```
@ECHO OFF
IF -%1==- GOTO NoFileSpecified
IF NOT EXIST %1 GOTO FileNotExist

:: Вывод сообщения о найденном файле
ECHO Файл '%1' найден.
GOTO :EOF

:NoFileSpecified
:: Файл запущен без параметров
ECHO В командной строке не указано имя файла.
GOTO :EOF
```

```
:FileNotExist
:: Параметр командной строки задан, но файл не найден
ECHO Файл '%1' не найден.
```

1.1.12.3. Проверка наличия переменной среды

Аналогично файлам команда IF позволяет проверить наличие в системе определенной переменной среды:

IF DEFINED переменная команда1 [ELSE команда2]

Здесь условие DEFINED применяется подобно условию EXISTS наличия заданного файла, но принимает в качестве аргумента имя переменной среды и возвращает истинное значение, если эта переменная определена. Например:

```
@ECHO OFF
CLS
IF DEFINED MyVar GOTO :VarExists
ECHO Переменная MyVar не определена
GOTO :EOF
:VarExists
ECHO Переменная MyVar определена,
ECHO ее значение равно %MyVar%
```

1.1.12.4. Проверка кода завершения предыдущей команды

Еще один способ использования команды IF – это проверка кода завершения (кода выхода) предыдущей команды. Синтаксис для IF в этом случае имеет следующий вид:

IF [NOT] ERRORLEVEL число команда1 [ELSE команда2]

Здесь условие считается истинным, если последняя запущенная команда или программа завершилась с кодом возврата, равным, либо превышающим указанное число.

Рассмотрим командный файл, который копирует файл my.txt на диск C: без вывода на экран сообщений о копировании, а в случае возникновения какой-либо ошибки выдает предупреждение:

```
@ECHO OFF
XCOPY my.txt C:\ > NUL
:: Проверка кода завершения копирования
IF ERRORLEVEL 1 GOTO ErrOccurred
ECHO Копирование выполнено без ошибок.
GOTO :EOF

:ErrOccurred
ECHO При выполнении команды XCOPY возникла ошибка!
```

В операторе IF ERRORLEVEL … можно также применять операторы сравнения чисел, приведенные в табл. 2. Например:

IF ERRORLEVEL LEQ 1 GOTO Case1

Замечание. Иногда более удобным для работы с кодами завершения программ может оказаться использование переменной %ERRORLEVEL%. (строковое представление текущего значения кода ошибки ERRORLEVEL).

1.1.13. Организация циклов

В командных файлах для организации циклов используются несколько разновидностей оператора FOR, которые обеспечивают следующие функции:

- выполнение заданной команды для всех элементов указанного множества;
- выполнение заданной команды для всех подходящих имен файлов;
- выполнение заданной команды для всех подходящих имен каталогов;

- выполнение заданной команды для определенного каталога, а также всех его подкаталогов;
- получение последовательности чисел с заданными началом, концом и шагом приращения;
- чтение и обработка строк из текстового файла;
- обработка строк вывода определенной команды.

1.1.13.1. Цикл FOR … IN … DO …

Самый простой вариант синтаксиса команды FOR для командных файлов имеет следующий вид:

FOR %%переменная IN (множество)
DO команда [параметры]

Внимание!

Перед названием переменной должны стоять именно два знака процента (%%), а не один, как это было при использовании команды FOR непосредственно из командной строки!

Пример 5. Если в командном файле заданы строки

```
@ECHO OFF
FOR %%i IN (Раз, Два, Три) DO ECHO %%i
```

то в результате его выполнения на экран будет выведено следующее:

```
Раз
Два
Три
```

Параметр множество в команде FOR задает одну или более текстовых строк, разделенных запятыми, которые необходимо обработать с помощью заданной команды. Скобки здесь **обязательны**. Параметр команда [параметры] задает команду, выполняемую для каждого элемента множества, при этом вложенность команд FOR на одной строке **не допускается**. Если в строке, входящей во множество, используется запятая, то значение этой строки нужно заключить в кавычки. Например, в результате выполнения файла с командами

```
@ECHO OFF
FOR %%i IN ("Раз,Два",Три) DO ECHO %%i
```

на экран будет выведено

```
Раз,Два
Три
```

Параметр %%переменная представляет подставляемую переменную (счетчик цикла), причем здесь могут использоваться только имена переменных, **состоящие из одной буквы**. При выполнении команда FOR заменяет подставляемую переменную текстом каждой строки в заданном множестве, пока команда, стоящая после ключевого слова DO, не обработает все такие строки.

Замечание. Чтобы избежать путаницы с параметрами командного файла %0 — %9, для переменных следует использовать любые символы кроме 0 – 9.

Параметр множество в команде FOR может также представлять одну или несколько групп файлов. Например, чтобы вывести в файл список всех файлов с расширениями txt и prn, находящихся в каталоге C:\TEXT, без использования команды DIR, можно использовать командный файл следующего содержания:

```
@ECHO OFF
FOR %%f IN (C:\TEXT\*.txt C:\TEXT\*.prn) DO ECHO %%f >> list.txt
```

При таком использовании команды FOR процесс обработки продолжается, пока не обработаются все файлы (или группы файлов), указанные во множестве.

1.1.13.2. Цикл FOR /D … IN … DO …

Следующий вариант команды FOR реализуется с помощью ключа /D (directory – каталог):

FOR /D %переменная IN (набор) DO команда [параметры]

В случае, если набор содержит подстановочные знаки, то команда выполняется для всех подходящих имен каталогов, а не имен файлов. Скажем, выполнив следующий командный файл:

```
@ECHO OFF
CLS
FOR /D %%f IN (C:\*.*) DO ECHO %%f
```

мы получим список всех каталогов на диске C:, например:

```
C:\Arc
C:\CYR
C:\MSCAN
C:\Program Files
C:\TEMP
C:\WINNT
```

1.1.13.3. Цикл FOR /R … IN … DO …

С помощью ключа /R можно задать **рекурсию** в команде FOR:

FOR /R [[диск:]путь] %переменная IN (набор)
DO команда [параметры]

В этом случае заданная команда выполняется для каталога [диск:]путь, а также для **всех подкаталогов** этого пути. Если после ключа R не указано имя каталога, то выполнение команды начинается с текущего каталога.

Пример. Для распечатки всех файлов с расширением txt в текущем каталоге и всех его подкаталогах можно использовать следующий пакетный файл:

```
@ECHO OFF
CLS
FOR /R %%f IN (*.txt) DO PRINT %%f
```

Если вместо набора указана только точка (.), то команда проверяет все подкаталоги текущего каталога. Например, если мы находимся в каталоге C:\TEXT с двумя подкаталогами BOOKS и ARTICLES, то в результате выполнения файла:

```
@ECHO OFF
CLS
FOR /R %%f IN (.) DO ECHO %%f
```

на экран выведутся три строки:

```
C:\TEXT\.
C:\TEXT\BOOKS\.
C:\TEXT\ARTICLES\.
```

1.1.13.4. Цикл FOR /L … IN … DO …

Ключ /L позволяет реализовать с помощью команды FOR цикл со счетчиком, в этом случае синтаксис имеет следующий: вид:

FOR /L %переменная IN (начало,шаг,конец) DO команда [параметры]

Здесь заданная после ключевого слова IN тройка (начало, шаг, конец) задает последовательность чисел с заданными началом, концом и шагом приращения. Например, тройка (1, 1, 5) порождает последовательность (1 2 3 4 5), а тройка (5, -1, 1) - последовательность (5 4 3 2 1). Например, в результате выполнения следующего командного файла:

```
@ECHO OFF
CLS
FOR /L %%f IN (1,1,5) DO ECHO %%f
```

переменная цикла %%f получит значения от 1 до 5, и на экран будут выведены пять чисел:

1
2
3
4
5

Числа, получаемые в результате выполнения цикла FOR /L, можно использовать в **арифметических вычислениях**. Рассмотрим командный файл my.bat следующего содержания:

```
@ECHO OFF
CLS
FOR /L %%f IN (1,1,5) DO CALL :2 %%f
GOTO :EOF
:2
SET /A M=10*%1
ECHO 10*%1=%M%
```

В третьей строке в цикле происходит вызов нового контекста файла my.bat с текущим значением переменной цикла %%f в качестве параметра командной строки, причем управление передается на метку :2 (см. описание CALL в разделе "Изменения в командах перехода"). В шестой строке переменная цикла умножается на десять, и результат записывается в переменную M. Таким образом, в результате выполнения этого файла выведется следующая информация:

```
10*1=10
10*2=20
10*3=30
10*4=40
10*5=50
```

1.1.13.5. Цикл FOR /F … IN … DO …

Самые широкие возможности имеет команда FOR с ключом /F:

FOR /F ["ключи"] %переменная IN (набор) DO команда [параметры]

Здесь параметр набор содержит имена одного или нескольких файлов, которые по очереди открываются, читаются и обрабатываются. Обработка состоит в чтении файла, разбиении его на отдельные строки текста и выделении из каждой строки заданного числа подстрок. Затем найденная подстрока используется в качестве значения переменной при выполнении основного тела цикла (заданной команды).

По умолчанию ключ /F выделяет из каждой строки файла первое слово, очищенное от окружающих его пробелов. Пустые строки в файле пропускаются. Необязательный параметр "ключи" служит для переопределения заданных по умолчанию правил обработки строк. Ключи представляют собой заключенную в кавычки строку, содержащую приведенные в табл. 3 ключевые слова:

Таблица 3. Ключи в команде FOR /F

Ключ	Описание
EOL=C	Определение символа комментариев в начале строки (допускается задание только одного символа)
SKIP=N	Число пропускаемых при обработке строк в начале файла
DELIMS=XXX	Определение набора разделителей для замены заданных по умолчанию пробела и знака табуляции
TOKENS=X,Y,M-	Определение номеров подстрок, выделяемых из каждой строки

N	файла и передаваемых для выполнения в тело цикла

При использовании ключа TOKENS=X,Y,M-N создаются дополнительные переменные. Формат M-N представляет собой диапазон подстрок с номерами от M до N. Если последний символ в строке TOKENS= является звездочкой, то создается дополнительная переменная, значением которой будет весь текст, оставшийся в строке после обработки последней подстроки.

Разберем применение этой команды на примере пакетного файла parser.bat, который производит разбор файла myfile.txt:

```
@ECHO OFF
IF NOT EXIST myfile.txt GOTO :NoFile
FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN (myfile.txt) DO @ECHO %%i %%j
%%k
GOTO :EOF
:NoFile
ECHO Не найден файл myfile.txt!
```

Здесь во второй строке производится проверка наличия файла myfile.txt; в случае отсутствия этого файла выводится предупреждающее сообщение. Команда FOR в третьей строке обрабатывает файл myfile.txt следующим образом:

Пропускаются все строки, которые начинаются с символа точки с запятой (EOL=;).

Вторая и третья подстроки из каждой строки передаются в тело цикла, причем подстроки разделяются пробелами (по умолчанию) и/или запятыми (DELIMS=,).

В теле цикла переменная %%i используется для второй подстроки, %%j — для третьей, а %%k получает все оставшиеся подстроки после третьей.

Замечание. Имена переменных i, j, k должны следовать в алфавитном порядке.

В нашем примере переменная %%i явно описана в инструкции FOR, а переменные %%j и %%k описываются **неявно** с помощью ключа TOKENS=. Например, если в файле myfile.txt были записаны следующие три строки:

```
AAA BBBB CCCC,GGGG DDDD
EEEEE,JJJJ KKKK
;TTTT LLLL MMMMM
```

то в результате выполнения пакетного файла parser.bat на экран выведется следующее:

```
BBBB CCCC GGGG DDDD
JJJJ KKKK
```

Замечание

Ключ TOKENS= позволяет извлечь из одной строки файла до 26 подстрок, поэтому запрещено использовать имена переменных, начинающиеся не с букв английского алфавита (a–z). Следует помнить, что имена переменных FOR являются **глобальными**, поэтому одновременно не может быть активно более 26 переменных.

Команда FOR /F также позволяет обработать отдельную строку. Для этого следует ввести нужную строку в кавычках вместо набора имен файлов в скобках. Строка будет обработана так, как будто она взята из файла. Например, файл следующего содержания:

```
@ECHO OFF
FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN ("AA CC BB,GG DD") DO @ECHO
%%i %%j %%k
```

при своем выполнении напечатает

```
CC BB GG DD
```

Вместо явного задания строки для разбора можно пользоваться переменными среды, например:

```
@ECHO OFF
SET M=AAA БББ BBBB,ГГГГГ ДДДД
FOR /F "EOL=; TOKENS=2,3* DELIMS=,
  " %%i IN ("%M%") DO @ECHO %%i %%j %%k
```

Наконец, команда FOR /F позволяет обработать **строку вывода другой команды**. Для этого следует вместо набора имен файлов в скобках ввести строку вызова команды в апострофах (не в кавычках!). Строка передается для выполнения интерпретатору команд cmd.exe, а вывод этой команды записывается в память и обрабатывается так, как будто строка вывода взята из файла. Например, следующий командный файл:

```
@ECHO OFF
CLS
ECHO Имена переменных среды:
ECHO.
FOR /F "DELIMS==" %%i IN ('SET') DO ECHO %%i
```

выведет перечень имен всех переменных среды, определенных в настоящее время в системе.

В цикле FOR допускается применение тех же синтаксических конструкций (операторов), что и для заменяемых параметров - таблица 4.

Таблица 4. Операторы для переменных команды FOR

Операторы	Описание
%~Fi	Переменная %i расширяется до полного имени файла
%~Di	Из переменной %i выделяется только имя диска
%~Pi	Из переменной %i выделяется только путь к файлу
%~Ni	Из переменной %i выделяется только имя файла
%~Xi	Из переменной %i выделяется расширение имени файла
%~Si	Значение операторов N и X для переменной %i изменяется так, что они работают с кратким именем файла
%~Zi	Определяется длина (размер) файла с указанным именем

Замечание. Если планируется использовать расширения подстановки значений в команде FOR, то следует внимательно подбирать имена переменных, чтобы они не пересекались с обозначениями формата.

Например, если мы находимся в каталоге C:\Program Files\Far и запустим командный файл следующего содержания:

```
@ECHO OFF
CLS
FOR %%i IN (*.txt) DO ECHO %%~Fi
```

то на экран выведутся полные имена всех файлов с расширением txt:

```
C:\Program Files\Far\Contacts.txt
C:\Program Files\Far\ReadMe.txt
C:\Program Files\Far\register.txt
C:\Program Files\Far\WhatsNew.txt
```

Вычисление суммарной длины всех файлов в заданном подкаталоге

```
@ECHO OFF
SETLOCAL ENABLEDELAYEDEXPANSION
Set /a Size = 0
For %%I in (%1\*.*) do set /a Size= Size + %%~zI
Echo %Size%
```

1.1.13.6. Циклы и связывание времени выполнения для переменных

Как и в рассмотренном выше примере с составными выражениями, при обработке переменных среды внутри цикла могут возникать труднообъяснимые ошибки, связанные с ранними связыванием переменных. Рассмотрим пример. Пусть имеется командный файл следующего содержания:

```
SET a=
FOR %%i IN (Раз,Два,Три) DO SET a=%a%%%i
ECHO a=%a%
```

В результате его выполнения на экран будет выведена строка "a=Три", то есть фактически команда

```
FOR %%i IN (Раз,Два,Три) DO SET a=%a%%%i
```

равносильна команде

```
FOR %%i IN (Раз,Два,Три) DO SET a=%%i
```

Для исправления ситуации нужно, как и в случае с составными выражениями, вместо знаков процента (%) использовать восклицательные знаки и предварительно включить режим связывания времени выполнения командой SETLOCAL ENABLEDELAYEDEXPANSION. Таким образом, наш пример следует переписать следующим образом:

```
SETLOCAL ENABLEDELAYEDEXPANSION
SET a=
FOR %%i IN (One,Two,Three) DO SET a=!a!%%i
ECHO a=%a%
```

В этом случае на экран будет выведена строка "a=OneTwoThree".

1.1.13.7. Команда Findstr и ее использование в цикле

Назначение команды – поиск строк в текстовых файлах.

```
FINDSTR [/B] [/E] [/L] [/R] [/S] [/I] [/X] [/V] [/N] [/M] [/O] [/P] [/F:файл]
      [/C:строка] [/G:файл] [/D:список_папок] [/A:цвета] [/OFF[LINE]]
      строки [[диск:][путь]имя_файла[ ...]]
```

```
 /L        Поиск строк дословно.
 /R        Поиск строк как регулярных выражений.
 /S        Поиск файлов в текущей папке и всех ее подпапках.
 /I        Определяет, что поиск будет вестись без учета регистра.
 /X        Печатает строки, которые совпадают точно.
 /V        Печатает строки, не содержащие совпадений с искомыми.
 /N        Печатает номер строки, в которой найдено совпадение, и ее содержимое.
 /M        Печатает только имя файла, в которой найдено совпадение.
 /O        Печатает найденный строки через пустую строку.
 /P        Пропускает строки, содержащие непечатаемые символы.
 /F:файл    Читает список файлов из заданного файла (/ для консоли).
 /C:строка  Использует заданную строку как искомую фразу поиска.
  /D:список_папок    Поиск в списке папок (разделяются точкой с запятой).
 строка     Искомый текст.
 [диск:][путь]имя_файла - задает имя файла или файлов.
```

Использовать пробелы для разделения нескольких искомых строк, если аргумент не имеет префикса /C. Например, 'FINDSTR "Привет мир" a.b' ищет "Привет" или "мир" в файле a.b, а команда 'FINDSTR /C:"Привет мир" a.b' ищет строку "Привет мир" в файле a.b.

Краткая сводка по синтаксису регулярных выражений:

. Любой символ.

* Повтор: ноль или более вхождений предыдущего символа или класса

^ Позиция в строке: начало строки

$ Позиция в строке: конец строки

[класс] Класс символов: любой единичный символ из множества

[^класс] Обратный класс символов: любой единичный символ из дополнения

[x-y] Диапазон: любые символы из указанного диапазона

\x Служебный символ: символьное обозначение служебного символа x

\<xyz Позиция в слове: в начале слова

xyz\> Позиция в слове: в конце слова

Пример командного файла для поиска в файле num.txt по образцу строк, в которых присутствует хотя бы одна двоичная цифра.

```
@echo off
set /a kol=0
for /f %%b in ('findstr /rc:"[0-1]" num.txt') do set /a kol=kol+1
echo %kol%
```

2. МЕТОДИКА ВЫПОЛНЕНИЯ

9. Неформально ознакомиться с теоретическими сведениями.

10. Для подготовки текстов командных файлов рекомендуется использовать блокнот (Notepad). При этом следует избегать использования в выводимых на экран результатах работы командного файла букв русского алфавита.

11. Разработать и выполнить командные файлы (КФ), выполняющие следующие функции:

1. Вывод на экран имен всех файлов с указанным расширением, находящихся в каталоге, имя которого задается при запуске командного файла первым параметром. Расширение файлов задается вторым параметром.

2. Среди введенных с клавиатуры целых чисел (использовать SET /P) найти наибольшее и наименьшее. Признак конца ввода – знак -.

3. В заданном каталоге и его подкаталогах найти общее количество подкаталогов. На экран вывести только требуемый результат.

4. В каталогах, имена которых заданы первым и вторым параметрами командного файла, найти и вывести на экран имена файлов (расширения могут быть любые), присутствующие как в первом, так и во втором каталоге. Следует использовать только один оператор FOR.

5. Вычисление и вывод на экран значения факториала целого числа, задаваемого при запуске КФ. Предусмотреть проверку заданного значения и при задании отрицательного значения или значения, превышающего максимально возможную величину, выводить соответствующие сообщения. Для проверки правильности вычислений использовать калькулятор.

4. Разработать и выполнить КФ в соответствии с табл. 5 (индивидуальные задания для студентов).

Таблица 5. Индивидуальные задания для студентов

но-мер	Действия, выполняемые КФ
1	1.Подсчет количества целых чисел в текстовом файле. Считать, что слова в файле записаны в формате ОДНО СЛОВО В СТРОКЕ. Слово – это целое число (состоящее из десятичных цифр) или последовательность букв латинского алфавита

	(начинающаяся с буквы). Имя файла задается первым параметром КФ. 2. Вывод на экран списка файлов, хранящихся в указанном первым параметром каталоге и созданных в первом полугодии (месяцы 1-6) года, указанного вторым параметром КФ.
2	1. В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти файл наибольшего размера с расширением, указанным вторым параметром КФ. 2. В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти ТРИ файла самого большого размера. Вывести имена файлов, их размеры и даты создания
3	1. Разбиение текстового файла, имя которого задано первым параметром КФ, на три файла с именами 1.txt, 2.txt и 3.txt. Количество строк в каждом из этих файлов задано вторым, третьим и четвертым параметрами КФ. Проверить наличие указанного исходного файла и вывести сообщение о его отсутствии, проверить наличие остальных параметров и их значения на допустимость 2. В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти суммарный объем файлов, имеющих расширение, указанное вторым параметром КФ.
4	1. Удаление из каталога, заданного первым параметром, файлов, которые присутствуют и в каталоге, указанным вторым параметром. Предусмотреть запрос пользователю на подтверждение удаления. 2. В каталоге, указанном первым параметром КФ, и его подкаталогах, найти файлы, созданные во второй половине рабочего дня (после 14 часов) и скопировать их в отдельный подкаталог.
5	1. Нахождение суммарного объема файлов с атрибутом system, хранящихся в каталоге, имя которого задано первым параметром КФ. 2. Проверить наличие файла Numb.txt в каталоге, указанном первым параметром КФ. Прочитать целые числа из файла, найти среди них простые и вывести результаты на экран. Считать, что все числа не превышают значения 2500.
6	1. Поиск на диске C: (или любом доступном диске) файла с заданным именем. Если файл не найден – вывод сообщения. Если файл найден – открыть его для редактирования. 2. Проверка наличия на диске в каталоге, указанном первым параметром КФ, файла FNames.txt, содержащего список имен файлов и подкаталогов. Если он есть – проверка наличия перечисленных в списке файлов и вывод имен отсутствующих. Если файла FNames нет, создание его и запись имен файлов и подкаталогов.
7	1. Вывод списка DLL (хранящихся на доступном диске), созданных до 12.2008 размером до 12000 байтов. 2. Проверка наличия на диске в каталоге, указанном первым параметром КФ, файла Numbers.txt, содержащего 2 столбца целых чисел, столбцы располагаются с позиций 2 и 20 и отделены пробелами. Если файла нет – вывод сообщения. Если файл есть, создать новый файл, содержащий три столбца, в третий поместить сумму чисел из двух первых столбцов.
8	1. Просмотр содержимого каталога, указанного первым параметром КФ. Необходимо: 1. создать подкаталоги с именами EXE, TXT, CMD, DOC и OTHER. 2. В каждый подкаталог скопировать файлы с соответствующими расширениями. 3. Пустые подкаталоги удалить. 2. В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти файлы наибольшего и наименьшего размеров. Вывести имена файлов, их размеры и даты создания.
9	1. Проверка наличия трех текстовых файлов на диске и объединения их в один файл.

	2. Подсчет количества вещественных чисел и целых чисел в текстовом файле. Вещественные и целые числа подсчитать отдельно. Считать, что слова в файле записаны в формате ОДНО СЛОВО В СТРОКЕ. Слово – это целое число (состоящее из десятичных цифр) или последовательность букв латинского алфавита (начинающаяся с буквы) или последовательность десятичных цифр с точкой (.) внутри строки. Имя файла задается первым параметром КФ.
10	1. Подсчет количества слов в текстовом файле, содержащем целые числа и слова. Считать, что слова в файле записаны в формате ОДНО СЛОВО В СТРОКЕ. Число – это целое число (состоящее из десятичных цифр). Слово - последовательность букв латинского алфавита (начинающаяся с буквы). Имя файла задается первым параметром КФ. 2. Просмотр содержимого каталога, указанного первым параметром КФ. Необходимо: 1. создать подкаталоги с именами 1, 2, ..., 12. 2. В каждый подкаталог скопировать файлы, созданные в соответствующие месяцы. 3. Пустые подкаталоги удалить.
11	1. Подсчет количества строк в текстовом файле, имя которого задано первым параметром КФ. Проверить наличие указанного файла и вывести сообщение о его отсутствии. 2. С помощью команды DIR вывести на экран имена файлов, находящихся в каталоге, имя которого задано первым параметром КФ. Второй и остальные параметры задают расширения файлов, имена которых выводить не следует. Рекомендуется с помощью ATRIB присвоить некоторым файлам атрибут СКРЫТЫЙ – такие файлы DIR не показывает.
12	1.Поиск текстового файла по его содержимому. Считать, что слова в текстовых файлах записаны в формате ОДНО СЛОВО В СТРОКЕ. Искомое слово задается первым параметром КФ.
13	1. Вывод на экран аргументов, с которыми КФ был запущен. Число аргументов от 4 до 11. При неверном числе аргументов ничего не выполнять, сообщить об ошибке. 2. Поиск и вывод на экран минимального и максимального значения аргумента КФ. Предполагается, что все аргументы КФ – целые положительные числа.

Примечание. Для решения задач 1, 9 и 10 рекомендуется использовать команду Findstr

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Вывод сообщений и дублирование команд.
2. Использование параметров командной строки.
3. Переменные среды, получение и изменение их значений.
4. Операции со строковыми и числовыми переменными.
5. Проверка существования заданного файла и наличия переменной среды.
6. Выполнение заданной команды для всех элементов указанного множества.
7. Выполнение заданной команды для всех подходящих имен файлов.
8. Выполнение заданной команды для всех подходящих имен каталогов.
9. Выполнение заданной команды для определенного каталога, а также всех его подкаталогов.
10. Получение последовательности чисел с заданными началом, концом и шагом приращения.
11. Чтение и обработка строк из текстового файла.
12. Команда Findstr. Назначение. Ключи. Использование регулярных выражений в команде. Задание и использование класса цифр и класса букв через диапазон.

13.Операторы перехода и вызова.

14.Какое минимальное количество строк (включая @echo off) должен иметь командный файл, выводящий на экран минимальное значения двух числовых аргументов?

15. Какое минимальное количество строк (включая @echo off) должен иметь командный файл, выводящий на экран минимальное значения трех числовых аргументов?

Лабораторная работа №3. Оболочка командной строки Windows PowerShell (4 часа)

Цель работы – знакомство с основными возможностями оболочки командной строки Windows PowerShell

1.1. Цели и задачи создания новой оболочки

Новая оболочка Windows PowerShell была задумана разработчиками Microsoft как более мощная среда для написания сценариев и работы из командной строки. Разработчики PowerShell преследовали несколько целей, главная из которых – создание среды составления сценариев, которая наилучшим образом подходила бы для современных версий ОС Windows и была бы более функциональной, расширяемой и простой в использовании, чем какой-либо аналогичный продукт для любой другой ОС. В первую очередь эта среда должна была подходить для решения задач, стоящих перед системными администраторами, а также удовлетворять требованиям разработчиков программного обеспечения, предоставляя им средства для быстрой реализации интерфейсов управления к создаваемым приложениям.

Для достижения этих целей были решены следующие задачи:

- Обеспечение прямого доступа из командной строки к объектам COM, WMI и .NET. В новой оболочке присутствуют команды, позволяющие в интерактивном режиме работать с COM-объектами, а также с экземплярами классов, определенных в информационных схемах WMI и .NET.
- Организация работы с произвольными источниками данных в командной строке по принципу файловой системы. Например, навигация по системному реестру или хранилищу цифровых сертификатов выполняется из командной строки с помощью аналога команды CD интерпретатора Cmd.exe.
- Разработка интуитивно понятной унифицированной структуры встроенных команд, основанной на их функциональном назначении. В новой оболочке имена всех внутренних команд (в PowerShell они называются командлетами) соответствуют шаблону "глагол-существительное", например, Get-Process (получить информацию о процессе), Stop-Service (остановить службу), Clear-Host (очистить экран консоли) и т.д. Для одинаковых параметров внутренних команд используются стандартные имена, структура параметров во всех командах идентична, все команды обрабатываются одним синтаксическим анализатором. В результате облегчается запоминание и изучение команд.
- Обеспечение возможности расширения встроенного набора команд. Внутренние команды PowerShell могут дополняться командами, создаваемыми пользователем. При этом они полностью интегрируются в оболочку, информация о них может быть получена из стандартной справочной системы PowerShell.
- Организация поддержки знакомых команд из других оболочек. В PowerShell на уровне псевдонимов собственных внутренних команд поддерживаются наиболее часто используемые стандартные команды из оболочки Cmd.exe и Unix-оболочек. Например, если пользователь, привыкший работать с Unix-оболочкой, выполнит ls, то он получит ожидаемый результат: список файлов в текущем каталоге (то же самое относится к команде dir).
- Разработка полноценной встроенной справочной системы для внутренних команд. Для большинства внутренних команд в справочной системе дано подробное описание и примеры

использования. В любом случае встроенная справка по любой внутренней команде будет содержать краткое описание всех ее параметров.

- Реализация автоматического завершения при вводе с клавиатуры имен команд, их параметров, а также имен файлов и папок. Данная возможность значительно упрощает и ускоряет ввод команд с клавиатуры.

Главной особенностью среды PowerShell, отличающей ее от всех других оболочек командной строки, является то, что единицей обработки и передачи информации здесь является **объект**, а не строка текста.

1.2. **Отличие PowerShell от других оболочек – ориентация на объекты**

При разработке любого языка программирования одним из основных является вопрос о том, какие типы данных и каким образом будут в нем представлены. При создании PowerShell разработчики решили не изобретать ничего нового и воспользоваться унифицированной объектной моделью .NET.

Рассмотрим пример. В Windows есть консольная утилита tasklist.exe, которая выдает информацию о процессах, запущенных в системе:

```
C:\>tasklist

Имя образа                     PID Имя сессии    № сеанса    Память
========================= ====== ============= ======== ============
System Idle Process              0             0       16 КБ
System                           4             0       32 КБ
smss.exe                       560             0       68 КБ
csrss.exe                      628             0    4 336 КБ
winlogon.exe                   652             0    3 780 КБ
services.exe                   696             0    1 380 КБ
lsass.exe                      708             0    1 696 КБ
svchost.exe                    876             0    1 164 КБ
svchost.exe                    944             0    1 260 КБ
svchost.exe                   1040             0   10 144 КБ
svchost.exe                   1076             0      744 КБ
svchost.exe                   1204             0      800 КБ
spoolsv.exe                   1296             0    1 996 КБ
kavsvc.exe                    1516             0    9 952 КБ
klnagent.exe                  1660             0    5 304 КБ
klswd.exe                     1684             0       64 КБ
```

Предположим, что мы в командном файле интерпретатора Cmd.exe с помощью этой утилиты хотим определить, сколько оперативной памяти тратит процесс kavsvc.exe. Для этого нужно выделить из выходного потока команды tasklist соответствующую строку, извлечь из нее подстроку, содержащую нужное число и убрать пробелы между разрядами. В PowerShell задача решается с помощью команды get-process, которая возвращает **коллекцию объектов**, каждый из которых соответствует одному запущенному процессу. Для определения памяти, затрачиваемой процессом kavsvc.exe, нет необходимости в дополнительных манипуляциях с текстом, достаточно просто взять значение свойства WS объекта, соответствующего данному процессу.

Наконец, объектная модель .NET позволяет PowerShell напрямую использовать функциональность различных библиотек, являющихся частью платформы .NET. Например, чтобы узнать, каким днем недели было 9 ноября 1977 года, в PowerShell можно выполнить следующую команду:

```
(get-date "09.11.1974").dayofweek.tostring()
```

В этом случае команда get-date возвращает .NET-объект DateTime, имеющий свойство DayOfWeek, при обращении к которому вычисляется день недели для соответствующей даты.

1.3. Запуск оболочки. Выполнение команд

Для запуска оболочки следует нажать на кнопку Пуск (Start), открыть меню Все программы (All Programs), выбрать элемент Стандартные, Windows PowerShell и Windows PowerShell ISE. Другой вариант запуска оболочки – пункт Выполнить… (Run) в меню Пуск (Start), ввести имя файла powershell_ise и нажать кнопку OK.

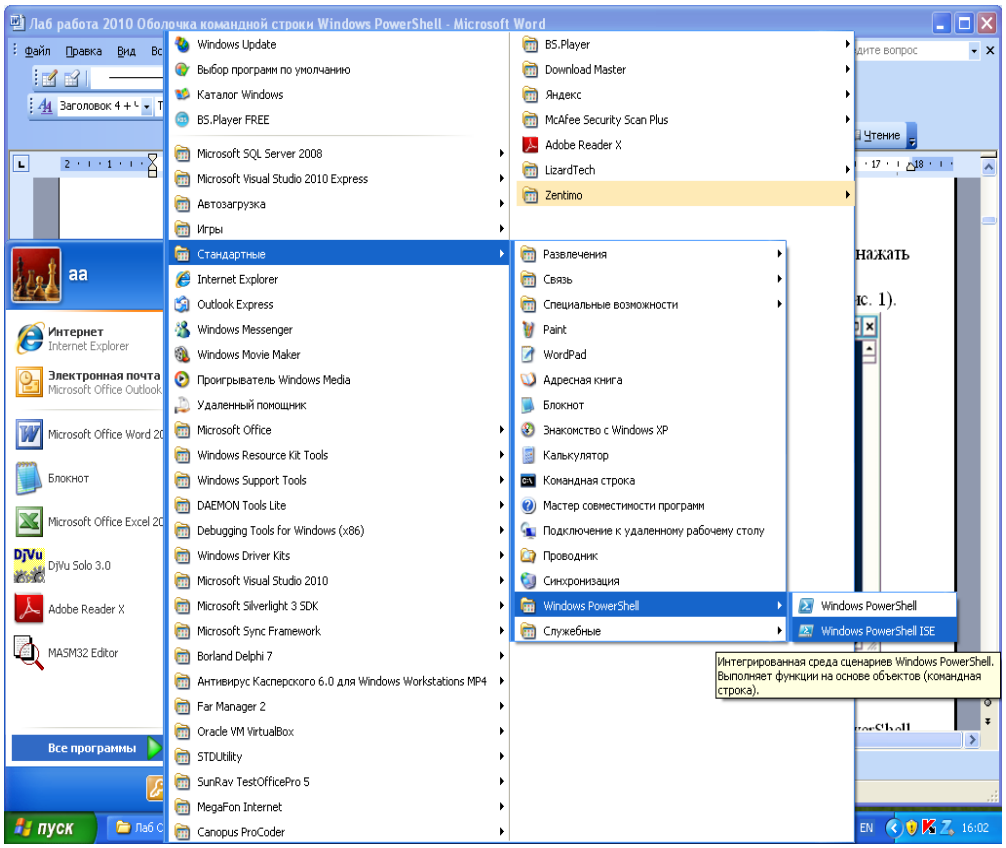


Рис. 1. Запуск PowerShell ISE с помощью меню

В результате откроется новое командное окно с приглашением вводить команды (рис. 2).

В нижней части окна вводятся команды. Средняя часть окна содержит результаты выполнения введенной команды или сообщения об ошибках.

Верхняя часть используется для работы с командными файлами.

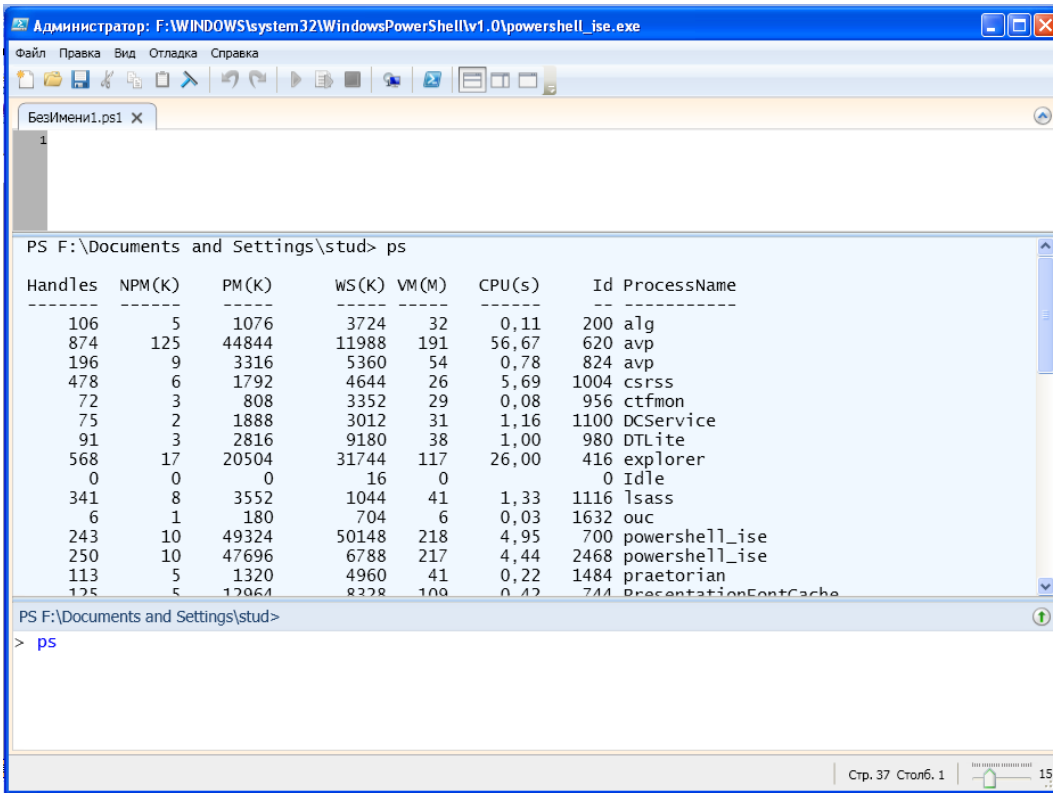


Рис. 2. Командное окно оболочки PowerShell ISE

Выполним первую команду в PowerShell - команду ps – список выполняющихся процессов (команды в PowerShell обрабатываются без учета регистра). На экран будет выведен список выполняющихся процессов.

Предыстория введенных команд работает также, как и в CMD.

1.4. Типы команд PowerShell

В оболочке PowerShell поддерживаются команды четырех типов: командлеты, функции, сценарии и внешние исполняемые файлы.

Первый тип – так называемые **командлеты** (cmdlet). Этот термин используется пока только внутри PowerShell. Командлет – аналог внутренней команды интерпретатора командной строки – представляет собой класс .NET, порожденный от базового класса Cmdlet; разрабатываются командлеты с помощью пакета PowerShell Software Developers Kit (SDK). Единый базовый класс Cmdlet гарантирует совместимый синтаксис всех командлетов, а также автоматизирует анализ параметров командной строки и описание синтаксиса командлетов для встроенной справки. Командлеты рассматриваются в данной работе. С командами других типов можно ознакомиться, используя [1].

Данный тип команд компилируется в динамическую библиотеку (DLL) и подгружается к процессу PowerShell во время запуска оболочки (то есть сами по себе командлеты не могут быть запущены как приложения, но в них содержатся исполняемые объекты). Командлеты – это аналог внутренних команд традиционных оболочек.

Следующий тип команд – **функции**. Функция – это блок кода на языке PowerShell, имеющий название и находящийся в памяти до завершения текущего сеанса командной оболочки. Функции, как и командлеты, поддерживают именованные параметры. Анализ синтаксиса функции производится один раз при ее объявлении.

Сценарий – это блок кода на языке PowerShell, хранящийся во внешнем файле с расширением ps1. Анализ синтаксиса сценария производится при каждом его запуске.

Последний тип команд – **внешние исполняемые файлы**, которые выполняются обычным образом операционной системой.

1.5. Имена и синтаксис командлетов

В PowerShell аналогом внутренних команд являются командлеты. Командлеты могут быть очень простыми или очень сложными, но каждый из них разрабатывается для решения одной, узкой задачи. Работа с командлетами становится по-настоящему эффективной при использовании их композиции (конвейеризации объектов между командлетами).

Команды Windows PowerShell следуют определенным правилам именования:

- Команды Windows PowerShell состоят из глагола и существительного (всегда в единственном числе), разделенных тире. Глагол задает определенное действие, а существительное определяет объект, над которым это действие будет совершено. Команды записываются на английском языке. Пример: Get-Help вызывает интерактивную справку по синтаксису Windows PowerShell.
- Перед **параметрами** ставится символ «-». Например: Get-Help –Detailed.
- В Windows PowerShell также включены псевдонимы многих известных команд. Это упрощает знакомство и использование Windows PowerShell. Пример: команды help (классический стиль Windows) и man (классический стиль Unix) работают так же, как и Get-Help.

Например, Get-Process (получить информацию о процессе), Stop-Service (остановить службу), Clear-Host (очистить экран консоли) и т.д. Чтобы просмотреть список командлетов, доступных в ходе текущего сеанса, нужно выполнить командлет Get-Command.

По умолчанию командлет Get-Command выводит сведения в трех столбцах: CommandType, Name и Definition. При этом в столбце Definition отображается синтаксис командлетов (многоточие (…) в столбце синтаксиса указывает на то, что данные обрезаны).

Замечание. Косые черты (/ и \) вместе с параметрами в оболочке Windows PowerShell не используются.

В общем случае синтаксис командлетов имеет следующую структуру:

имя_командлета –параметр1 -параметр2 аргумент1 аргумент2

Здесь параметр1 – параметр (переключатель), не имеющий значения; параметр2 – имя параметра, имеющего значение аргумент1; аргумент2 – параметр, не имеющий имени. Например, командлет Get-Process имеет параметр Name, который определяет имя процесса, информацию о котором нужно вывести. Имя этого параметра указывать необязательно. Таким образом, для получения сведений о процессе Far можно ввести либо команду Get-Process -Name Far, либо команду Get-Process Far.

1.6. Автоматическое завершение команд (автозавершение ввода команд)

Находясь в оболочке PowerShell, можно ввести часть какой-либо команды, нажать клавишу <Tab> и система попытается сама завершить ввод этой команды.

Подобное автоматическое завершение срабатывает, во-первых, для имен файлов и путей файловой системы. При нажатии клавиши <Tab> PowerShell автоматически расширит частично введенный путь файловой системы до первого найденного совпадения. При повторении нажатия клавиши <Tab> производится циклический переход по имеющимся возможностям выбора. Также в PowerShell реализована возможность автоматического завершения путей файловой системы на основе шаблонных символов (? и *). Например, если ввести команду cd c:\pro*files и нажать клавишу <Tab>, то в строке ввода появится команда cd 'C:\Program Files'.

Во-вторых, в PowerShell реализовано автозавершение имен командлетов и их параметров. Если ввести первую часть имени командлета (глагол) и дефис, нажать после этого клавишу <Tab>, то система подставит имя первого подходящего командлета (следующий подходящий вариант имени выбирается путем повторного нажатия <Tab>). Аналогичным образом автозавершение срабатывает для частично введенных имен параметров командлета: нажимая клавишу <Tab>, мы будем циклически перебирать подходящие имена.

Наконец, PowerShell позволяет автоматически завершать имена используемых переменных (объектов) и имена свойств объектов.

1.7. Псевдонимы команд

Механизм псевдонимов, реализованный в оболочке PowerShell, дает возможность пользователям выполнять команды по их **альтернативным именам** (например, вместо команды Get-Childitem можно пользоваться псевдонимом dir). В PowerShell заранее определено много псевдонимов, можно также добавлять собственные псевдонимы в систему.

Псевдонимы в PowerShell делятся на два типа. Первый тип предназначен для совместимости имен с разными интерфейсами. Псевдонимы этого типа позволяют пользователям, имеющим опыт работы с другими оболочками (Cmd.exe или Unix-оболочки), использовать знакомые им имена команд для выполнения аналогичных операций в PowerShell, что упрощает освоение новой оболочки, позволяя не тратить усилий на запоминание новых команд PowerShell. Например, пользователь хочет очистить экран. Если у него есть опыт работы с Cmd.exe, то он, естественно, попробует выполнить команду cls. PowerShell при этом выполнит командлет Clear-Host, для которого cls является псевдонимом и который выполняет требуемое действие – очистку экрана. Для пользователей Cmd.exe в PowerShell определены псевдонимы cd, cls, copy, del, dir, echo, erase, move, popd, pushd, ren, rmdir, sort, type; для пользователей Unix – псевдонимы cat, chdir, clear, diff, h, history, kill, lp, ls, mount, ps, pwd, r, rm, sleep, tee, write.

Узнать, какой именно командлет скрывается за знакомым псевдонимом, можно с помощью командлета Get-Alias:

```
PS C:\> Get-Alias cd
CommandType     Name        Definition
-----------     ----        ----------
Alias           cd          Set-Location
```

Псевдонимы второго типа (стандартные псевдонимы) в PowerShell предназначены для быстрого ввода команд. Такие псевдонимы образуются из имен командлетов, которым они соответствуют. Например, глагол Get сокращается до g, глагол Set сокращается до s, существительное Location сокращается до l и т.д. Таким образом, для командлету Set-Location соответствует псевдоним sl, а командлету Get-Location – псевдоним gl.

Просмотреть список всех пседонимов, объявленных в системе, можно с помощью командлета Get-Alias без параметров. Определить собственный псевдоним можно с помощью командлета Set-Alias.

1.8. Справочная система PowerShell

В PowerShell предусмотрено несколько способов получения справочной информации внутри оболочки.

Краткую справку по одному командлету можно получить с помощью параметра ? (вопросительный знак), указанного после имени этого командлета. Например:

```
PS C:\> get-process -?
```

Вместо help или man в Windows PowerShell можно также использовать команду Get-Help. Ее синтаксис описан ниже:

- Get-Help выводит на экран справку об использовании справки
- Get-Help * перечисляет все команды Windows PowerShell
- Get-Help *команда* выводит справку по соответствующей команде
- Get-Help команда -Detailed выводит подробную справку с примерами команды

Использование команды help для получения подробных сведений о команде help:

```
Get-Help
Get-Help -Detailed.
```

Комана Get-Help позволяет просматривать справочную информацию не только о разных командлетах, но и о синтаксисе языка PowerShell, о псевдонимах и т. д.

Например, чтобы прочитать справочную информацию об использовании массивов в PowerShell, нужно выполнить следующую команду: Get-Help about_array.

Командлет Get-Help выводит содержимое раздела справки на экран сразу целиком. Функции man и help позволяют справочную информацию выводить поэкранно (аналогично команде MORE интерпретатора Cmd.exe), например: man about_array.

1.9. Конвейеризация и управление выводом команд Windows PowerShell

Ранее было рассмотрено понятие конвейеризации (или композиции) команд интерпретатора Cmd.exe, когда выходной поток одной команды перенаправлялся во входной поток другой, объединяя тем самым две команды вместе. Подобные конвейеры команд используются в большинстве оболочек командной строки и являются средством, позволяющим передавать информацию между разными процессами. Механизм композиции команд представляет собой, вероятно, наиболее ценную концепцию, используемую в интерфейсах командной строки. Конвейеры не только снижают усилия, прилагаемые при вводе сложных команд, но и облегчают отслеживание потока работы в командах.

В оболочке PowerShell также очень широко используется механизм конвейеризации команд, однако здесь по конвейеру передается не поток текста, как во всех других оболочках, а объекты. При этом с элементами конвейера можно производить различные манипуляции: фильтровать объекты по определенному критерию, сортировать и группировать объекты, изменять их структуру (ниже мы подробнее рассмотрим операции фильтрации и сортировки элементов конвейера).

1.9.1. Конвейеризация объектов в PowerShell

Конвейер в PowerShell – это последовательность команд, разделенных между собой знаком | (вертикальная черта). Каждая команда в конвейере получает объект от предыдущей команды, выполняет определенные операции над ним и передает следующей команде в конвейере. С точки зрения пользователя, объекты упаковывают связанную информацию в форму, в которой информацией проще манипулировать как единым блоком и из которой при необходимости извлекаются определенные элементы.

Передача данных между командами в виде объектов имеет большое преимущество над обычным обменом информацией посредством потока текста. Ведь команда, принимающая поток текста от другой утилиты, должна его проанализировать, разобрать и выделить нужную ей информацию, а это может быть непросто, так как обычно вывод команды больше ориентирован на визуальное восприятие человеком (это естественно для интерактивного режима работы), а не на удобство последующего синтаксического разбора.

При передаче по конвейеру объектов этой проблемы не возникает, здесь нужная информация извлекается из элемента конвейера простым обращением к соответствующему свойству объекта. Однако возникает новый вопрос: каким образом узнать, какие именно свойства есть у объектов, передаваемых по конвейеру? Ведь при выполнении того или иного командлета мы на экране видим только одну или несколько колонок отформатированного текста.

Пример Запустим командлет Get-Process, который выводит информацию о запущенных в системе процессах:

```
PS C:\> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    158      11    45644      22084   126   159.69   2072 AcroRd32
     98       5     1104        284    32     0.10    256 alg
```

```
 39      1     364       364    17     0.26   1632 ati2evxx
 57      3    1028       328    30     0.38    804 atiptaxx
434      6    2548      3680    27    21.96    800 csrss
 64      3     812       604    29     0.22   1056 ctfmon
364     11   14120      9544    69    11.82    456 explorer
 24      2    1532      2040    29     5.34   2532 Far
```

Фактически на экране мы видим только сводную информацию (результат форматирования полученных данных), а не полное представление выходного объекта. Из этой информации непонятно, сколько точно свойств имеется у объектов, генерируемых командой Get-Process, и какие имена имеют эти свойства. Например, мы хотим найти все "зависшие" процессы, которые не отвечают на запросы системы. Можно ли это сделать с помощью командлета Get-Process, какое свойство нужно проверять у выводимых объектов?

Для ответа на подобные вопросы нужно научиться исследовать структуру объектов PowerShell, узнавать, какие свойства и методы имеются у этих объектов.

1.9.2. Просмотр структуры объектов

Для анализа структуры объекта, возвращаемого определенной командой, проще всего направить этот объект по конвейеру на командлет Get-Member (псевдоним gm), например:

```
PS C:\> Get-Process | Get-Member
   TypeName: System.Diagnostics.Process

Name                 MemberType     Definition
----                 ----------     ----------
Handles              AliasProperty  Handles = Handlecount
Name                 AliasProperty  Name = ProcessName
NPM                  AliasProperty  NPM = NonpagedSystemMemorySize
PM                   AliasProperty  PM = PagedMemorySize
VM                   AliasProperty  VM = VirtualMemorySize
WS                   AliasProperty  WS = WorkingSet
. . .
Responding           Property       System.Boolean Responding {get;}
. . .
```

Здесь мы видим имя .NET-класса, экземпляры которого возвращаются в ходе работы исследуемого командлета (в нашем примере это класс System.Diagnostic.Process), а также полный список элементов объекта (в частности, интересующее нас свойство Responding, определяющего "зависшие" процессы). При этом на экран выводится очень много элементов, просматривать их неудобно. Командлет Get-Member позволяет перечислить только те элементы объекта, которые являются его **свойствами**. Для этого используется параметр MemberType со значением Properties:

```
PS C:\> Get-Process | Get-Member -MemberType Property
   TypeName: System.Diagnostics.Process
Name                   MemberType Definition
----                   ---------- ----------
BasePriority           Property   System.Int32 BasePriority {get;}
EnableRaisingEvents    Property   System.Boolean EnableRaisingEvents...
ExitCode               Property   System.Int32 ExitCode {get;}
ExitTime               Property   System.DateTime ExitTime {get;}
Handle                 Property   System.IntPtr Handle {get;}
HandleCount            Property   System.Int32 HandleCount {get;}
HasExited              Property   System.Boolean HasExited {get;}
Id                     Property   System.Int32 Id {get;}
. . .
```

Responding Property System.Boolean Responding {get;}

. . .

Процессам ОС соответствуют объекты, имеющие очень много свойств, на экран же при работе командлета Get-Process выводятся лишь несколько из них (способы отображения объектов различных типов задаются конфигурационными файлами в формате XML, находящимися в каталоге, где установлен файл powershell.exe).

Рассмотрим наиболее часто используемые операции над элементами конвейера: фильтрации и сортировки.

1.9.3. Фильтрация объектов в конвейере

В PowerShell поддерживается возможность **фильтрации объектов** в конвейере, т.е. удаление из конвейера объектов, не удовлетворяющих определенному условию. Данную функциональность обеспечивает командлет Where-Object, позволяющий проверить каждый объект, находящийся в конвейере, и передать его дальше по конвейеру, только если объект удовлетворяет условиям проверки.

Например, для вывода информации о "зависших" процессах (объекты, возвращаемые командлетом Get-Process, у которых свойство Responding равно False) можно использовать следующий конвейер:

```
Get-Process | Where-Object {-not $_.Responding}
```

Другой пример – оставим в конвейере только те процессы, у которых значение идентификатора (свойство Id) больше 1000:

```
Get-Process | Where-Object {$_.Id -gt 1000}
```

В блоках сценариев командлета Where-Object для обращения к текущему объекту конвейера и извлечения нужных свойств этого объекта используется **специальная переменная** $_, которая создается оболочкой PowerShell автоматически. Данная переменная используется и в других командлетах, производящих обработку элементов конвейера.

Условие проверки в Where-Object задается в виде **блока сценария** – одной или нескольких команд PowerShell, заключенных в фигурные скобки {}. Результатом выполнения данного блока сценария должно быть значение логического типа: True (истина) или False (ложь). Как можно понять из примеров, в блоке сценария используются специальные операторы сравнения.

Замечание. В PowerShell для операторов сравнения не используются обычные символы > или <, так как в командной строке они обычно означают перенаправление ввода/вывода.

Основные операторы сравнения приведены в табл. 1.

Таблица 1. Операторы сравнения в PowerShell

Оператор	Значение	Пример (возвращается значение True)
-eq	равно	10 -eq 10
-ne	не равно	9 -ne 10
-lt	меньше	3 -lt 4
-le	меньше или равно	3 –le 4
-gt	больше	4 -gt 3
-ge	больше или равно	4 -ge 3
-like	сравнение на совпадение с учетом подстановочного знака в тексте	"file.doc" –like "f*.doc"
-notlike	сравнение на несовпадение с учетом подстановочного знака в тексте	"file.doc" –notlike "f*.rtf"
-contains	содержит	1,2,3 –contains 1

-notcontains	не содержит	1,2,3 –notcontains 4

Операторы сравнения можно соединять друг с другом с помощью логических операторов (см. табл. 2). **Таблица 2. Логические операторы в PowerShell**

Оператор	Значение	Пример (возвращается значение True)
-and	логическое И	(10 -eq 10) –and (1 –eq 1)
-or	логическое ИЛИ	(9 -ne 10) –or (3 –eq 4)
-not	логическое НЕ	-not (3 –gt 4)
!	логическое НЕ	!(3 -gt 4)

1.9.4. Сортировка объектов

Сортировка элементов конвейера – еще одна операция, которая часто применяется при конвейерной обработке объектов. Данную операцию осуществляет командлет Sort-Object: ему передаются имена свойств, по которым нужно произвести сортировку, а он возвращает данные, упорядоченные по значениям этих свойств.

Например, для вывода списка запущенных в системе процессов, упорядоченного по затраченному процессорному времени (свойство cpu), можно воспользоваться следующим конвейером:

PS C:\> Get-Process | Sort-Object cpu

Для сортировки в обратном порядке используется параметр Descending:

PS C:\> Get-Process | Sort-Object cpu -Descending

В рассмотренных нами примерах конвейеры состояли из двух командлетов. Это не обязательное условие, конвейер может объединять и большее количество команд, например:

Get-Process | Where-Object {$_.Id -gt 1000} | Sort-Object cpu –Descending

1.9.5. Использование переменных

В переменных хранятся все возможные значения, даже если они являются объектами. Имена переменных в PowerShell всегда должны начинаться с символа «$». Можно сохранить список процессов в переменной, это позволит в любое время получать доступ к списку процессов. Присвоить значение переменной легко:

$a = get-process | sort-object CPU

Вывести содержимое переменной можно, просто напечатав в командной строке $a.

1.9.6. Создание и использование массивов

Для создания и инициализации массива достаточно присвоить значения его элементам. Значения, добавляемые в массив, разделяются запятыми и отделяются от имени массива символом присваивания. Например, следующая команда создаст массив $a из трех элементов:

PS C:\> $a=1,5,7

PS C:\>$a

1

5

7

Можно создать и инициализировать массив, используя оператор диапазона (..). Например, команда

PS C:\> $b=10..15

создает и инициализирует массив $b, содержащий 6 значений 10, 11, 12, 13, 14 и 15.

Для создания массива может использоваться операция ввода значений его элементов из текстового файла:

PS C:\> $f = Get-Content c:\data\numb.txt –TotalCount 25

```
PS C:\>$f.length
25
```

В приведенном примере результат выполнения командлета Get-Content присваивается массиву $f. Необязательный параметр –TotalCount ограничивает количество прочитанных элементов величиной 25. Свойство объекта массив – length – имеет значение, равное количеству элементов массива, в примере оно равно 25 (предполагается, что в текстовом файле munb.txt по крайней мере 25 строк).

1.9.6.1. Обращение к элементам массива

Длина массива (количество элементов) хранится в свойстве Length. Для обращения к определенному элементу массива нужно указать его индекс в квадратных скобках после имени переменной. Нумерация элементов массива **всегда начинается с нуля**. В качестве индекса можно указывать и отрицательные значения, отсчет будет вестись с конца массива – индекс -1 соответствует последнему элементу массива.

1.9.6.2. Операции с массивами

По умолчанию массивы PowerShell могут содержать элементы разных типов (целые 32-х разрядные числа, строки, вещественные и другие), то есть являются полиморфными. Можно создать массив с жестко заданным типом, содержащий элементы только одного типа, указав нужный тип в квадратных скобках перед именем переменной. Например, следующая команда создаст массив 32-х разрядных целых чисел:

```
PS C:\> [int[]]$a=1,2,3
```

Массивы PowerShell базируются на .NET-массивах, имеющих фиксированную длину, поэтому обращение за предел массива фиксируется как ошибка. Имеется способ увеличения первоначально определенной длины массива. Для этого можно воспользоваться оператором конкатенации + или +=. Например, следующая команда добавит к массиву $a два новых элемента со значениями 5 и 6:

```
PS C:\> $a
1
2
3
4
PS C:\>$a+=5,6
PS C:\>$a
1
2
3
4
5
6
```

При выполнении оператора += происходит следующее:

- создается новый массив, размер которого достаточен для помещения в него всех элементов;
- первоначальное содержимое массива копируется в новый массив;
- новые элементы копируются в конец нового массива.

Таким образом, на самом деле создается новый массив большего размера.

Можно объединить два массива, например $b и $c в один с помощью операции конкатенации +. Например:

```
PS C:\> $d=$b+$c
```

1.10. Регулярные выражения – назначение и использование

Регулярные выражения (или сокращенно “регэкспы” (regexp, regular expressions)) обладают огромной мощью, и способны сильно упростить жизнь системного администратора или программиста. В PowerShell регулярные выражения легко доступны, удобны в использовании и максимально функциональны. PowerShell использует реализацию регулярных выражений .NET.

Регулярные выражения - это специальный мини-язык, служащий для разбора (parsing) текстовых данных. С его помощью можно разделять строки на компоненты, выбирать нужные части строк для дальнейшей обработки, производить замены и т. д.

Знакомство с регулярными выражениями начнем с более простой технологии, служащей подобным целям - с **подстановочных символов**. Наверняка вы не раз выполняли команду dir, указывая ей в качестве аргумента маску файла, например *.exe. В данном случае звёздочка означает “любое количество любых символов”. Аналогично можно использовать и знак вопроса, он будет означать “один любой символ”, то есть dir ??.exe выведет все файлы с расширением .exe и именем из двух символов. В PowerShell можно применять и еще одну конструкцию – **группы символов**. Так например [a-f] будет означать “один любой символ от a до f, то есть (a,b,c,d,e,f)”, а [smw] любую из трех букв (s, m или w). Таким образом команда get-childitem [smw]??.exe выведет файлы с расширением .exe, у которых имя состоит из трех букв, и первая буква либо s, либо m, либо w.

1.10.1. Оператор PowerShell -match

Для начала изучения мы будем использовать оператор PowerShell -match, который позволяет сравнивать текст слева от него, с регулярным выражением справа. В случае если текст подпадает под регулярное выражение, оператор выдаёт True, иначе – False.

```
PS C:\> "PowerShell" -match "Power"
True
```

При сравнении с регулярным выражением ищется лишь вхождение строки, полное совпадение текста необязательно (разумеется, это можно изменить). То есть достаточно, чтобы регулярное выражение встречалось в тексте.

```
PS C:\> "Shell" -match "Power"
False
PS C:\> "PowerShell" -match "rsh"
True
```

Еще одна тонкость: оператор -match по умолчанию не чувствителен к регистру символов (как и другие текстовые операторы в PowerShell), если же нужна чувствительность к регистру, используется -cmatch:

```
PS C:\> "PowerShell" -cmatch "rsh"
False
```

1.10.2. Использование групп символов

В регулярных выражениях можно использовать и группы символов:

```
PS C:\> Get-Process | where {$_.name -match "sy[ns]"}

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
------    -----      -----     -----   ------     -- -----------
165      11     2524       8140    79     0,30   5228 mobsync
114      10     3436       3028    83    50,14   3404 SynTPEnh
149      11     2356        492    93     0,06   1592 SynTPStart
810       0      116        380     6            4 System
```

И диапазоны в этих группах:

```
PS C:\> "яблоко","апельсин","груша","абрикос" -match "а[а-п]"
апельсин
```

абрикос

В левой части оператора -match находится массив строк, и оператор соответственно вывел лишь те строки, которые подошли под регулярное выражение.

Перечисления символов можно комбинировать, например группа [агдэ-я] будет означать “А или Г или Д или любой символ от Э до Я включительно”. Но гораздо интереснее использовать диапазоны для определения целых **классов символов**. Например [а-я] будет означать любую букву русского алфавита, а [a-z] английского. Аналогично можно поступать с цифрами – следующая команда выведет все процессы, в именах которых встречаются цифры:

```
PS C:\> Get-Process | where {$_.name -match "[0-9]"}

Handles      NPM(K)      PM(K)      WS(K)   VM(M)   CPU(s)      Id    ProcessName
-------  ------    -----     -----  ----- ------     -- -----------
     57       2     404       1620    16     0,05    984 ati2evxx
    110       4    2540       4868    36     0,20    852 hpgs2wnd
    105       3     940       3292    36     0,19   2424 hpgs2wnf
     91       3    2116       3252    34     0,06    236 rundll32
```

Так как эта группа используется достаточно часто, для неё была выделена специальная последовательность – \d (от слова digit). По смыслу она полностью идентична [0-9], но короче.

```
PS C:\> Get-Process | where {$_.name -match "\d"}

Handles  NPM(K)    PM(K)      WS(K) VM(M)    CPU(s)      Id     ProcessName
---  ------    -----   ----- -----  ------   -- -----------
 93      10     1788    2336    70   1,25    548 FlashUtil10c
158      12     6500    1024    96   0,14   3336 smax4pnp
 30       6      764     160    41   0,02   3920 TabTip32
```

Так же последовательность была выделена для группы “любые буквы любого алфавита, любые цифры, или символ подчеркивания” эта группа обозначается как \w (от word) она примерно эквивалентна конструкции [a-za-я_0-9] (в \w еще входят символы других алфавитов которые используются для написания слов).

Другая популярная группа: \s – “пробел, или другой пробельный символ” (например символ табуляции). Сокращение от слова space. В большинстве случаев вы можете обозначать пробел просто как пробел, но эта конструкция добавляет читабельности регулярному выражению.

Не менее популярной группой можно назвать символ . (точка). Точка в регулярных выражениях аналогична по смыслу знаку вопроса в подстановочных символах, то есть обозначает один любой символ.

Все вышеперечисленные конструкции можно использовать как отдельно, так и в составе групп, например [\s\d] будет соответствовать любой цифре или пробелу. Если вы хотите указать внутри группы символ - (тире/минус) то надо либо экранировать его символом \ (обратный слеш), либо поставить его в начале группы, чтобы он не был случайно истолкован как диапазон:

```
PS C:\> "?????","Word","123","-" -match "[-\d]"
123
-
```

1.10.3. Отрицательные группы и якоря

Рассмотрим некоторые более “продвинутые” конструкции регулярных выражений.

Предполагается, что вы уже знаете, как указать регулярному выражению, какие символы и/или их последовательности должны быть в строке для совпадения. А что если нужно указать не те символы, которые должны присутствовать, а те которых не должно быть? То есть если

нужно вывести лишь согласные буквы, вы можете их перечислить, а можете использовать и отрицательную группу с гласными, например:

```
PS C:\> "a","b","c","d","e","f","g","h" -match "[^aoueyi]"
b
c
d
f
g
h
```

"Крышка" в качестве первого символа группы символов означает именно **отрицание**. То есть на месте группы может присутствовать любой символ кроме перечисленных в ней. Для того чтобы включить отрицание в символьных группах (\d, \w, \s), не обязательно заключать их в квадратные скобки, достаточно перевести их в **верхний регистр**. Например \D будет означать "что угодно, кроме цифр", а \S "всё кроме пробелов"

```
PS C:\> "a","b","1","c","45" -match "\D"
a
b
c
PS C:\> "a","-","*","c","&" -match "\W"
-
*
&
```

Символьные группы позволяют указать лишь содержимое одной позиции, один символ, находящийся в неопределенном месте строки. А что если надо например выбрать все слова которые начинаются с буквы w? Если просто поместить эту букву в регулярное выражение, то оно совпадёт для всех строк, где w вообще встречается, и не важно – в начале, в середине или в конце строки. В таких случаях на помощь приходят "**якоря**". Они позволяют производить сравнение, начиная с определенной позиции в строке.

^ (крышка) является **якорем начала строки**, а $ (знак доллара) - обозначает конец строки.

Не запутайтесь - ^ как символ отрицания используется лишь в начале группы символов, а вне группы - этот символ является уже якорем. Авторам регулярных выражений явно не хватало специальных символов, и они по возможности использовали их более чем в одном месте.

Пример. Вывод списка процессов, имена которых начинаются с буквы w:

```
PS C:\> Get-Process | where {$_.name -match "^w"}

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
 ---  ------    -----      ----- -----   ------     -- -----------
  80      10     1460       156    47     0,11    452 wininit
 114       9     2732      1428    55     0,56   3508 winlogon
 162      11     3660      1652    44     0,14   3620 wisptis
 225      20     5076      4308    95    31,33   3800 wisptis
```

Эта команда вывела процессы, у которых сразу после начала имени (^) следует символ w. Иначе говоря, имя начинается на w. Для усложнения примера, и для упрощения понимания, добавим сюда “крышку” в значении отрицательной группы:

```
PS C:\> Get-Process | where {$_.name -match "^w[^l-z]"}

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -------
```

```
 80     10    1460    156    47     0,11    452 wininit
 114     9    2732   1428    55     0,56   3508 winlogon
 162    11    3660   1652    44     0,14   3620 wisptis
 225    20    5076   4308    95    31,50   3800 wisptis
```

Теперь команда вывела процессы, у которых имя начинается с символа w, а следующий символ является чем угодно, только не символом из диапазона l-z.

Для закрепления опробуем второй якорь – конец строки:

```
PS C:\> "Яблоки","Груши","Дыня","Енот","Апельсины","Персик" -match "[ыи]$"
Яблоки
Груши
Апельсины
```

Это выражение вывело нам все слова в которых последняя буква И или Ы.

Если вы можете точно описать содержимое всей строки, то вы можете использовать и оба якоря одновременно:

```
PS C:\> "abc","adc","aef","bca","aeb","abec","abce" -match "^a.[cb]$"
abc
adc
aeb
```

Это регулярное выражение выводит все строки, которые начинаются с буквы A, за которой следует один любой символ (точка), затем символ C или B и затем конец строки.

Обозначения некоторых классов символов (метасимволы) приведены в табл. 3Таблица 3. Метасимволы, используемые в регулярных выражениях

Метасимвол	Описание метасимвола
.(точка)	Предполагает, что в конечном выражении на ее месте будет стоять любой символ. Продемонстрируем это на примере набора английских слов: **Исходный набор строк:** wake make machine cake maze **Регулярное выражение:** ma.e **Результат:** make maze
\w	Замещает любые символы, которые относятся к буквам, цифрам и знаку подчеркивания. Пример: **Исходный набор строк:** abc a$c a1c a c **Регулярное выражение:** a\wc **Результат:** abc a1c
\W	Замещает все символы, кроме букв, цифр и знака подчеркивания (то есть является обратным метасимволу \w). Пример: **Исходный набор строк:**

	abc a$c a1c a c **Регулярное выражение:** a\Wc **Результат:** a$c a c
\d	Замещает все цифры. Продемонстрируем его действие на том же примере: **Исходный набор строк:** abc a$c a1c a c **Регулярное выражение:** a\dc **Результат:** alc
\D	Замещает все символы, кроме цифр, например: **Исходный набор строк:** abc a$c alc a c **Регулярное выражение:** a\Dc **Результат:** abc a$c a c

1.10.4. Количественные модификаторы (квантификаторы)

Обычно регулярные выражения гораздо сложнее, чем приведенные выше, и записывать их по одному символу было бы тяжеловато. Например, нужно отобрать строки, состоящие из четырех символов, каждый из которых может быть буквой от A до F или цифрой? Регулярное выражение могло бы выглядеть примерно так:

```
PS C:\> "af12","1FE0","1fz1","B009","C1212" -match "^[a-f\d][a-f\d][a-f\d][a-f\d]$"
af12
1FE0
B009
```

Не слишком то лаконично, не правда ли? К счастью всю эту конструкцию можно значительно сократить. Для этого в регулярных выражениях существует специальная конструкция – "**количественные модификаторы**" (квантификаторы). Эти модификаторы приписываются к любой группе справа, и определяют количество вхождений этой группы. Например, количественный модификатор {4} означает 4 вхождения. Посмотрим на приведенном выше примере:

```
PS C:\> "af12","1FE0","1fz1","B009","C1212" -match "^[a-f\d]{4}$"
af12
1FE0
B009
```

Данное регулярное выражение полностью эквивалентно предыдущему – "4 раза по [a-f\d]". Но этот количественный модификатор не обязательно жестко оговаривает количество

повторений. Например можно задать количество как "от 4 до 6". Делается это указанием внутри фигурных скобок двух чисел через запятую – минимума и максимума:

```
PS C:\> "af12","1FE0","1fA999","B009","C1212","A00062","FF00FF9" -match "^[a-f\d]{4,6}$"
af12
1FE0
1fA999
B009
C1212
A00062
```

Если максимальное количество вхождений безразлично, например нужно указать "3 вхождения или больше", то максимум можно просто опустить (оставив запятую на месте), например "строка состоящая из 3х или более цифр":

```
PS C:\> "1","12","123","1234","12345" -match "^\d{3,}$"
123
1234
12345
```

Минимальное значение опустить нельзя, но можно просто указать единицу:

```
PS C:\> "1","12","123","1234","12345" -match "^\d{1,3}$"
1
12
123
```

Как и в случае с символьными группами, для особенно популярных значений количественных модификаторов, есть короткие псевдонимы:

+ (плюс), эквивалентен {1,} то есть, "одно или больше вхождений"

* (звездочка), то же самое что и {0,} или на русском языке – "любое количество вхождений, в том числе и 0"

? (вопросительный знак), равен {0,1} – "либо одно вхождение, либо полное отсутствие вхождений".

В регулярных выражениях, количественные модификаторы **сами по себе** использоваться не могут. Для них обязателен символ или символьная группа, которые и будут определять их смысл. Вот несколько примеров:

.+ Один или более любых символов. Аналог ?* в простых подстановках (как в cmd.exe).

Следующее выражение выбирает процессы, у которых имя "начинается с буквы S, затем следует 1 или более любых символов, затем снова буква S и сразу после неё конец строки". Иначе говоря "имена которые начинаются и заканчиваются на S":

```
PS C:\> Get-Process | where {$_.name -match "^s.+s$"}

Handles  NPM(K)    PM(K)      WS(K)   VM(M)   CPU(s)     Id    ProcessName
------ -----   -----  -----   ------    --   -----------
  257   14     6540    5220    53     5,97  508 services
   30    2      424     128     5     0,08  280 smss
```

\S* Любое количество символов не являющихся пробелами. Подобное выражение может совпасть и с ""(с пустой строкой), ведь под любым количеством подразумевается и ноль, то есть 0 вхождений – тоже результат.

```
PS C:\> "abc", "cab", "a c","ac","abdec" -match "a\S*c"
abc
ac
abdec
```

Заметьте, строка "ac" тоже совпала, хотя между буквами A и C вообще не было символов. Если заменить * на + то будет иначе:

```
PS C:\> "abc", "cab", "a c","ac","abdec" -match "a\S+c"
```

abc
abdec

бобры? (Это не вопрос, а регулярное выражение). Последовательность "бобр", после которой может идти символ "ы", а может и отсутствовать:

```
PS C:\> "бобр","бобры","бобрята" -match "^бобры?$"
бобр
бобры
```

1.10.5. Группы захвата и переменная $matches

Теперь, когда мы можем с помощью регулярных выражений описывать и проверять строки по достаточно сложным правилам, пора познакомится с другой не менее важной возможностью регулярных выражений – "группами захвата" (capture groups). Как следует из названия, группы можно использовать для группировки. К группам захвата, как и к символам и символьным группам, можно применять количественные модификаторы. Например, следующее выражение означает "Первая буква в строке – S, затем одна или больше групп, состоящих из “знака - (минус) и любого количества цифр за ним” до конца строки":

```
PS C:\> "S-1-5-21-1964843605-2840444903-4043112481" -match "^S(-\d+)+$"
True
```

Или:

```
PS C:\> "Ноут","Ноутбук","Лептоп" -match "Ноут(бук)?"
Ноут
Ноутбук
```

Эти примеры показывают, как можно использовать группы захвата для группировки, но это вовсе не главное их качество. Гораздо важнее то, что часть строки, подпавшая под подвыражение, находящееся внутри такой группы, помещается в специальную переменную – $matches. $Matches - это массив, и в нем может находится содержимое нескольких групп. Причем под индексом 0 туда помещается вся совпавшая строка, начиная с единицы идет содержимое групп захвата. Рассмотрим пример:

```
PS C:\> "At 17:04 Firewall service was stopped." -match "(\d\d:\d\d) (\S+)"
True
PS C:\> $matches

Name                           Value
----                           -----
2                              Firewall
1                              17:04
0                              17:04 Firewall
```

Под индексом 0 находится вся часть строки, подпавшая под регулярное выражение, под 1 находится содержимое первых скобок, и под 2 соответственно содержимое вторых скобок. К содержимому $matches можно обращаться как к элементам любого другого массива в PowerShell:

```
PS C:\> $matches[1]
17:04
PS C:\> $matches[2]
Firewall
```

Если в строке присутствует много групп захвата, то бывает полезно дать им имена, это сильно облегчает дальнейшую работу с полученными данными:

```
PS C:\> "At 17:04 Firewall service was stopped." -match "(?<Время>\d\d:\d\d) (?<Служба>\S+)"
True
```

```
PS C:\> $matches
Name                           Value
----                           -----
Время                          17:04
Служба                         Firewall
0                              17:04 Firewall

PS C:\> $matches.Время
17:04
PS C:\> $matches["Служба"]
Firewall
```

Регулярное выражение конечно усложнилось, но зато работать с результатами гораздо приятнее. Синтаксис именования следующий:

(?<Название Группы>подвыражение)

Не перепутайте порядок, сначала следует знак вопроса. Количественные модификаторы, в том числе ? могут применяться только после группы, и следовательно в начале подвыражения – бессмысленны. Поэтому в группах знак вопроса, следующий сразу за открывающей скобкой, означает особый тип группы, в нашем примере – **именованную.**

Другой тип группы, который часто используется – **незахватывающая** группа. Она может пригодиться в тех случаях, когда не нужно захватывать содержимое группы, а надо применить её только для группировки. Например, в вышеприведённом примере с SID, такая группа была бы более уместна:

```
PS C:\> "S-1-5-21-1964843605-2840444903-4043112481" -match "^S(?:-\d+)+$"
True
PS C:\> $matches

Name                           Value
----                           -----
0                              S-1-5-21-1964843605-2840444903-4043112481
```

Синтаксис такой группы: (?:подвыражение). Группы можно и вкладывать одну в другую:

```
PS C:\> "MAC address is '00-19-D2-73-77-6F'." -match "is '([a-f\d]{2}(?:-[a-f\d]{2}){5})'"
True
PS C:\> $matches

Name                           Value
----                           -----
1                              00-19-D2-73-77-6F
0                              is '00-19-D2-73-77-6F'
```

1.11. Управляющие инструкции

1.11.1. Инструкция If …ElseIf … Else

В общем случае синтаксис инструкции If имеет вид

```
If (условие1)
   {блок_кода1}
[ElseIf (условие2)]
   {блок_кода2}]
[Else
   {блок_кода3}]
```

При выполнении инструкции If проверяется истинность условного выражения *условие1*.

Если *условие1* имеет значение $True, то выполняется блок_кода1, после чего выполнение инструкции if завершается. Если *условие1* имеет значение $False, проверяется истинность условного выражения *условие2*. Если *условие2* имеет значение $True, то выполняется блок_кода2 и выполнение инструкции if завершается. Если и *условие1,* и *условие2* имеют значение $False, то выполняется блок_кода3 и выполнение инструкции if завершается.

Пример использования инструкции if в интерактивном режиме работы. Сначала переменной $a присвоим значение 10:

```
PS C:\> $a=10
```

Затем сравним значение переменной с числом 15:

```
PS C:\> If ($a –eq 15) {
>> ‘Значение $a равно 15’
>> }
>> Else {‘ Значение $a не равно 15‘}
>>
Значение $a не равно 15
```

Из приведенного примера видно также, что в оболочке PS в интерактивном режиме можно выполнять инструкции, состоящие из нескольких строк, что полезно при отладке сценариев.

1.11.2. Циклы While и Do … While

Самый простой из циклов PS – цикл While, в котором команды выполняются до тех пор, пока проверяемое условие имеет значение $True. Инструкция While имеет следующий синтаксис:

While (условие) {блок_команд}

Цикл Do … While похож на цикл While, однако условие в нем проверяется не до блока команд, а после: Do {блок_команд} While (условие). Например:

```
PS C:\> $val=0
PS C:\>Do {$val++; $val} While ($val –ne 3)
1
2
3
```

1.11.3. Цикл For

Обычно цикл For применяется для прохождения по массиву и выполнения определенных действий с каждым из его элементов. Синтаксис инструкции For:

For (инициация; условие; повторение) {блок_команд}. Пример

```
PS C:\> For ($i=0; $i –lt 3; $i++) {$i }
0
1
2
```

1.11.4. Цикл ForEach

Инструкция ForEach позволяет последовательно перебирать элементы коллекций. Самый простой тип коллекции – массив. Особенность цикла ForEach состоит в том, что его синтаксис и выполнение зависят от того, где расположена инструкция ForEach: вне конвейера команд или внутри конвейера.

Инструкция ForEach вне конвейера команд

В этом случае синтаксис цикла ForEach имеет вид:

ForEach ($элемент in $коллекция) {блок_команд}

При выполнении цикла ForEach автоматически создается переменная $элемент. Перед каждой итерацией в цикле этой переменной присваивается значение очередного элемента в коллекции. В разделе блок_команд содержатся команды, выполняемые на каждом элементе коллекции. Приведенный ниже цикл ForEach отображает значения элементов массива $lettArr:

```
PS C:\> $lettArr = “a”, “b”,”c”
PS C:\> ForEach ($lett in $lettArr) {Write-Host $lett}
a
b
c
```

Инструкция ForEach может также использоваться совместно с командлетами, возвращающими коллекции элементов. Например:

```
PS C:\> $ln = 0; ForEach ($f in Dir *.txt) {$ln +=$f.length}
```

В примере создается и обнуляется переменная $ln, затем в цикле ForEach с помощью командлета dir формируется коллекция файлов с расширением txt, находящихся в текущем каталоге. Инструкция ForEach перебирает все элементы этой коллекции, на каждом шаге к текущему файлу выполняется обращение с помощью переменной $f. В блоке команд цикла ForEach к текущему значению переменной $ln добавляется значение свойства Length (размер файла) переменной $f. В результате выполнения цикла в переменной $ln будет получен суммарный размер файлов в текущем каталоге, которые имеют расширение txt.

Инструкция ForEach внутри конвейера команд

Если инструкция ForEach появляется внутри конвейера команд, то PS использует псевдоним ForEach, соответствующий командлету ForEach-Object. В этом случае фактически выполняется командлет ForEach-Object и не требуется часть инструкции ($элемент in $коллекция), так как элементы коллекции блоку команд предоставляет предыдущая команда конвейера.

Синтаксис инструкции ForEach внутри конвейера команд имеет вид:

```
команда | ForEach {блок_команд}
```

Рассмотренный выше пример подсчета суммарного размера файлов из текущего каталога для данного варианта инструкции ForEach примет следующий вид:

```
PS C:\> $ln = 0; dir *.txt |  ForEach { $ln +=$_.Length}
```

В приведенном примере специальная переменная $_ используется для обращения к текущему объекту конвейера и извлечения его свойств.

1.12. Управление выводом команд в PowerShell

Рассмотрим, каким образом система формирует строки текста, которые выводятся на экран в результате выполнения той или иной команды (напомним, что командлеты PowerShell возвращают .NET-объекты, которые, как правило, не знают, каким образом отображать себя на экране).

В PowerShell имеется база данных (набор XML-файлов), содержащая модули форматирования по умолчанию для различных типов .NET-объектов. Эти модули определяют, какие свойства объекта отображаются при выводе и в каком формате: списка или таблицы. Когда объект достигает конца конвейера, PowerShell определяет его тип и ищет его в списке объектов, для которых определено правило форматирования. Если данный тип в списке обнаружен, то к объекту применяется соответствующий модуль форматирования; если нет, то PowerShell просто отображает свойства этого .NET-объекта.

Также в PowerShell можно явно задавать правила форматирования данных, выводимых командлетами, и подобно командному интерпретатору Cmd.exe перенаправлять эти данные в файл, на принтер или в пустое устройство.

1.12.1. Форматирование выводимой информации

В традиционных оболочках команды и утилиты сами форматируют выводимые данные. Некоторые команды (например, dir в интерпретаторе Cmd.exe) позволяют настраивать формат вывода с помощью специальных параметров.

В оболочке PowerShell вывод форматируют только четыре специальных командлета Format (табл. 4). Это упрощает изучение, так как не нужно запоминать средства и параметры форматирования для других команд (остальные командлеты вывод не форматируют).

Таблица 4. Командлеты PowerShell для форматирования вывода

Командлет	**Описание**
Format-Table	Форматирует вывод команды в виде таблицы, столбцы которой содержат свойства объекта (также могут быть добавлены вычисляемые столбцы). Поддерживается возможность группировки выводимых данных
Format-List	Вывод форматируется как список свойств, в котором каждое свойство отображается на новой строке. Поддерживается возможность группировки выводимых данных
Format-Custom	Для форматирования вывода используется пользовательское представление (view)
Format-Wide	Форматирует объекты в виде широкой таблицы, в которой отображается только одно свойство каждого объекта

Как уже отмечалось, если ни один из командлетов Format явно не указан, то используется модуль форматирования по умолчанию, который определяется по типу отображаемых данных. Например, при выполнении командлета Get-Service данные по умолчанию выводятся как таблица с тремя столбцами (Status, Name и DisplayName):

```
PS C:\> Get-Service

Status   Name               DisplayName
------   ----               -----------
Stopped  Alerter            Оповещатель
Running  ALG                Служба шлюза уровня приложения
Stopped  AppMgmt            Управление приложениями
Stopped  aspnet_state       ASP.NET State Service
Running  Ati HotKey Poller  Ati HotKey Poller
Running  AudioSrv           Windows Audio
Running  BITS               Фоновая интеллектуальная служба пер...
Running  Browser            Обозреватель компьютеров
Stopped  cisvc              Служба индексирования
Stopped  ClipSrv            Сервер папки обмена
Stopped  clr_optimizatio... .NET Runtime Optimization Service v...
Stopped  COMSysApp          Системное приложение COM+
Running  CryptSvc           Службы криптографии
Running  DcomLaunch         Запуск серверных процессов DCOM
Running  Dhcp               DHCP-клиент
...
```

Для изменения формата выводимых данных нужно направить их по конвейеру соответствующему командлету Format. Например, следующая команда выведет список служб с помощью командлета Format-List:

```
PS C:\> Get-Service | Format-List
```

```
Name                : Alerter
DisplayName         : Оповещатель
Status              : Stopped
DependentServices   : {}
ServicesDependedOn  : {LanmanWorkstation}
CanPauseAndContinue : False
CanShutdown         : False
CanStop             : False
ServiceType         : Win32ShareProcess

Name                : ALG
DisplayName         : Служба шлюза уровня приложения
Status              : Running
DependentServices   : {}
ServicesDependedOn  : {}
CanPauseAndContinue : False
CanShutdown         : False
CanStop             : True
ServiceType         : Win32OwnProcess
. . .
```

При использовании формата списка выводится больше сведений о каждой службе, чем в формате таблицы (вместо трех столбцов данных о каждой службе в формате списка выводятся девять строк данных). Однако это вовсе не означает, что командлет Format-List извлекает дополнительные сведения о службах. Эти данные содержатся в объектах, возвращаемых командлетом Get-Service, однако командлет Format-Table, используемый по умолчанию, отбрасывает их, потому что не может вывести на экран более трех столбцов.

При форматировании вывода с помощью командлетов Format-List и Format-Table можно указывать имена свойства объекта, которые должны быть отображены (напомним, что просмотреть список свойств, имеющихся у объекта, позволяет рассмотренный ранее командлет Get-Member). Например:

```
PS C:\> Get-Service | Format-List Name, Status, CanStop

Name    : Alerter
Status  : Stopped
CanStop : False

Name    : ALG
Status  : Running
CanStop : True

Name    : AppMgmt
Status  : Stopped
CanStop : False
. . .
```

Вывести все имеющиеся у объектов свойства можно с помощью параметра *, например:

```
PS C:\> Get-Service | Format-table *
```

1.12.2. Перенаправление выводимой информации

В оболочке PowerShell имеются несколько командлетов, с помощью которых можно управлять выводом данных. Эти командлеты начинаются со слова Out, их список можно получить с помощью команлета:

```
PS C:\> Get-Command out-* | Format-Table Name

Name
----
Out-Default
Out-File
Out-Host
Out-Null
Out-Printer
Out-String
```

По умолчанию выводимая информация передается командлету Out-Default, который, в свою очередь, делегирует всю работу по выводу строк на экран командлету Out-Host. Для понимания данного механизма нужно учитывать, что архитектура PowerShell подразумевает различие между собственно ядром оболочки (интерпретатором команд) и главным приложением (host), которое использует это ядро. В принципе, в качестве главного может выступать любое приложение, в котором реализован ряд специальных интерфейсов, позволяющих корректно интерпретировать получаемую от PowerShell информацию. В нашем случае главным приложением является консольное окно, в котором мы работаем с оболочкой, и командлет Out-Host передает выводимую информацию в это консольное окно.

Параметр Paging командлета Out-Host, подобно команде more интерпретатора Cmd.exe, позволяет организовать постраничный вывод информации, например:

```
Get-Help Get-Process –Full | Out-Host -Paging
```

1.12.3. Сохранение данных в файл

Командлет Out-File позволяет направить выводимые данные вместо окна консоли в текстовый файл. Аналогичную задачу решает оператор перенаправления (>), однако командлет Out-File имеет несколько дополнительных параметров, с помощью которых можно более гибко управлять выводом: задавать тип кодировки файла (параметр Encoding), задавать длину выводимых строк в знаках (параметр Width), выбирать режим перезаписи файла (параметр Append). Например, следующая команда направит информацию о выполняющихся на компьютере процессах в файл C:\Process.txt, причем данный файл будет записан в формате ASCII:

```
Get-Process | Out-File -FilePath C:\Process.txt -Encoding ASCII
```

1.12.4. Подавление вывода

Командлет Out-Null служит для поглощения любых своих входных данных. Это может пригодиться для подавления вывода на экран ненужных сведений, полученных в качестве побочного эффекта выполнения какой-либо команды. Например, при создании каталога командой mkdir на экран выводится его содержимое:

```
PS C:\> mkdir spo


    Каталог: Microsoft.PowerShell.Core\FileSystem::C:\
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----           03.01.2009 1:01          spo
```

Если эта информация не нужна, то результат выполнения команды mkdir необходимо передать по конвейеру командлету Out-Null:

```
mkdir spo | Out-Null
```

1.12.5. Преобразование данных в формат html, сохранение в файле и просмотр результатов

Для преобразования данных в формат html служит командлет Convertto-html. Параметр Property определяет свойства объектов, включаемые в выходной документ. Например, для получения списка выполняемых процессов в формате html, включающего имя процесса и затраченное время CPU и записи результата в файл processes.html можно использовать команду

Get-Process | Convertto-html -Property Name, CPU > Processes.htm

Для просмотра содержимого файла можно использовать командлет

Invoke-Item "имя документа"

Например Invoke-Item "processes.htm"

1.12.6. Инвентаризация и диагностика Windows-компьютеров

Для вывода сведений о процессоре ПК служит командлет Get-wmiobject

Get-wmiobject -Class Win32_Processor | Format-list *

1.12.7. Командлеты для измерения свойств объектов

Для иэмерения времени выполнения командлетов PS служит командлет Measure-Command

В качестве примера рассмотрим получение времени выполнение командлета dir

(Measure-Command {dir}).TotalSeconds

Для получения статистических данных служит командлет Measure-Object. Для числовых массивов с его помощью можно получить максимальное, минимальное, среднее значение элементов массива и их сумму. Если имеется инициализированный массив ms, для указанной цели используется командлет

$ms | measure-object –maximum –minimum –average -sum

3. МЕТОДИКА ВЫПОЛНЕНИЯ

12. Ознакомиться с теоретическими сведениями.
13. Запустить оболочку PowerShell.
14. Увеличить ширину окна оболочки до максимальной, увеличить высоту окна и задать цвет фона и цвет шрифта (рекомендуется синий фон и белый шрифт).
15. Вывести содержимое каталога Windows (для бригад 5 и 10 – и подкаталогов) по указанному в табл. 5 формату на экран и в текстовый файл.

Таблица 5. Варианты заданий

Номера	Что выводить (имена, размер, дата создания, атрибуты)	Сортировать по	Условие отбора
1, 6	Только файлы	По размеру	Размер > 10000
2, 7	Файлы и подкаталоги	По дате	Первые буквы имени SY
3, 8	Только подкаталоги	Именам	Последняя буква имени S или T
4, 9	Только файлы bmp	По размеру	Размер >50000
5, 10	Только файлы jpg	Именам	Любые

Рекомендуется использовать фильтр по Extension или Attributes (в зависимости от варианта задания)

16. Вывести в текстовый файл список свойств процесса, возвращаемый командлетом Get-process и на экран – их общее количество.
17. Создать текстовый файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в табл.6 параметра. Имена параметров процессов указаны в табл. 6.

Таблица 6. Варианты заданий

Номера	Список выводимых параметров процессов	Сортировать по значению параметра	Вывести процессы, у которых
1, 4	Имя процесса, BasePriority, Company	Имя процесса	BasePriority > 7
2, 6	Id, Имя процесса, время старта, Handles	Время старта	Id > 40
3, 5	Имя процесса, Id , PriorityClass, UserprocessorTime, TotalProcessorTime	TotalProcessorTime	Id > 100
7, 8	Имя процесса, PriorityClass, ProductVersion, Id	Имя процесса	Id > 100
9, 10	Id, Имя процесса, WorkingSet, CPU	Id	CPU > 5

18. Создать HTML-файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в табл.5 параметра. Имена параметров процессов указаны в табл. 5.

19. Найти суммарный объем всех графических файлов (bmp, jpg), находящихся в каталоге Windows и всех его подкаталогах.

20. Вывести на экран сведения о ЦП компьютера.

21. Найти максимальное, минимальное и среднее значение времени выполнение командлетов dir и ps

11. Выполнить индивидуальные задания для студентов согласно табл. 7.

Таблица 7. Варианты заданий

№	Содержание задания – разработать командлет для:
1	1. вычисления факториала от целочисленной переменной с именем numb 2. нахождения минимального и максимального значений чисел, хранящихся в файле nn.txt
2	1. нахождения количества различных чисел, хранящихся в файле nn.txt 2. нахождения количества наибольших чисел, хранящихся в файле nn.txt
3	1. нахождения количества положительных чисел, хранящихся в файле nn.txt 2. нахождения количества четных чисел, хранящихся в файле nn.txt
4	1. нахождения в заданном каталоге файла наибольшего размера 2. нахождения в заданном каталоге трех файлов наименьшего размера
5	1. нахождения среди выполняющихся процессов имен процессов, выполняющихся в двух или более экземплярах 2. нахождения среди выполняющихся процессов имени процесса, запущенного последним
6	1. нахождения среди выполняющихся процессов имен трех процессов, использовавших более всего процессорного времени 2. нахождения среди выполняющихся процессов имени процесса с наибольшим размером рабочего множества страниц
7	1. нахождения среди выполняющихся процессов имен процессов с наименьшим значением BasePriority 2. нахождения среди выполняющихся процессов имен процессов, у которых значения параметра WorkingSet одинаковы
8	1. проверки наличия в текущем каталоге файлов одинакового размера. Если такие файлы есть – вывести их имена 2. нахождения среди выполняющихся процессов имен процессов с наибольшим значением приоритета
9	1.нахождения в каталоге windows\system32 имен трех dll наибольшего и

	наименьшего размеров 2. нахождения в каталоге windows\system32 имен трех dll с самой ранней датой создания
10	1. нахождения среди выполняющихся процессов имен трех процессов, работающих в системе дольше всего 2. нахождения среди выполняющихся процессов имен процессов, имеющих одинаковые ProductVersion
11	1. разбиения текстового файла, содержащего четное количество строк, на два текстовых файла, в каждый из которых записать одинаковое количество строк 2. нахождения в каталоге windows и его подкаталогах имен библиотек dll из шести символов, начинающихся на mfc и заканчивающихся буквой u
12	1. нахождения в каталоге windows и его подкаталогах имен файлов, записанных русскими буквами и имеющих расширение jpg. 2. нахождения в текстовом файле, содержащем слова английского и русского языков (одно слово в строке), слов русского языка и вывода их на экран
13	1. нахождения в текстовом файле, содержащем слова английского и русского языков и числа (одно слово или число в строке), чисел и вывода их на экран
14	1. нахождения в текстовом файле, содержащем слова английского и русского языков (одно слово в строке), слов, написанных с заглавной буквы и вывода их на экран

5. ОТЧЕТ О РАБОТЕ

Содержание отчета:

1. Тексты командлетов, использованных при выполнении заданий 4 - 11.
2. Результаты, полученные при выполнении заданий 5, 6, 8, 9, 10, 11.
3. Письменный ответ на контрольные вопросы.

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

16. Типы команд PowerShell (PS).
17. Имена и структура командлетов.
18. Псевдонимы команд.
19. Просмотр структуры объектов.
20. Фильтрация объектов в конвейере. Блок сценария.
21. Какую информацию выводит команда Get-Help * ?
22. Командлеты для форматирования выводимой информации.
23. Перенаправление выводимой информации.
24. Управляющие инструкции PS.
25. Назначение регулярных выражений.
26. Сохранение данных в текстовом файле и html-файле.
27. Получение справочной информации в PS.
28. Как создать массив в PS?
29. Как объединить два массива?
30. Как увеличить размер созданного в PS массива?
31. Как ввести данные в массив?
32. Использование командлета Out-Null.
33. Оператор PowerShell –match.
34. Использование символа ^ в командлетах.
35. Использование символа $ в командлетах.
36. Количественные модификаторы (квантификаторы).
37. Использование групп захвата.
38. Командлеты для измерения свойств объектов.

Лабораторная работа № 4. Исследование и мониторинг производительности ос windows с помощью системного монитора (4 часа)

Цель работы: практическое знакомство с методикой использования системного монитора (монитора производительности) perfmon для поиска узких мест в вычислительной системе

1.1. Мониторинг производительности ОС с помощью системного монитора

Цель мониторинга работы ОС – поиск узких мест в системе, обусловленных нехваткой ресурсов – аппаратных или информационных. В качестве исходных данных для анализа узких мест могут использоваться данные, получаемые со счетчиков производительности.

Счетчики производительности. Семейство операционных систем MS Windows (Windows NT 4.0, Windows 2000, Windows XP, Windows Vista, Windows 7) получает информацию о производительности от аппаратных и программных компонентов компьютера. Системные компоненты (драйверы режима ядра) в ходе своей работы генерируют данные о производительности. Такие компоненты называются *объектами производительности.* В ОС имеется ряд объектов производительности, обычно соответствующих аппаратным компонентам, таким как память, процессоры, внешние устройства и т. д.

Каждый объект производительности предоставляет счетчики, которые собирают данные производительности (**performance counters**). Счетчик производительности представляет собой механизм, с помощью которого в MS Windows производится сбор сведений о производительности различных системных ресурсов. В MS Windows имеется предопределенный набор счетчиков производительности, с которыми можно взаимодействовать — некоторые из этих счетчиков присутствуют на всех компьютерах с установленной ОС Windows, а некоторые относятся к определенным приложениям и имеются только на некоторых компьютерах. Каждый счетчик относится к определенной области функций системы. В качестве примера можно привести счетчики, следящие за загрузкой процессора, использованием памяти и количеством полученных или переданных по сети байтов. Экземпляр компонента PerformanceCounter можно использовать для непосредственного подключения к существующим счетчикам производительности и для динамического взаимодействия с данными этих счетчиков.

Счетчик производительности следит за поведением объектов производительности компьютера. Эти объекты включают в себя физические компоненты, такие как процессоры, диски, память и системные объекты, такие как процессы, потоки и задания. Системные счетчики, относящиеся к одному и тому же объекту производительности, группируются в категории, отражающие их общую направленность. При создании экземпляра компонента PerformanceCounter сначала указывается категория, с которой будет взаимодействовать компонент, затем внутри этой категории выбирается счетчик, с которым будет осуществляться взаимодействие.

Примером категории счетчиков производительности в Windows является категория «Память». Системные счетчики в этой категории отслеживают такие данные, как количество доступных и кэшируемых байтов. Чтобы узнать в приложении количество кэшируемых байтов, нужно создать экземпляр компонента PerformanceCounter и связать его с категорией «Память», а затем выбрать в этой категории соответствующий счетчик (в данном случае счетчик кэшируемых байтов).

Некоторые объекты (такие как Память и Сервер) имеют только один экземпляр, другие объекты производительности могут иметь множество экземпляров. Если объект имеет множество экземпляров, то можно добавить счетчики для отслеживания статистики по каждому экземпляру или для всех экземпляров одновременно.

Например, если в системе установлены несколько процессоров, или процессор имеет несколько ядер, то объект Процессор будет иметь множество экземпляров. В случае, если объект поддерживает множество экземпляров, то при объединении экземпляров в группу появятся родительский экземпляр и дочерние экземпляры, которые будут принадлежать данному родительскому экземпляру.

В счетчиках производительности сохраняются данные о различных частях системы. Эти значения не запоминаются как записи, но они сохраняются, пока для заданной категории дескриптор остается открытым в памяти. Процесс извлечения данных из счетчика производительности называется получением выборки данных. При получении выборки происходит извлечение непосредственного или рассчитанного значения счетчика.

В зависимости от определения счетчика это значение может соответствовать текущему использованию ресурса (мгновенное значение) или может быть средним значением двух измерений за период времени между выборками. Например, при извлечении значения счетчика потоков из категории Process для конкретного процесса извлекается число потоков на момент последнего измерения. Полученная величина является мгновенным значением. Тем не менее, при извлечении значения счетчика Pages/Sec категории Memory извлекается значение в секундах, которое вычисляется на основе среднего числа страниц, полученных между двумя последними выборками.

Использование ресурсов может сильно изменяться в зависимости от работы в различное время дня. Поэтому счетчики производительности, отражающие процент использования ресурсов за интервал, являются более информативным средством измерения, чем вычисление среднего на основе мгновенных значений счетчиков. Средние значения могут включать в себя данные, соответствующие запуску службы или другим событиям, что на короткий период приведет к выходу значений далеко за пределы диапазона, и, следовательно, к искажению результатов.

Для работы со счетчиками производительности используется встроенная в ОС Windows (NT, 2000, XP, Vista, 7) программа Perfomance Monitor (perfmon.exe). Она не представлена в Главном меню, но ее всегда можно запустить посредством команды “Выполнить”, далее в строке набрать perfmon.exe. В ОС MS Vista используется меню Поиск, в строке поиска вводится имя запускаемого приложения. Для добавления счетчиков необходимо вызвать правой кнопкой мыши контекстное меню на поле графиков (рис. 1), выбрать объект, счетчик, экземпляры счетчика и нажать кнопку “Добавить”.

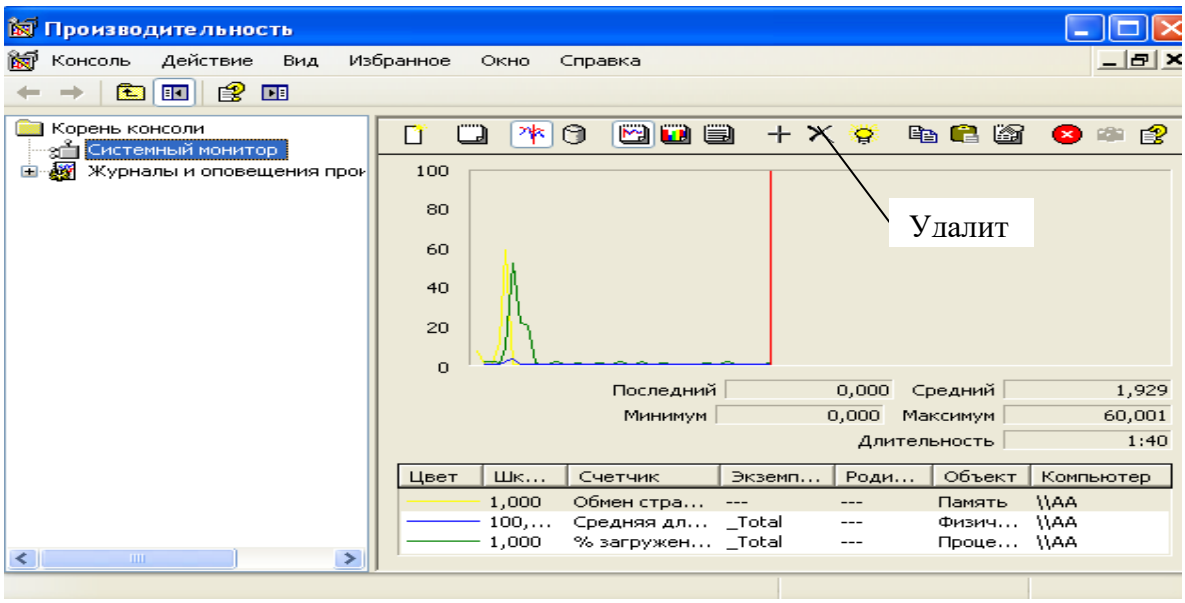


Рисунок 1. Внешний вид программы Perfomance Monitor в MS Windows

В качестве примера рассмотрим последовательность действий при построения графика зависимости размера рабочего множества страниц процесса Блокнот (Notepad) от времени.

1.Запустить Блокнот.

2.Запустить системный монитор perfmon.

3.Используя кнопку Удалить (рис. 1), очистить окно вывода и перечень выводимых графиков.

4.Правой кнопкой мыши вызвать контекстное меню, выбрать Пункт Добавить счетчики.

5.В окне Добавить счетчики (рис.2) выбрать из списка Объект категорию Процесс, далее из списка процессов выбрать процесс notepad, выбрать счетчик Рабочее множество из списка счетчиков – рис. 2.

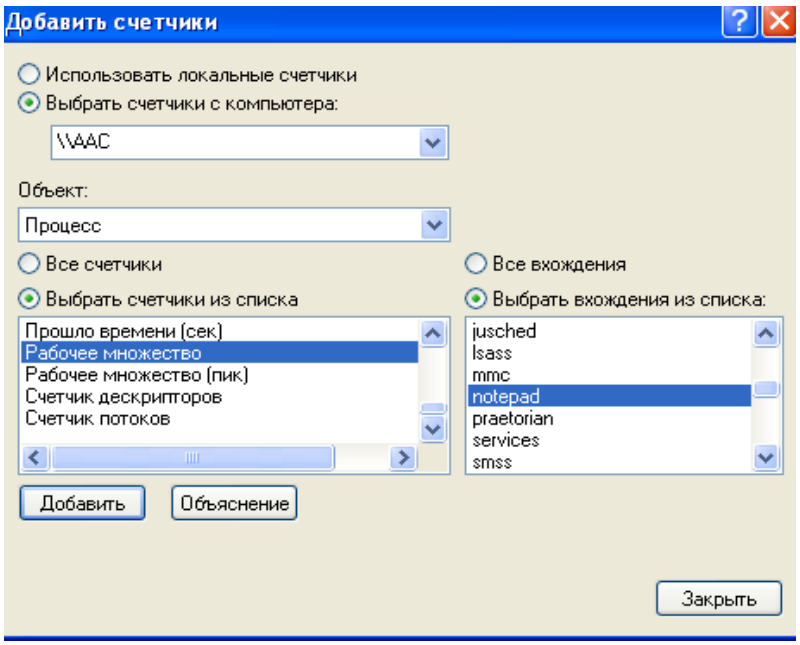


Рисунок 2. Добавление нового счетчика

6. Нажать кнопки Добавить и Закрыть.

Примечание. Для просмотра пояснений о том, какие данные предоставляет конкретный счетчик, используется кнопка Объяснение в диалоговом окне Добавить счетчики (рис. 2).

Управление формой представления графиков производится с помощью окна свойств, которое открывается с помощью кнопки Свойства.

Диапазон значений вертикальной шкалы задается в окне Свойства: системный монитор см. рис. 3.

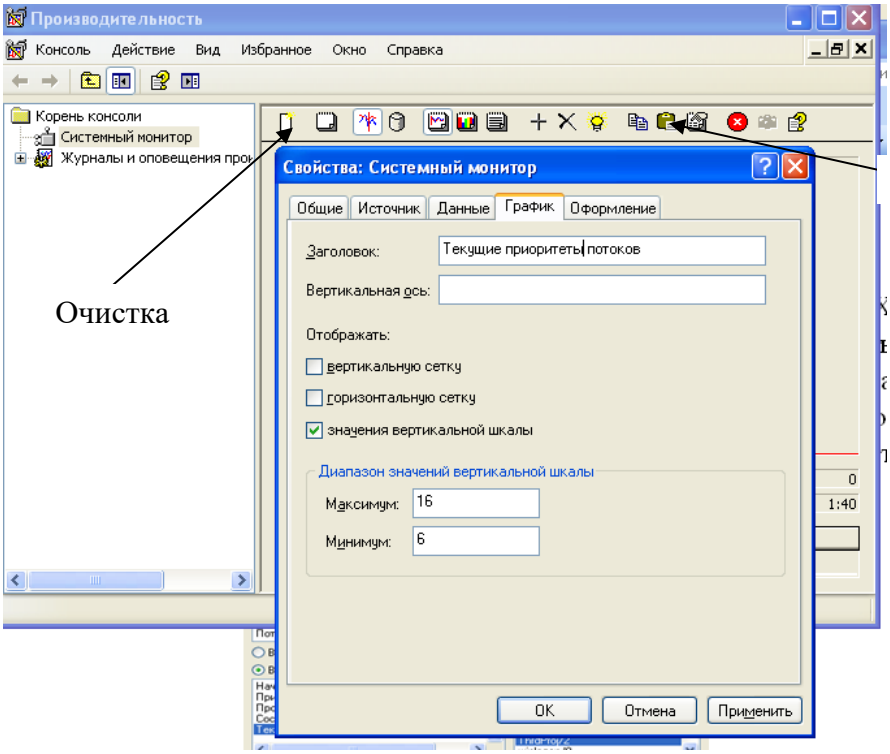


Рисунок 3. Окно Свойства: системный монитор, закладка График

В окне Свойства необходимо задать максимальное и минимальное значения вертикальной шкалы и нажать кнопку Применить.

На рис. 4 показан полученный график изменения рабочего множества программы notepad в процессе создания текстового файла.

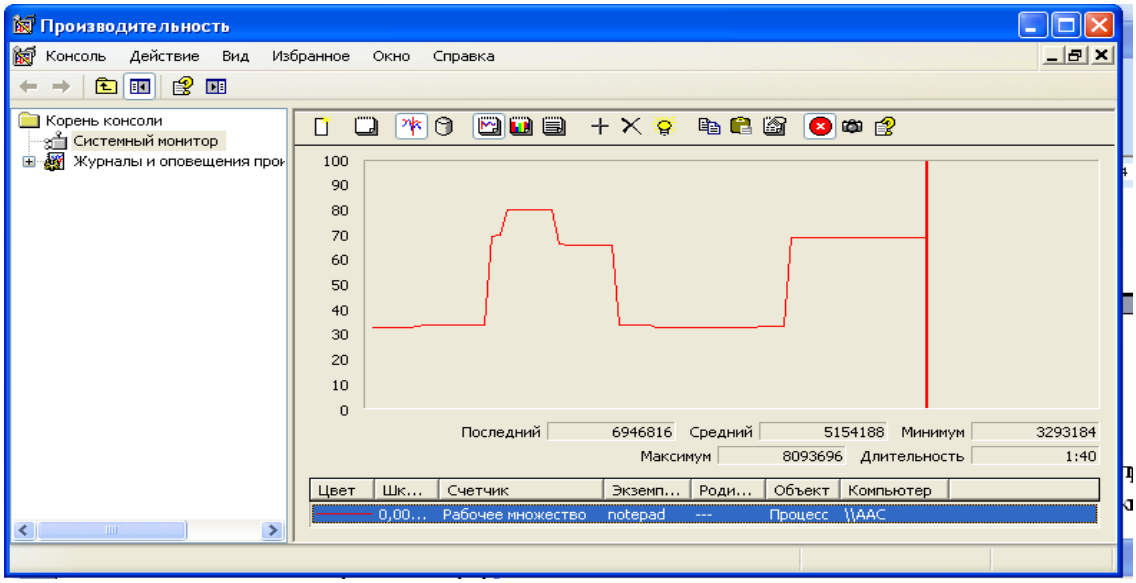


Рисунок 4 График изменения рабочего множества процесса notepad при создании файла

2. МЕТОДИКА ВЫПОЛНЕНИЯ

1. Построить графики изменения количества потоков приложений Notepad и Open Office при создании документа, содержащего текст из одного слова.
2. Для приложения Калькулятор построить 2-3 наиболее динамично изменяющихся графика

изменения текущего приоритета потоков при вычислении значения арифметического выражения, перемещении калькулятора по экрану, перемещении курсора мыши по экрану в области окна калькулятора.

3. Для приложения Open Office построить график изменения объема используемого файла подкачки при последовательном открытии 3-4 файлов увеличивающегося размера.
4. Выполнить индивидуальные задания для студентов согласно табл. 1

Таблица 1-Индивидуальные задания для студентов

№№	Задание
1, 3	Для программы Проводник построить графики изменения количества потоков в процессе запуска приложения
2, 4	Показать характер изменения во времени общего количества выполняющихся с системе потоков
5, 7,	Для каждого ядра процессора выяснить, в каком режиме ядро работает больше времени – пользовательском или системном
6, 8,	Для каждого ядра процессора выяснить, сколько процентов времени ядро выполняет обработку прерываний.
9,11	Для игры Сапер построить графики изменения количества потоков в процессе запуска приложения
10,12	Для программы Paint построить графики изменения количества потоков в процессе запуска приложения
13,15	Для каждого ядра процессора выяснить, в каком режиме ядро работает больше времени – пользовательском или системном
14	Для каждого ядра процессора выяснить, сколько процентов времени ядро выполняет обработку прерываний.

3. ОТЧЕТ О РАБОТЕ

Готовится в письменном виде с помощью текстового процессора.
Содержание отчета:

1.Результаты, полученные при выполнении заданий 1 - 3.
2.Результаты, полученные при выполнении индивидуальных заданий.
3.Выводы по работе.

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение счетчиков производительности.
2. Категории и экземпляры счетчиков.
3. Управление параметрами создаваемых графиков (масштаб, цвет и толщина линий).
4. Влияние активности окна приложения на текущий приоритет его потоков.

Лабораторная работа № 5. Симметричная мультипроцессорная обработка (4 часа)

Цель работы – знакомство с особенностями многопоточной обработки информации на многоядерных процессорах под управлением ОС MS Windows и методом оценки трудоемкости алгоритмов

1.1. Мультипроцессорная и симметричная мультипроцессорная обработка (SMP)

Мультипроцессорная обработка - способ организации вычислительного процесса в системах с несколькими процессорами (ядрами процессора), при котором несколько потоков могут одновременно выполняться на разных процессорах (ядрах) системы.

Мультипроцессорные системы часто характеризуют либо как симметричные, либо как несимметричные. При этом нужно определять, к какому аспекту мультипроцессорной системы относится эта характеристика – к типу архитектуры или к способу организации вычислительного процесса.

Симметричная архитектура мультипроцессорной системы предполагает однородность всех процессоров и единообразие включения процессоров в общую схему мультипроцессорной системы. Традиционные симметричные мультипроцессорные конфигурации разделяют общую оперативную память между всеми процессорами (ядрами процессоров).

Масштабируемость (возможность наращивания числа процессоров) в симметричных системах ограничена вследствие того, что все они пользуются одной оперативной памятью и должны располагаться в одном корпусе. Такая конструкция, называемая *масштабируемой по вертикали*.

В симметричных архитектурах обеспечивается достаточно высокая производительность для тех приложений, в которых несколько задач должны активно взаимодействовать между собой.

В асимметричной архитектуре разные процессоры могут отличаться как своими характеристиками, так и функциональной ролью, которая поручается им в системе. Функциональная неоднородность в асимметричных архитектурах влечет за собой структурные отличия во фрагментах системы, содержащих разные процессоры системы. Масштабирование в асимметричной архитектуре реализуется иначе, чем в симметричной. Так как требование единого корпуса отсутствует, система может состоять из нескольких устройств, каждое из которых содержит один или несколько процессоров. Это масштабирование по *горизонтали.* Каждое такое устройство называется *кластером*, а вся мультипроцессорная система - кластерной.

Асимметричное мультипроцессирование является наиболее простым способом организации вычислительного процесса в системах с несколькими процессорами. Этот способ часто называют также «ведущий-ведомый». Асимметричная организация вычислительного процесса может быть реализована как для симметричной мультипроцессорной архитектуры, так и для несимметричной.

Симметричное мультипроцессирование как способ организации вычислительного процесса может быть реализовано в системах только с симметричной мультипроцессорной архитектурой. Симметричное мультипроцессирование реализуется общей для всех процессоров операционной системой. При симметричной организации все процессоры равноправно участвуют и в управлении вычислительным процессом, и в выполнении прикладных задач.

Наиболее распространенной целью объединения процессоров является симметричная мультипроцессорная обработка (SMP). В системе SMP каждый процессор решает свою задачу, порученную ему операционной системой. В документации Intel симметрия рассматривается в двух аспектах:

- симметрия памяти – все процессоры пользуются общей памятью и одной копией ОС;
- симметрия ввода-вывода – все процессоры разделяют общие устройства ввода-вывода и общие контроллеры прерываний.

Система может быть симметричной по памяти, но асимметричной по прерываниям от ввода-вывода, если для обслуживания этих прерываний выделяется собственный процессор. В x86 симметрию по прерываниям обеспечивают контроллеры APIC. Аппаратная (физическая) реализация SMP может быть различной:

- объединение нескольких физических процессоров на одной локальной шине – процессоры Pentium, P6, Pentium 4;
- размещение на одном кристалле нескольких логических процессоров с разделяемыми операционными блоками – «гиперпотоковые» (hyperthreading) модели Pentium 4;

- размещение на одном кристалле нескольких независимых процессорных ядер с разделяемым вторичным кэшем – мультиядерные модели Pentium 4.

Применение SMP требует поддержки со стороны BIOS, ОС и приложений (чтобы работать быстрее, они должны быть многопоточными). Поддержку SMP имеют такие ОС, как Microsoft Windows NT/2000x/XP/Vista/7 и различные диалекты Unix и Linux.

Несколько лет назад цена мультипроцессорных версий ОС, как правило, была значительно выше цены соответствующих однопроцессорных версий, что становилось препятствием к применению гиперпотоковых и мультиядерных процессоров в системных платах рабочих станций. Теперь число процессоров, на которое лицензируется ОС, соответствует числу физических процессоров. Это открывает возможности широкого распараллеливания на уровне процессоров.

1.2. Оценка достигаемого выигрыша в производительности – закон Амдала

Для количественной оценки выигрыша в производительности ПК при параллельной работе нескольких ядер обычно используется закон Дж. Амдала (1967 г).

Закон Амдала описывает максимальный теоретический выигрыш в производительности параллельного решения по отношению к лучшему последовательному решению [1]

$$V = \frac{1}{S + (1 - S)/n}$$

В данном уравнении V – выигрыш в производительности при использовании n ядер центрального процессора, S – время, потраченное на выполнение последовательной части параллельной версии.

При n=1 (одно ядро) ускорения нет. Если используется два ядра, которые половину всей работы выполняют параллельно, S=0,5 и V= 2 / 1,5 = 1,33. В случае выполнения всей работы двумя ядрами параллельно максимально возможный теоретический выигрыш равен 2.

1.3. Оценка трудоемкости алгоритма

Цель анализа трудоёмкости алгоритма – нахождение оптимального алгоритма для решения задачи. В качестве критерия оптимальности алгоритма выбирается трудоемкость алгоритма, определяемая как количество операций, которые необходимо выполнить для решения задачи с помощью данного алгоритма. Функцией трудоемкости называется соотношение, связывающее размер данные алгоритма с количеством элементарных операций, необходимых для получения решения задачи с помощью данного алгоритма.

Трудоёмкость алгоритмов по-разному зависит от входных данных. Для некоторых алгоритмов трудоемкость зависит только от объёма данных, для других алгоритмов — от значений данных. Трудоёмкость многих алгоритмов может в той или иной мере зависеть от всех перечисленных выше факторов. Одним из упрощенных методов анализа, используемых на практике, является *асимптотический анализ трудоемкости алгоритмов*. Целью анализа является сравнение затрат времени различными алгоритмами, предназначенными для решения одной и той же задачи, при больших объёмах входных данных. Используемая в асимптотическом анализе оценка функции трудоёмкости, называемая **сложностью алгоритма**, позволяет определить, как быстро растет трудоёмкость алгоритма с увеличением объёма данных.

Например, трудоемкость алгоритма сложения векторов A(n) и B(n) равна O(n), потому что количество операций сложения равно количеству элементов векторов. Трудоемкость алгоритма умножения квадратных матриц равна $O(n^3)$. Реализующая алгоритм программа умножения матриц содержит 3 вложенных арифметических цикла.

4. МЕТОДИКА ВЫПОЛНЕНИЯ

1. В работе оценивается трудоемкость простейших алгоритмов и эффективность их параллельного выполнения на многоядерных процессорах под управлением ОС MS Windows 7,

поддерживающей SMP. Для оценки трудоемкости применяется оценка времени выполнения реализующей алгоритм программы на одном или нескольких ядрах ЦП.

2. Для управления количеством используемых процессорных ядер используется диспетчер задач (контекстное меню, пункт *задать соответствие*). Каждая программа должна быть многократно запущена на одном, двух, трех, четырех и более (сколько есть у ЦП) ядрах.

3. Так как время выполнения программы в многозадачной ОС MS Windows зависит от нескольких факторов, для оценки времени следует выполнить программу 5-7 раз с неизменными начальными условиями и в качестве оценки времени выполнения выбрать **наименьшее значение**.

4. Размер обрабатываемого массива следует задавать в пределах 100-500, при этом время выполнения приложения не должно быть менее 500 мсек. Количество различных значений должно лежать в пределах от 4 до 10.

5. Наименование программы, количество используемых ядер ЦП, количество потоков программы (по данным диспетчера задач), размер обрабатываемого массива и время выполнения при каждом запуске **записать в таблицу** (форма таблицы произвольная).

6. Полученные результаты обработать: вычислить реальное значение выигрыша по производительности и сравнить со значением выигрыша, найденного по закону Амдала. Сравнить характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива) и асимптотической оценки трудоемкости алгоритма, который реализует исследуемая программа. Для нахождения оценок параметров модели, описывающей зависимость времени выполнения программы от размера обрабатываемых данных, **рекомендуется** использовать метод наименьших квадратов.

7. Каждая бригада должна выполнить исследование алгоритма УМНОЖЕНИЯ МАТРИЦ и алгоритма, указанного в индивидуальном задании – таблица 1. Исследование алгоритма состоит в последовательном выполнении пунктов 1-6.

Текст программы умножения матриц (на языке PascalABC.Net) приведен ниже. Распараллеливание циклов For реализуется с помощью директив OpenMP.

```
uses Arrays;
procedure ParallelMult(a,b,c: array [,] of real; n: integer);
begin
  {$omp parallel for }
  for var i:=0 to n-1 do
    for var j:=0 to n-1 do
    begin
      c[i,j]:=0;
      for var l:=0 to n-1 do
        c[i,j]:=c[i,j]+a[i,l]*b[l,j];
    end;
end;

procedure Mult(a,b,c: array [,] of real; n: integer);
begin
  for var i:=0 to n-1 do
    for var j:=0 to n-1 do
    begin
      c[i,j]:=0;
      for var l:=0 to n-1 do
        c[i,j]:=c[i,j]+a[i,l]*b[l,j];
    end;
end;
```

```
const n = 300;

begin
  var a := Arrays.CreateRandomRealMatrix(n,n);
  var b := Arrays.CreateRandomRealMatrix(n,n);
  var c := new real[n,n];
  ParallelMult(a,b,c,n);
  writeln('Параллельное перемножение матриц: ',Milliseconds,' миллисекунд');
  var d := Milliseconds;
  Mult(a,b,c,n);
  writeln('Последовательное перемножение матриц: ',Milliseconds-d,' миллисекунд');
end.
```

При отсутствии на ПК платформы PascalABC.Net для выполнения работы используется файл multMatrix.exe из каталога лабораторной работы.

Таблица 1. Индивидуальные задания

№	Алгоритм
1	Быстрая сортировка QuickSort
2	Ханойские башни Hanoi
3	Нахождение суммы простых чисел SumOfPrime
4	Сложение матриц
5	Сортировка методом пузырька

5. ОТЧЕТ О РАБОТЕ

Содержание отчета:

4. Таблицы, содержащие результаты выполнения п. 2 (времена выполнения программ на одном, двух и более ядрах ЦП)
5. Результаты обработки данных построенных в п.1 таблиц – теоретическая и эмпирическая оценки выигрыша в производительности.
6. Оценки трудоемкости исследованных алгоритмов.
7. Выводы

8.КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Симметричная и асимметричная архитектуры аппаратных и программных средств.
2. Достоинства симметричной архитектуры.
3. Понятие SMP
4. Закон Амдала
5. Трудоемкость алгоритма
6. Трудоемкость алгоритмов умножения матриц, сложения матриц и сортировки массива методом пузырька.
7. Трудоемкость алгоритма быстрой сортировки

Лабораторная работа № 6. Файловые системы операционной системы LINUX (4 часа)

Цель работы – практическое знакомство с организацией данных основной файловой системы ОС Linux и используемыми утилитами.

Файл

Данные, хранящиеся на любом носителе, образуют *файл* Linux. Более того, многие устройства, подключенные к компьютеру (начиная с клавиатуры и заканчивая любыми внешними устройствами), Linux представляет как файлы (так называемые *специальные файлы*). В Linux определено несколько различных типов файлов. В основном пользователь имеет дело с файлами трех типов: обычными файлами, предназначенными для хранения данных, *каталогами* и *файлами-ссылками.*

Система файлов: каталоги

Файловая система имеет иерархическую структуру. Linux может работать с различными типами файловых систем. В этой работе будут описаны возможности файловой системы Ext3fs. В файловой системе Ext3fs каждый *каталог* - это отдельный *файл* особого типа ("d", от англ. "directory"), отличающийся от обычного файла с данными: в нем могут содержаться только ссылки на другие файлы и каталоги.

Допустимые имена файлов и каталогов

Linux всегда **различает** заглавные и строчные буквы в именах файлов и каталогов, поэтому "student", "Student" и "STUDENT" будут тремя **разными** именами.

Есть несколько символов, допустимых в именах *файлов* и *каталогов*, которые нужно использовать с осторожностью. Это *спецсимволы* "*", "\", "&", "<", ">", ";", "(", ")", "|", а также символы пробела и табуляции.

Кодировки и расширения

В Linux в именах *файлов* и *каталогов* допустимо использовать не только символы латинского алфавита, но и любые символы любого языка.

В файловой системе Linux нет никаких предписаний по поводу расширения: в *имени файла* может быть любое количество точек (в том числе ни одной), а после последней точки может стоять любое количество символов. Хотя расширения не обязательны, они широко используются: расширение позволяет программе, не открывая файл, только по его имени определить, какого типа данные в нем содержатся. Определить тип содержимого файла можно и на основании самих данных (сигнатур). Многие форматы предусматривают указание в начале файла, как следует интерпретировать дальнейшую информацию.

В Linux есть утилита file, которая предназначена для определения типа содержащихся в файле данных. Эта утилита никогда не доверяет расширению *файла* (если оно присутствует), а анализирует сами данные. file различает не только разные данные, но и разные типы *файлов*.

1.1. Дерево каталогов

В большинстве современных *файловых систем* используется иерархическая модель организации данных: существует один *каталог*, объединяющий все данные в *файловой системе* - это "корень" всей *файловой системы*, ***корневой каталог***. *Корневой каталог* может содержать любые объекты *файловой системы*, и в частности, *подкаталоги*. Подкаталоги также могут содержать любые объекты *файловой системы* и *подкаталоги* и т. д. Таким образом, **все**, что записано на диске - *файлы*, *каталоги* и специальные *файлы* - обязательно "принадлежит" *корневому каталогу*: либо непосредственно (содержится в нем), либо на некотором уровне вложенности.

Структуру *файловой системы* можно представить наглядно в виде дерева, "корнем" которого является *корневой каталог*, а в вершинах расположены все остальные *каталоги*. На рис. 1 изображено дерево *каталогов*, курсивом обозначены имена *файлов*, прямым начертанием - имена *каталогов*.

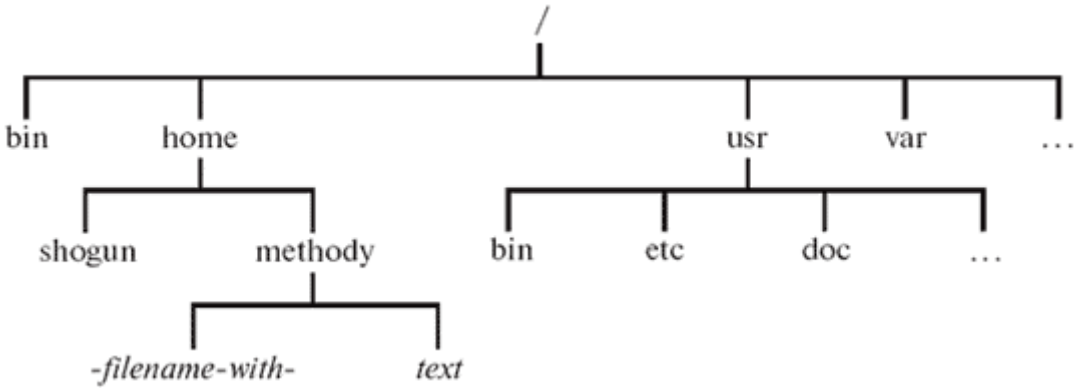


Рис. 1. Дерево каталогов в ext3fs

В любой *файловой системе* Linux всегда есть только один *корневой каталог*, который называется "/". Пользователь Linux всегда работает с единым деревом *каталогов*, даже если разные данные расположены на разных носителях: жестких или сетевых дисках, съемных дисках, CD-ROM и т. п. Такое представление отличается от технологии, применяемой в Windows, где для каждого устройства, на котором есть файловая система, используется свой корневой каталог, обозначенный литерой, например "a", "c", "d" и т. д. Для того чтобы отключать и подключать файловые системы на разных устройствах в состав одного общего дерева, используются процедуры монтирования и размонтирования. После того, как файловые системы на разных носителях подключены к общему дереву, содержащиеся на них данные доступны так, как если бы все они составляли единую файловую систему: пользователь может даже не знать, на каком устройстве какие файлы хранятся.

Положение любого *каталога* в дереве *каталогов* описывается при помощи *полного пути*. *Полный путь* всегда начинается от корневого каталога и состоит из перечисления всех вершин, встретившихся при движении по ветвям дерева до искомого *каталога* включительно. Названия соседних вершин разделяются символом "/" ("слэш"). В Linux *полный путь*, например, до *каталога* "methody" в *файловой системе*, приведенной на рис. 1, записывается следующим образом /home/methody.

1.2. Размещение компонентов системы: стандарт FHS

Фрагмент дерева *каталогов* типичной файловой системы Linux приведен на рис. 1. Утилита ls выведет список всего, что в этом каталоге содержится.

Пример 1. Стандартные каталоги в / . Использование утилиты ls

```
[student@localhost ~]$ ls /
bin  dev  home  lost+fount  misc  net  proc  tmp  var
boot  etc  lib  sbin  usr
```

В примере 1 утилита ls вывела список подкаталогов корневого каталога. Этот список будет примерно таким же в любом дистрибутиве Linux. В *корневом каталоге* Linux-системы обычно находятся только подкаталоги со **стандартными** именами. Более того, не только имена, но и **тип данных**, которые могут попасть в тот или иной *каталог*, также регламентированы стандартом **Filesystem Hierarchy Standard** ("стандартная структура *файловых систем*"). Краткое описание стандартной иерархии каталогов Linux можно получить, выполнив команду man hier. Полный текст и последнюю редакцию стандарта FHS можно прочесть по адресу http://www.pathname.com/fhs/

Содержимое подкаталогов корневого *каталога*.

/bin	Название этого *каталога* происходит от слова "binaries" ("двоичные", "исполняемые"). В этом *каталоге* находятся исполняемые *файлы* самых необходимых утилит, которые могут понадобиться системному администратору или другим пользователям.

/boot	"Boot" - загрузка системы. В этом *каталоге* находятся *файлы*, необходимые для загрузки ядра - и, обычно, само *ядро*. Пользователю практически никогда не требуется непосредственно работать с этими *файлами*.
/dev	В этом *каталоге* находятся все имеющиеся в системе *файлы* особого типа, предназначенные для обращения к различным системным ресурсам и устройствам. Например, *файлы* /dev/ttyN соответствуют *виртуальным консолям*, где N - номер *виртуальной консоли*. Данные, введенные пользователем на первой *виртуальной консоли*, система считывает из *файла* /dev/tty1; в этот же *файл* записываются данные, которые нужно вывести пользователю на эту консоль. В специальных *файлах* в действительности не хранятся никакие данные, при их помощи данные **передаются**.
/etc	*Каталог* для системных *конфигурационных файлов*. Здесь хранится информация о специфических настройках данной системы: информация о зарегистрированных пользователях, доступных ресурсах, настройках различных программ.
/home	Здесь расположены *каталоги*, принадлежащие пользователям системы - ***домашние каталоги***, отсюда и название "home". Отделение всех *файлов*, создаваемых пользователями, от прочих системных *файлов* дает очевидное преимущество: серьезное повреждение системы или необходимость обновления не затронет пользовательских *файлов*.
/lib	Название этого *каталога* - сокращение от "libraries" (англ. "библиотеки"). Чтобы не включать эти функции в текст каждой программы, используются стандартные функции библиотек - это значительно экономит место на диске и упрощает написание программ. В этом *каталоге* содержатся библиотеки, необходимые для работы наиболее важных системных утилит, размещенных в /bin и /sbin.
/mnt	*Каталог* для *монтирования* (от англ. "mount") - временного подключения *файловых систем*, например, на съемных носителях (CD-ROM и др.).
/proc	В этом *каталоге* все *файлы* "виртуальные" - они располагаются не на диске, а в оперативной памяти. В этих *файлах* содержится информация о программах (*процессах*), выполняемых в данный момент в системе.
/root	*Домашний каталог* администратора системы - пользователя root. Смысл размещать его отдельно от *домашних каталогов* остальных пользователей состоит в том, что /home может располагаться на отдельном устройстве, которое не всегда доступно (например, на сетевом диске), а *домашний каталог* root должен присутствовать в любой ситуации.
/sbin	*Каталог* для важнейших системных утилит (название *каталога* - сокращение от "system binaries"): в дополнение к утилитам /bin здесь находятся программы, необходимые для загрузки, резервного копирования, восстановления системы. Полномочия на исполнение этих программ есть только у системного администратора.
/tmp	Этот *каталог* предназначен для *временных файлов*: в таких *файлах* программы хранят необходимые для работы промежуточные данные. После завершения работы программы *временные файлы* теряют смысл и должны быть удалены. Обычно *каталог* /tmp очищается при каждой загрузке системы.
/usr	Здесь можно найти такие же *подкаталоги* bin, etc, lib, sbin, как и в *корневом каталоге*. Однако в *корневой каталог* попадают только утилиты, **необходимые** для загрузки и восстановления системы в аварийной ситуации - **все остальные** программы и данные располагаются в *подкаталогах* /usr. Этот раздел *файловой системы* может быть очень большим.
/var	Название этого *каталога* - сокращение от "variable" ("переменные" данные). Здесь размещаются те данные, которые создаются в *процессе* работы разными программами и предназначены для передачи другим программам и системам (очереди печати,

	электронной почты и др.) или для сведения системного администратора (системные журналы, содержащие протоколы работы системы). В отличие от каталога /tmp сюда попадают те данные, которые могут понадобиться после того, как создавшая их программа завершила работу.

Стандарт *FHS* регламентирует не только перечисленные каталоги, но и их подкаталоги, а иногда даже приводит список конкретных файлов, которые должны присутствовать в определенных каталогах. Этот стандарт последовательно соблюдается во всех Linux-системах.

Командная оболочка "знает", что исполняемые файлы располагаются в каталогах /bin, /usr/bin и т. д. - именно в этих каталогах она ищет исполняемый файл cat. Благодаря этому каждая вновь установленная в системе программа немедленно оказывается доступна пользователю из командной строки. Для этого не требуется ни перезагружать систему, ни запускать какие-либо процедуры - достаточно просто поместить исполняемый файл в один из соответствующих каталогов.

Рекомендации стандарта по размещению файлов и каталогов основываются на принципе размещения файлов, которые по-разному используются в системе, в разных подкаталогах. По типу использования файлы можно разделить на следующие группы:

1. **пользовательские/системные файлы**

Пользовательские файлы - это все файлы, созданные пользователем и не принадлежащие ни одному из компонентов системы.

2. **изменяющиеся/неизменные файлы**

К неизмененным файлам относятся все статические компоненты программного обеспечения: библиотеки, исполняемые файлы и т. д. - все, что не изменяется само без вмешательства системного администратора. Изменяющиеся файлы изменяются без вмешательства человека в процессе работы системы: системные журналы, очереди печати и пр. Выделение неизменных файлов в отдельную структуру (например, /usr) позволяет использовать соответствующую часть файловой системы в режиме "только чтение", что уменьшает вероятность случайного повреждения данных и позволяет применять для хранения этой части файловой системы CD-ROM и другие носители, доступные только для чтения.

3. **разделяемые/неразделяемые файлы**

Это разграничение становится полезным, если речь идет о сети, в которой работает несколько компьютеров. Значительная часть информации при этом может храниться на одном из компьютеров и использоваться всеми остальными по сети (к такой информации относятся, например, многие программы и домашние каталоги пользователей). Однако часть файлов нельзя разделять между системами (например, файлы для начальной загрузки системы).

Полный путь к каталогу формально ничем не отличается от пути к файлу, т. е. по полному пути нельзя сказать наверняка, является его последний элемент файлом или каталогом. Чтобы отличать путь к каталогу, иногда используют запись с символом "/" в конце пути, например "/home/student/".

1.3. Текущий каталог

Каждая выполняемая программа "работает" в строго определенном *каталоге* файловой системы. Такой каталог называется *текущим каталогом***.** Можно представлять, что программа во время работы "находится" именно в этом каталоге, это ее "рабочее место". В зависимости от текущего каталога поведение программы может меняться: зачастую программа будет по умолчанию работать с файлами, расположенными именно в текущем каталоге - до них она "дотянется" в первую очередь. Текущий каталог есть у любой программы, в том числе и у командной оболочки (shell) пользователя. Поскольку пользователь взаимодействует с системой через командную оболочку, можно говорить о том, что пользователь "находится" в том *каталоге*, который в данный момент является текущим каталогом его командной оболочки.

Все команды, выполняемые пользователем при помощи shell, наследуют *текущий каталог* shell, т. е. "работают" в том же *каталоге*. По этой причине пользователю важно знать *текущий каталог* shell. Для этого служит утилита pwd:

Команда pwd (print working directory) возвращает *полный путь* текущего каталога командной оболочки - естественно, именно той командной оболочки, при помощи которой была выполнена команда pwd.

Почти все утилиты по умолчанию читают и создают файлы в *текущем каталоге*. Например, утилита cat (concatenation – конкатенация) - выводит на экран содержимое файла "text":

[student@localhost student]$ cat text

В действительности, командная оболочка, прежде чем передавать параметр "text" (имя файла) утилите cat, подставляет значение *текущего каталога* - получается *полный путь* к этому файлу в *файловой системе*: "/home/student/text". Содержимое данного файла утилита cat выведет на экран.

Относительный путь(relative path) - путь к объекту *файловой системы*, не начинающийся в *корневом каталоге*. Для каждого *процесса* Linux определен *текущий каталог,* с которого система начинает *относительный путь* при выполнении файловых операций.

Относительный путь строится точно так же, как и полный - перечислением через "/" всех названий *каталогов*, встретившихся при движении к искомому *каталогу* или файлу. Между полным и *относительным путем* есть только одно существенное различие: *относительный путь* начинается **от текущего каталога**, в то время как *полный путь* всегда начинается **от корневого каталога**. *Относительный путь* любого файла или каталога в файловой системе может иметь любую конфигурацию - чтобы добраться до искомого файла, можно двигаться как по направлению к *корневому каталогу*, так и от него. Linux различает полный и *относительный пути* очень просто: если имя объекта начинается на "/" - это *полный путь*, в любом другом случае - относительный.

Отделить *путь* к файлу от его имени можно с помощью команд dirname и basename соответственно.

1.4. Домашний каталог

В Linux у каждого пользователя обязательно есть **собственный каталог**, который и становится текущим сразу после *регистрации в системе* - домашний каталог.

Домашний каталог (home directory) - это каталог, предназначенный для хранения собственных данных пользователя Linux. Как правило, является текущим непосредственно после регистрации пользователя в системе. *Полный путь* к *домашнему каталогу* хранится в *переменной окружения* HOME. Имя домашнего каталога ~

Поскольку каждый пользователь располагает собственным *каталогом* и по умолчанию работает в нем, решается задача разделения файлов разных пользователей. Обычно доступ других пользователей к чужому *домашнему каталогу* ограничен: наиболее типична ситуация, когда пользователи могут читать содержимое файлов друг друга, но не имеют права их изменять или удалять.

1.5. Информация о содержимом каталога – утилита ls

Чтобы иметь возможность ориентироваться в *файловой системе*, нужно знать, что содержится в каждом *каталоге*. Просмотреть содержимое любого *каталога* можно при помощи утилиты ls (сокращение от англ. "list" - "список"):

Команда ls без параметров выводит список файлов и каталогов, содержащихся в текущем каталоге. Утилита ls принимает один параметр - имя каталога, содержимое которого нужно вывести. Имя может быть задано любым доступным способом: в виде полного или *относительного пути*.

Кроме параметра, утилита ls может использовать множество ключей, которые нужны для того, чтобы выводить дополнительную информацию о файлах в каталоге или выводить список файлов выборочно. Чтобы узнать обо всех возможностях ls, нужно прочесть *руководство* по этой утилите с помощью команды man ls.

Ключ –F используется, чтобы отличать файлы от каталогов. При наличии этого ключа ls в конце имени каждого каталога ставит символ "/", чтобы показать, что в нем может содержаться что-то еще.

Утилита ls по умолчанию не выводит информацию об объектах, чье имя начинается с "." - в том числе о "." и "..". Для того чтобы посмотреть полный список содержимого *каталога*, и используется ключ "-a" (all). Как правило, с "." начинаются имена *конфигурационных файлов* и *конфигурационных каталогов* (вроде .bashrc), работа с которыми (т. е. **настройка** окружения, "рабочего места") не пересекается с работой над какой-нибудь прикладной задачей.

Родительский каталог (parent directory) - это *каталог*, в котором содержится данный. Для корневого каталога родительским является он сам.

Ссылки на текущий и на *родительский каталог* обязательно присутствуют в **каждом каталоге** в Linux. Даже если каталог пуст, т. е. не содержит ни одного файла или подкаталога, команда "ls -a" выведет список из двух имен: "." и "..". За ссылками на текущий и *родительский каталоги* могут следовать несколько файлов и каталогов, имена которых начинаются с ".". В них содержатся настройки командной оболочки (файлы, начинающиеся с ".bash") и других программ. В *домашнем каталоге* каждого пользователя Linux всегда присутствует несколько таких файлов. Использование этих файлов позволяет пользователям независимо друг от друга настраивать поведение командной оболочки и других программ - организовывать свое "рабочее место" в системе.

1.6. Перемещение по дереву каталогов – команда cd

Пользователь может работать с файлами не только в своем *домашнем каталоге*, но и в других *каталогах*. В этом случае будет удобно **сменить текущий каталог**. Для смены *текущего каталога* командной оболочки используется команда cd (от англ. "change directory" - "сменить *каталог*"). Команда cd принимает один параметр: имя *каталога*, в который нужно переместиться - сделать текущим. В качестве имени *каталога* можно использовать полный или *относительный путь*. В *приглашении командной строки* часто указывается *текущий каталог* shell - чтобы пользователю легче было ориентироваться, в каком *каталоге* он "находится" в данный момент.

Командная оболочка умеет **достраивать** имена файлов и *каталогов*: пользователю достаточно набрать несколько первых символов имени файла или *каталога* и нажать Tab. Если есть только один вариант завершения имени - оболочка закончит его сама, и пользователю не придется набирать оставшиеся символы. Достраивание - весьма существенное средство экономии усилий и повышения эффективности при работе с командной строкой. Современные командные оболочки умеют достраивать имена файлов и *каталогов*, а также имена команд. Достраивание наиболее развито в командном интерпретаторе zsh. Оболочка PowerShell также умеет достраивать имена.

Для перемещения в родительский каталог ("/home") удобно воспользоваться ссылкой "..". Необходимость вернуться в домашний каталог из произвольной точки файловой системы возникает довольно часто, поэтому командная оболочка поддерживает обозначение домашнего каталога при помощи символа "~". Поэтому чтобы перейти в домашний каталог из любого другого, достаточно выполнить команду "cd ~". При исполнении команды символ "~" будет заменен командной оболочкой на *полный путь* к *домашнему каталогу* пользователя.

При помощи символа "~" можно ссылаться и на *домашние каталоги* других пользователей: "~имя пользователя". Команда cd, поданная без параметров, эквивалентна команде "cd ~" и делает *текущим каталогом* домашний каталог пользователя.

1.7. Создание каталогов – утилита mkdir

В *домашнем каталоге*, как и в любом другом, можно создавать сколько угодно подкаталогов, в них - свои подкаталоги и т. д. Иными словами, пользователю принадлежит фрагмент (поддерево) *файловой системы*, корнем которого является его *домашний каталог*.

Чтобы организовать такое поддерево, потребуется создать *каталоги* внутри домашнего. Для этого используется утилита mkdir. Она применяется с одним обязательным параметром: именем создаваемого *каталога*. По умолчанию *каталог* будет создан в *текущем каталоге*.

1.7.1. Создание нового пустого файла – команда touch

Для создания пустого файла с текущим временем создания служит команда touch имя_нового_файла. Для указания даты создания в формате ГГГГММДДhhmm используется ключ –t. Например

touch –t 0904080000 tst файл создан 8 апреля 2009 г.

1.8. Копирование и перемещение файлов

Для перемещения файлов и *каталогов* предназначена утилита mv (от англ. "move" - "перемещать"). У mv два обязательных параметра: первый - перемещаемый файл или *каталог*, второй - файл или *каталог* назначения. Имена файлов и *каталогов* могут быть заданы в любом допустимом виде: при помощи полного или относительного пути. Кроме того, mv позволяет перемещать не только один файл или *каталог*, а сразу несколько. За подробностями о допустимых параметрах и ключах следует обратиться к руководству по mv:

Перемещение файла внутри одной *файловой системы* в действительности равнозначно его **переименованию**: данные самого файла при этом остаются на тех же секторах диска, а изменяются *каталоги*, в которых произошло перемещение. Перемещение предполагает удаление ссылки на файл из того *каталога*, откуда он перемещен, и добавление ссылки на этот самый файл в тот *каталог*, куда он перемещен. В результате изменяется полное имя файла - *полный путь*, т. е. положение файла в *файловой системе*.

Иногда требуется создать копию файла: для большей сохранности данных, для того, чтобы создать модифицированную версию файла и т. п. В Linux для этого предназначена утилита cp (от англ. "copy" - "копировать"). Утилита cp требует использования двух обязательных параметров: первый - копируемый файл или *каталог*, второй - файл или *каталог* назначения. Как обычно, в именах файлов и *каталогов* можно использовать полные и *относительные пути*. Существует несколько вариантов комбинации файлов и *каталогов* в параметрах cp - о них можно прочесть в руководстве. Нужно иметь в виду, что в Linux утилита cp нередко настроена таким образом, что при попытке скопировать файл поверх уже существующего файла никакого предупреждения не выводится. В этом случае файл будет просто перезаписан, а данные, которые содержались в старой версии файла, безвозвратно потеряны. Поэтому при использовании cp следует всегда быть внимательным и проверять имена файлов, которые нужно скопировать.

Созданная при помощи cp копия файла связана с оригиналом только в воспоминаниях пользователя, в файловой же системе исходный файл и его копия - две совершенно независимые и ничем не связанные единицы. Поэтому при наличии нескольких копий одного и того же файла в рамках **одной файловой системы** повышается вероятность запутаться в копиях или забыть о некоторых из них. Если задача состоит в том, чтобы обеспечить доступ к одному и тому же файлу из разных точек *файловой системы*, нужно использовать специально предназначенный для этого механизм *файловой системы* Linux - ссылки.

1.9. Файл и его имена: ссылки

1.9.1. Жесткие ссылки – утилита ln

Каждый файл представляет собой область данных на жестком диске компьютера или на другом носителе информации, которую можно найти **по имени**. В файловой системе Linux содержимое файла связывается с его именем при помощи *жестких ссылок*. Создание файла с помощью любой программы означает, что будет создана *жесткая ссылка* - имя файла, и открыта новая область данных на диске. Причем количество ссылок на одну и ту же область данных (файл) не ограничено, то есть у файла может быть несколько имен.

Пользователь Linux может добавить файлу еще одно имя (создать еще одну *жесткую ссылку* на файл) при помощи утилиты ln (от англ. "link" - "соединять, связывать"). Первый параметр - это имя файла, на который нужно создать ссылку, второй - имя новой ссылки. По умолчанию ссылка будет создана в *текущем каталоге*:

Пример 2. Создание жестких ссылок

```
[student@localhost ~]$ ln text text-hardlink
```

В примере 2 в *домашнем каталоге* пользователя student создана жесткая ссылка с именем "text-hardlink" на файл "text". Если вывести подробный список файлов *текущего каталога* и его подкаталогов ("ls -lR"), то у файлов "text" и "text-hardlink" совпадут и размер, и время создания. Теперь "text-hardlink" и "text" - это два имени одного и того же файла.

Доступ к одному и тому же файлу при помощи нескольких имен может понадобиться в следующих случаях:

1. Одна и та же программа известна под несколькими именами.
2. Доступ пользователей к некоторым *каталогам* в системе может быть ограничен из соображений безопасности. Однако если все же нужно организовать доступ пользователей к файлу, который находится в таком *каталоге*, можно создать *жесткую ссылку* на этот файл в другом *каталоге*.
3. Современные *файловые системы* даже на домашних персональных компьютерах могут насчитывать до нескольких десятков тысяч файлов и тысячи *каталогов*. Обычно у таких *файловых систем* сложная многоуровневая иерархическая организация - в результате пути ко многим файлам становятся очень длинными. Чтобы организовать более удобный доступ к файлу, который находится очень "глубоко" в иерархии *каталогов*, также можно использовать *жесткую ссылку* в более доступном *каталоге*.
4. Полное имя некоторых программ может быть весьма длинным (например, i586-alt-linux-gcc-3.3), к таким программам удобнее обращаться при помощи сокращенного имени (*жесткой ссылки*) - gcc-3.3.

1.9.2. Индексные дескрипторы

Поскольку благодаря *жестким ссылкам* у файла может быть несколько имен, понятно, что вся существенная информация о файле в *файловой системе* привязана не к имени. В *файловых системах* Linux вся информация, необходимая для работы с файлом, хранится в *индексном дескрипторе*. Для **каждого** файла существует *индексный дескриптор*: не только для обычных файлов, но и для каталогов, *файлов-дырок* и т. д. Каждому файлу соответствует **один индексный дескриптор**.

Индексный дескриптор - это описание файла, в котором содержится:

- тип файла (обычный файл, *каталог*, специальный *файл* и т. д.);
- *права доступа* к файлу;
- информация о том, кому принадлежит файл;
- отметки о времени создания, модификации, последнего доступа к файлу;
- размер файла;
- указатели на физические блоки на диске, принадлежащие этому файлу - в этих блоках хранится "содержимое" файла.

Все *индексные дескрипторы* пронумерованы, поэтому номер *индексного дескриптора* - это уникальный идентификатор файла в *файловой системе* - в отличие от **имени** файла

(*жесткой ссылки* на него), которых может быть несколько. Узнать номер *индексного дескриптора* любого файла можно при помощи утилиты ls с ключом –i

Если вывести номера *индексных дескрипторов* файла "text" и *жесткой ссылки* на него "text-hardlink" – можно увидеть, что эти номера совпадают, то есть этим двум именам соответствует один *индексный дескриптор*, т. е. один и тот же файл.

Все операции с *файловой системой* - создание, удаление и перемещение файлов - производятся на самом деле над *индексными дескрипторами*, а имена нужны только для того, чтобы пользователь мог легко ориентироваться в файловой системе. Более того, имя (или имена) файла в его **индексном дескрипторе не указаны**. В *файловой системе* Ext2 имена файлов хранятся в *каталогах*: каждый *каталог* представляет собой список имен файлов и номеров их *индексных дескрипторов*. *Жесткую ссылку* (имя файла, хранящееся в *каталоге*) можно представлять как каталожную карточку, на которой указан номер *индексного дескриптора* - идентификатор файла.

Жесткая ссылка (hard link) - запись вида имя файла+номер *индексного дескриптора* в *каталоге*. *Жесткие ссылки* в Linux - основной способ обратиться к файлу по имени.

1.9.3 Символьные ссылки

У *жестких ссылок* есть два существенных ограничения:

1. *Жесткая ссылка* может указывать только на файл, но не на *каталог*, потому что в противном случае в *файловой системе* могут возникнуть циклы - бесконечные пути.
2. *Жесткая ссылка* не может указывать на файл в другой *файловой системе*. Например, невозможно создать на жестком диске *жесткую ссылку* на файл, расположенный на дискете. Чтобы избежать этих ограничений, были разработаны *символьные ссылки*. ***Символьная ссылка*** - это просто файл, в котором содержится имя другого файла. *Символьные ссылки*, как и жесткие, предоставляют возможность обращаться к одному и тому же файлу по разным именам. Кроме того, *символьные ссылки* могут указывать и на *каталог*, чего не позволяют *жесткие ссылки*. *Символьные ссылки* называются так потому, что содержат **символы** - *путь* к файлу или *каталогу*.

Символьная ссылка (symbolic link, файл-ссылка) - это файл особого типа ("l"), в котором содержится *путь* к другому файлу. Если на пути к файлу встречается *символьная ссылка*, система выполняет подстановку: исходный *путь* заменяется тем, что содержится в ссылке.

Символьную ссылку можно создать при помощи команды ln с ключом "-s" (сокращение от "symbolic").

Если выполнить команду cat имя_файла-ссылки**,** то на экран будет выведено содержимое файла, на который указывает ссылка.

Символьная ссылка вполне может содержать имя несуществующего файла. В этом случае ссылка будет существовать, но не будет "работать": например, если попробовать вывести содержимое такой "битой" ссылки при помощи команды cat, будет выдано сообщение об ошибке. Узнать, куда указывает *символьная ссылка*, можно при помощи утилиты realpath.

1.10. Удаление файлов и каталогов – утилиты rm и rmdir

В ОС Linux для удаления файлов предназначена утилита rm (сокращение от англ. "remove" - "удалять"):

Если удалить файл text в домашнем каталоге пользователя student, файл text-hardlink, который является *жесткой ссылкой* на удаленный файл text, сохранится, количество *жестких ссылок* на этот файл уменьшится с "2" до "1" - действительно, text-hardlink - теперь единственное имя этого файла. Однако если удалить и *жесткую ссылку* text-hardlink, у этого файла больше не останется ни одного имени, он станет недоступным пользователю и будет уничтожен.

Утилита rm предназначена именно для удаления *жестких ссылок*, а не самих файлов. В Linux, чтобы полностью удалить файл, требуется последовательно удалить все *жесткие ссылки* на него. При этом все *жесткие ссылки* на файл (его имена) равноправны - среди них нет "главной", с исчезновением которой исчезнет файл. Пока есть хоть одна ссылка, файл продолжает существовать. Впрочем, у большинства файлов в Linux есть только одно имя (одна *жесткая ссылка* на файл), поэтому команда rm имя файла в большинстве случаев успешно удаляет файл.

Как уже говорилось, *символьные ссылки* - это отдельные файлы, поэтому после удаления файла text, text-symlink, который ссылался на этот файл, продолжает существовать, однако теперь это - "битая ссылка", поэтому его также можно удалить командой rm.

Для удаления *каталогов* предназначена другая утилита - rmdir (от англ. "remove directory"). Впрочем, rmdir согласится удалить *каталог* только в том случае, если он пуст - в нем нет никаких файлов и подкаталогов. Удалить *каталог* вместе со всем его содержимым можно командой rm с ключом "-r" (recursive). Команда rm -r *каталог* - очень удобный способ потерять в одночасье **все** файлы: она рекурсивно обходит весь *каталог*, удаляя все, что попадется: файлы, подкаталоги, *символьные ссылки*... а ключ "-f" (force) делает ее работу еще неотвратимее, так как подавляет запросы вида "удалить защищенный от записи файл", так что rm работает безмолвно и безостановочно.

ПОМНИТЕ: если вы удалили файл, значит, он уже не нужен, и не подлежит восстановлению!

В Linux не предусмотрено процедуры восстановления удаленных файлов и *каталогов*. Поэтому стоит быть **очень** внимательным, отдавая команду rm и, тем более, rm -r: нет никакой гарантии, что случайно удаленные данные удастся восстановить.

1.11. Права доступа в файловой системе

1.11.1. Идентификатор пользователя

Говоря о *правах доступа* пользователя к файлам, заметим, что в действительности манипулирует файлами не сам пользователь, а запущенный им *процесс* (например, утилита rm или cat). Поскольку и файл, и *процесс* создаются и управляются системой, ей нетрудно организовать какую угодно политику доступа одних к другим, основываясь на любых свойствах *процессов* как субъектов и файлов как объектов системы.

В Linux, однако, используются не какие угодно свойства, а результат *идентификации* пользователя – его UID. Каждый *процесс* системы обязательно **принадлежит** какому-нибудь пользователю, и *идентификатор пользователя* (UID) – обязательное свойство любого *процесса* Linux. Когда программа login запускает стартовый командный интерпретатор, она приписывает ему UID, полученный в результате диалога. Обычный запуск программы (exec()) или порождение нового *процесса* (fork()) не изменяют UID процесса, поэтому **все** процессы, запущенные пользователем во время терминальной сессии, будут иметь его идентификатор.

Поскольку UID однозначно определяется входным именем, оно нередко используется **вместо** *идентификатора* – для наглядности. Например, вместо выражения "*идентификатор пользователя*, соответствующий *входному имени* student", говорят "UID student" (в приведенном ниже примере этот *идентификатор* равен **500**):

Пример 3. Как узнать идентификаторы пользователя и членство в группах

```
[student@localhost student]$ id
 uid=500 (student) gid=500(student) группы=500 (student)
```

Утилита id выводит *входное имя* пользователя и соответствующий ему UID, а также *группу по умолчанию* и полный список *групп*, членом которых он является.

1.11.2. Идентификатор группы

Пользователь может быть членом нескольких *групп*, равно как и несколько пользователей могут быть членами одной и той же *группы*. Исторически сложилось так, что одна из *групп – группа по умолчанию* – является для пользователя основной - когда говорят о "GID пользователя", имеют в виду именно идентификатор *группы по умолчанию*. GID пользователя вписан в *учетную запись* и хранится в /etc/passwd, а информация о соответствии **имен** *групп* их идентификаторам, равно как и о том, в какие **еще** *группы* входит пользователь – в файле /etc/group. Из этого следует, что пользователь **не может не быть** членом как минимум одной *группы*.

1.11.3. Ярлыки объектов файловой системы

При создании объектов файловой системы – файлов, каталогов и т. п. – каждому приписывается *ярлык*. ***Ярлык*** включает в себя UID – *идентификатор пользователя-хозяина файла*, GID – идентификатор *группы*, которой принадлежит файл, тип объекта и набор так называемых *атрибутов* (код доступа), а также некоторую дополнительную информацию. *Атрибуты* (или код доступа) определяют, кто и что имеет право делать с файлом, они описаны ниже:

Пример 4. Атрибуты каталогов, показанные командой ls -l

```
 итого 88
drwxr-xr-x    2 root root  4096 Апр  4  2011 bin
drwxr-xr-x    4 root root  4096 Апр  4  2011 boot
drwxr-xr-x   10 root root  3520 Апр  5 14:26 dev
drwxr-xr-x   90 root root  8192 Апр  5 14:22 etc
drwxr-xr-x    3 root root  4096 Апр  4 21:22 home
drwxr-xr-x   11 root root  4096 Апр  4  2011 lib
drwx------    2 root root 16384 Апр  4  2011 lost+found
drwxr-xr-x    4 root root  4096 Апр  5 14:22 media
drwxr-xr-x    2 root root  4096 Июл 11  2010 misc
drwxr-xr-x    2 root root  4096 Окт 20  2011 mnt
drwxr-xr-x    2 root root     0 Апр  5 14:21 net
drwxr-xr-x    2 root root  4096 Окт 20  2011 opt
dr-xr-xr-x  106 root root     0 Апр  5  2010 proc
drwxr-x---   31 root root  4096 Апр  5 14:29 root
drwxr-xr-x    2 root root  8192 Апр  4  2010 sbin
drwxr-xr-x    2 root root  4096 Окт 20  2010 selinux
drwxr-xr-x    2 root root  4096 Окт 20  2010 srv
drwxr-xr-x   11 root root     0 Апр  5  2010 sys
drwxrwxrwt 16 root root  4096 Апр  5 14:26 tmp
drwxr-xr-x   15 root root  4096 Апр  4  2010 usr
drwxr-xr-x   21 root root  4096 Апр  4  2010 var
```

Ключ "-l" утилиты ls определяет длинный (**l**ong) формат выдачи (справа налево): имя файла, время последнего изменения файла, размер в байтах, группа, хозяин, количество *жестких ссылок* и строчка *атрибутов*. Первый символ в строчке *атрибутов* определяет тип файла. Тип "-" отвечает "обычному" файлу, а тип "d" – каталогу (**d**irectory).

Несмотря на то, что создание *жестких ссылок* на каталог невозможно, значение поля "количество *жестких ссылок*" (второй столбец) для всех каталогов примера равно **двум**, а не одному. На самом деле этого и следовало ожидать, потому что **любой** каталог файловой системы Linux всегда имеет не менее двух имен: собственное (например, tmp) и имя "." в самом этом каталоге (tmp/.). Если же в каталоге создать подкаталог, количество *жестких ссылок* на этот каталог увеличится на 1 за счет имени ".." в подкаталоге (например, tmp/subdir1/..):

1.11.4. Иерархия прав доступа

Рассмотрим более подробно, чему соответствуют девять символов в строке *атрибутов*, выдаваемой ls. Эти девять символов имеют вид "rwxrwxrwx", где некоторые "r", "w" и "x" могут

заменяться на "-". Очевидно, буквы отражают принятые в Linux три вида доступа – чтение, запись и использование – однако в *ярлыке* они присутствуют в трех экземплярах!

Дело в том, что любой пользователь (*процесс*) Linux по отношению к любому файлу может выступать в трех **ролях**: как *хозяин* (**u**ser), как член *группы*, которой принадлежит файл (**g**roup), и как *посторонний* (**o**ther), никаких отношений собственности на этот файл не имеющий. Строка *атрибутов* – это три тройки "rwx", описывающие *права доступа* к файлу *хозяина* этого файла (первая тройка, "u"), *группы*, которой принадлежит файл (вторая тройка, "g") и *посторонних* (третья тройка, "o"). Если в какой-либо тройке не хватает буквы, а вместо нее стоит "-", значит, пользователю в соответствующей роли будет в соответствующем виде доступа отказано.

При выяснении отношений между файлом и пользователем, запустившим *процесс*, роль определяется так:

1. Если *UID* файла совпадает с *UID процесса*, пользователь – *хозяин файла*
2. Если *GID* файла совпадает с *GID* **любой** *группы*, в которую входит пользователь, он – член *группы*, которой принадлежит файл.
3. Если ни *UID*, ни *GID* файла не пересекаются с *UID процесса* и списком *групп*, в которые входит запустивший его пользователь, этот пользователь – *посторонний*.

Именно в роли *хозяина* пользователь (*процесс*) может **изменять** *ярлык файла*. Единственное, чего не может делать *хозяин* со своим файлом – менять ему *хозяина*.

1.12. Использование прав доступа в Linux

1.12.1. Использование групп

В Linux определено несколько *системных групп*, задача которых – обеспечивать доступ членов этих *групп* к разнообразным ресурсам системы. Часто такие *группы* носят говорящие названия: "disk", "audio", "cdwriter" и т. п. Если обычным пользователям доступ к некоторому файлу, каталогу или специальному файлу Linux закрыт, он открыт членам *группы*, которой этот объект принадлежит.

Например, в Linux почти всегда используется *виртуальная файловая система* /proc – каталог, в котором в виде подкаталогов и файлов представлена информация из *таблицы процессов*. Имя подкаталога /proc совпадает с *PID* соответствующего *процесса*, а содержимое этого подкаталога отражает свойства *процесса*. *Хозяином* такого подкаталога будет хозяин *процесса* (с правами на чтение и использование), поэтому **любой** пользователь сможет посмотреть информацию о **своих** *процессах*. Именно каталогом /proc пользуется утилита ps. Использование утилиты ps для просмотра выполняющихся процессов Linux рассматривается в работе “Процессы ОС Linux“.

1.13. Суперпользователь

Суперпользователь - единственный пользователь в Linux, на которого не распространяются ограничения *прав доступа*. Имеет нулевой *идентификатор пользователя*.

Суперпользователь в Linux – это **выделенный** пользователь системы, на которого **не распространяются** ограничения *прав доступа*. *UID* суперпользовательских *процессов* равен 0: так система отличает их от *процессов* других пользователей. Именно суперпользователь имеет возможность произвольно изменять владельца и *группу* файла. Ему открыт доступ на чтение и запись к **любому файлу** системы и доступ на чтение, запись и использование к **любому каталогу**. Наконец, суперпользовательский *процесс* может на время сменить **свой собственный** *UID* с нулевого на любой другой. Именно так и поступает программа login, когда, проведя процедуру идентификации пользователя, запускает стартовый командный интерпретатор.

Среди *учетных записей* Linux всегда есть запись по имени root ("корень"), соответствующая нулевому идентификатору, поэтому вместо "суперпользователь" часто говорят "root". Множество системных файлов принадлежат root, множество файлов только ему

доступны на чтение или запись. Пароль этой учетной записи – одна из самых больших драгоценностей системы. Именно с ее помощью системные администраторы выполняют самую ответственную работу.

Существует два различных способа получить права суперпользователя. Первый – это зарегистрироваться в системе под этим именем, ввести пароль и получить *стартовую оболочку,* имеющую нулевой *UID*. Это – самый неправильный способ, пользоваться которым стоит, только если нельзя применить другие. В ОС Ubuntu описанный способ не используется, вместо него используется второй способ.

Второй способ —воспользоваться специальной утилитой sudo, которая позволяет выполнить одну или несколько команд от лица другого пользователя. По умолчанию эта утилита выполняет команду от лица пользователя root, то есть запускает командный интерпретатор с нулевым *UID*. Отличие от предыдущего способа в том, что всегда известно, кто именно запускал sudo, а значит, ясно, с кого спрашивать за последствия.

1.14. Поиск файлов

Для поиска файла по имени или его части используется утилита locate. Параметр задает имя файла. Для поиска без учета регистра служит ключ –i.

Для ограничения объема выводимой информации используется ключ –n число. Построчный вывод получается, если результаты поиска направить по конвейеру в программу less, например

locate mp3 | less

Утилита locate ведет поиск в базе данных, которая должна периодически обновляться утилитой updatedb, выполняемой с правами администратора.

Другой способ найти файл предоставляет утилита **find.** Ее ключи приведены в таблице 1.

Таблица 1. Ключи утилиты find

Ключ	Назначение
-name	Задает имя файла или его часть
-size	Задает размер файла, например 12k
-type	Задает тип объекта для поиска: f-обычный файл d-каталог l-символьная ссылка
-a	Логическая связка and
-o	Логическая связка or
-user	Задает имя пользователя

Достоинствами утилиты find являются независимость от базы данных и широкие функциональные возможности, недостаток – меньшая скорость поиска по сравнению с locate.

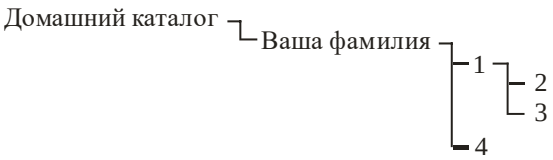
2.МЕТОДИКА ВЫПОЛНЕНИЯ

1. Ознакомиться с теоретическими сведениями.
2. После загрузки ОС Linux и запроса имени ввести имя и пароль пользователя.
3. По окончании загрузки ОС запустить терминал.

Все задания работы **следует** выполнить в режиме командной строки с помощью терминала.

Задания

1. Создать в домашнем каталоге следующую структуру подкаталогов (существующие каталоги не удалять!):



2. Скопировать файл **/etc/group** в каталоги **1**, 2, 3 и 4 используя абсолютные имена копируемого файла и каталога назначения.

3.С помощью утилиты file вывести на экран сведения о 3 - 4 различных файлах (в том числе из каталогов /bin и /dev).

4.Выполнить команду ls –l /dev

Используя таблицу 2 обозначений типов файлов

Таблица 2. Обозначения типов файлов

Символ	Тип файла
d	Каталог
l	Символьная ссылка
s	Сокет
b	Блочное устройство
c	Символьное устройство
p	Именованный канал

перечислить типы файлов, хранящихся в каталоге /dev

5.Используя справочную систему, ознакомиться с ключами утилиты ls -R, -1 (единица), -m, --color, ключи, определяющие порядок вывода на экран

6. Создать жесткую и символическую ссылки для одного из созданных в п.2 файлов.

Таблица 3 Индивидуальные задания

Номер варианта	Задание
1	1. Вывести список имен файлов из /var, используя ключ –l Список упорядочить по размерам файлов. 2. Найти файлы, имена которых оканчиваются на pdf
2	1. Вывести список имен файлов из /bin, используя ключ –l Список упорядочить по датам создания 2. Найти файлы, имена которых оканчиваются на jpg
3	1. Вывести список имен файлов из /sbin, используя ключ –l Список упорядочить по именам 2. Найти файлы, размеры которых превышают 25к (запись +25k)
4	1. Вывести список имен файлов из /tmp, используя ключ –l Список упорядочить по именам 2. Найти файлы, имена которых оканчиваются на text
5	1. Вывести список имен файлов из /usr, используя ключ –l Список упорядочить по размерам файлов. 2. Найти файлы, имена которых оканчиваются на jpg и размеры более 1к
6	1. Вывести список имен файлов из /bin, используя ключ –l Список упорядочить по датам создания 2. Найти файлы, размеры которых превышают 15к (запись +15k)
7	1. Вывести список имен файлов из /usr, используя ключ –l Список упорядочить по размерам файлов. 2. Найти файлы, размеры которых превышают 25к (запись +25k) и имена начинаются на s
8	1. Вывести список имен файлов из /var, используя ключ –l

	Список упорядочить по датам создания 2. Найти файлы, размеры которых превышают 25к (запись +25k) и имена начинаются на s, а заканчиваются на jpg
9	1. Вывести список имен файлов из /sbin, используя ключ –l Список упорядочить по размерам файлов 2. Найти файлы, размеры которых превышают 1M (запись +1m)
10	1. Вывести список имен файлов из /bin, используя ключ –l Список упорядочить по именам 2. Найти файлы, размеры которых превышают 5к (запись +5k)

3.ОТЧЕТ О РАБОТЕ

Готовится в письменном виде один на бригаду. Содержание отчета:

9. построенное в задании 1 дерево каталогов.
10. описания назначений ключей команды ls.
11. результаты выполнения заданий.

4.КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Типы файлов ОС Linux
2. Назначение утилиты file.
3. Структура дерева каталогов ОС Linux.
4. Отличия структуры файловых систем ОС Windows и Linux.
5. В чем отличие каталогов /var и /tmp.
6. Назначение утилиты pwd.
7. Назначение утилиты cat.
8. Назначение утилиты ls. Использование ключей –F, -a.
9. Утилита mkdir.
10. Утилиты копирования и перемещения файлов.
11. Жесткие ссылки: назначение и создание.
12. Создание файлов.
13. Символьные ссылки.
14. Удаление файлов и каталогов. Как восстановить ошибочно удаленный файл?
15. Назначение утилиты id.
16. Ярлыки объектов файловой системы.
17. Права доступа к файлу.
18. Суперпользователь и его права.
19. Назначение утилиты sudo.
20. Утилиты поиска файлов locate и find, их достоинства и недостатки.

Лабораторная работа 7. Контроль использования ресурсов ОС Linux (4 часа)

Цель работы – практическое знакомство с командами, используемыми для контроля использования ресурсов и виртуальной файловой системой /proc

1.1 Команды для контроля системных ресурсов

1.1.1 Вывод информации о процессах, выполняющихся в системе

Для вывода списка всех выполняющихся на компьютере в текущий момент процессах используется команда

ps aux

Значения используемых опций: a - all – процессы всех пользователей; u – ориентированная на пользователей (отображение информации о владельце); x – процессы, не контролируемые ttys.

Полезные ключи - -e – вывод сведений обо всех процессах и o – пользовательский вывод, например

ps –eo pid, ppid, pri, stat, pgid, nice, comm.

В столбце STAT содержится информация о состоянии процесса. Наиболее важные состояния: S-спящий; R –выполняющийся; T- остановленный; Z – зомби.

Буква Z указывает, что процесс завис и его нельзя завершить. Избавиться от подобной программы можно с помощью перезагрузки системы.

Более подробную информацию представляет опция –l (long format))

Для просмотра дерева процессов используются команды:

ps axjf

pstree

Завершение выполняющегося процесса

Завершение процесса выполняется командой

kill сигнал PID

Сначала процессу посылается сигнал -15. Если это не помогает, используется крайняя мера - посылается сигнал -9.

Отображение динамически обновляемого списка выполняющихся процессов – команда top

В отличие от ps, команда top представляет динамически обновляемые сведения о процессах и о том, какой объем системных ресурсов использует каждый из них (рис.1).

В первых пяти строках команда top отображает подробную информацию о системе, а затем описывает каждый выполняющийся процесс. Выходные данные сортируются по уровню загрузки ЦП данным процессом.

Для удаления процесса следует ввести символ k. Появится сообщение PID to kill:

Необходимо ввести PID процесса, программа запросит номер сигнала:

kill PID NNNN with signal [15]:

Как правило, данное значение является приемлемым, и через секунду после ввода Enter информация о выбранном процессе исчезнет с экрана.

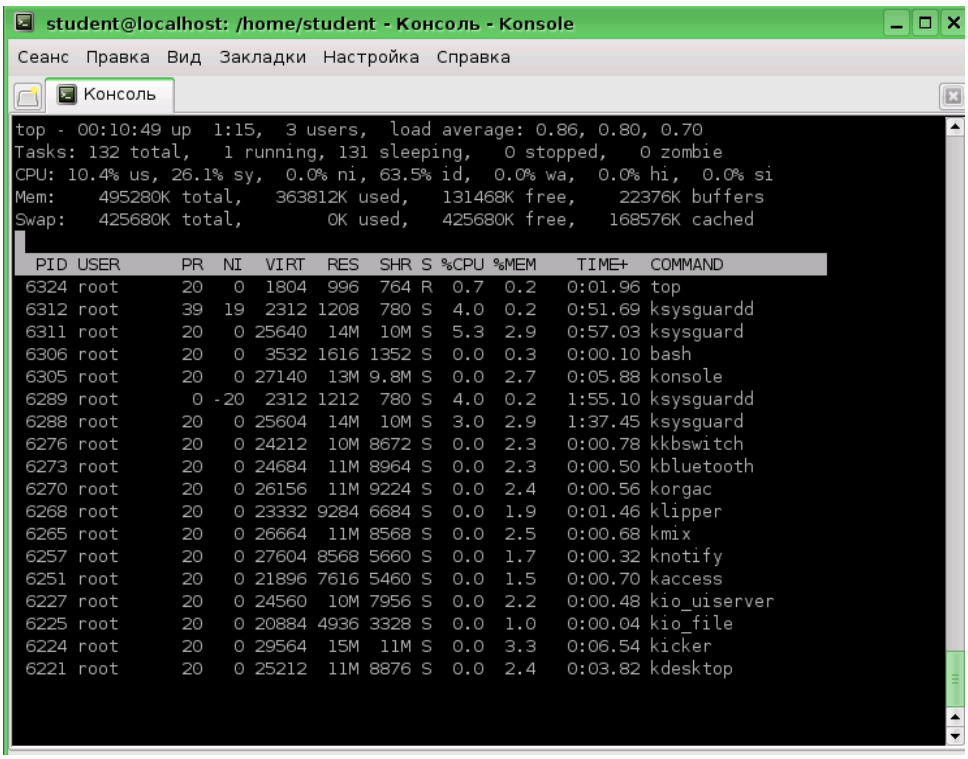


Рис. 1 – результат выполнение команды top

В первой строке программа сообщает текущее время, время работы системы (1 час 15 мин), количество зарегистрированных (login) пользователей (3 users), общая средняя загрузка системы (load average). Общей средней загрузкой системы называется среднее число процессов, находящихся в состоянии выполнения (R) или в состоянии ожидания (D). Общая средняя загрузка измеряется каждые 1, 5 и 15 минут.

Во второй строке вывода программы top сообщается, что в списке процессов находятся 132 процесса, из них 131 спит (состояние готовности или ожидания), 1 выполняется (на виртуальной машине только 1 процессор), 0 процессов зомби и 0 остановленных процессов.

В третьей-пятой строках приводится информация о загрузке процессора CPU в режиме пользователя и системном режиме, использования памяти и файла подкачки.

В таблице отображается различная информация о процессе. Рассмотрим колонки PID (идентификатор процесса), USER (пользователь, запустивший процесс), S (состояние процесса) и COMMAND (команда, которая была введена для запуска процесса).

Колонка S может содержать следующие значения:

R - процесс выполняется или готов к выполнению (состояние готовности)

D - процесс в "беспробудном сне" - ожидает дискового ввода/вывода

T - процесс остановлен (stopped) или трассируется отладчиком

S - процесс в состоянии ожидания (sleeping)

Z - процесс-зомби

N – процесс с низким приоритетом, nice, pri<19

< - процесс с высоким приоритетом, pri>19

+ - процесс в группе фоновых процессов

l – процесс с двумя и более потоками, многопоточный

s – ведущий процесс сеанса.

Колонка PR содержит приоритет процесса – целое число от 0 до 39. Колонка NI (NICE) (фактор уступчивости процесса) содержит задаваемое значение от -19 (наименее уступчивый) до 20 (самый уступчивый, вытесняется всеми). Значение NICE прибавляется к числу 20 для получения значения приоритета

PR=19+NICE

Для управления командой top используются односимвольные команды:

h – вывод справки о командах;

r – renice – изменение приоритета, в режиме администратора (через sudo) можно задавать значения от-19 до 20;

q – завершение работы с командой и другие (см. информацию справки).

Недостаток команды – вывод только первых N<26 строк информации о процессах.

Для управления процессами с использованием графического интерфейса используется утилита Системный монитор, которая запускается из системного меню Система-Администрирование – Системный монитор (рис.2).

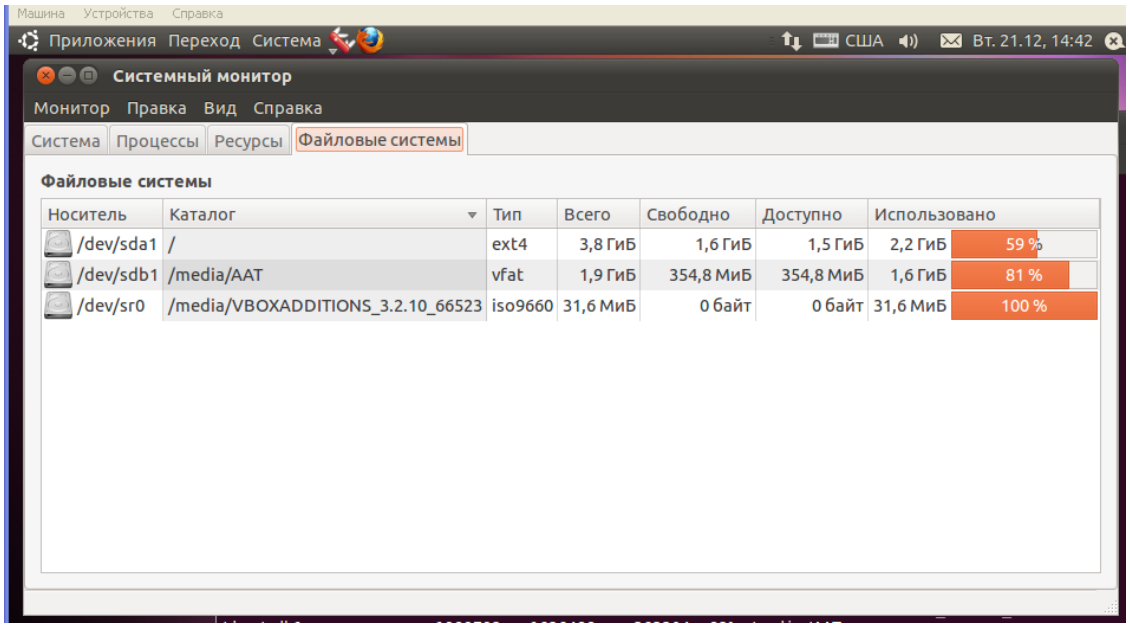


Рис. 2. Системный монитор

На закладке Процессы монитор показывает сведения о запущенных процессах, на закладке система – сведения об использовании ЦП и памяти. Достоинства монитора – использование графического интерфейса, отображение списка всех процессов.

Вывод дерева процессов

Для построения дерева процессов используются команды

pstree

ps –ejH

ps axjt

Получение информации о потоках

Как известно, процесс может иметь параллельно выполняющиеся потоки (threads) или облегченные процессы (LWP, Light Weight Process). Для получения информации о потоках заданного процесса используется опция –L, например ps –fLC swriter.bin выводит список потоков приложения writer Open Office. Процессы, использующие более одного потока – редактор

звуковых файлов audacity и soffice.bin, а также демоны (службы в по терминологии Windows). Как указано выше, многопоточные процессы помечено символом l в колонке состояния.

1.1.2 Получение списка открытых файлов

Команда lsof (List open files) без параметров выводит полный список открытых файлов. Пользователь-администратор получит несколько тысяч строк текста.

Для получения списка файлов, открытых конкретным пользователем, служит команда

lsof -u имя_пользователя

Получение списка пользователей конкретного файла

Необходимо ввести команду lsof с указанием имени файла. Например lsof /bin/bash

Отображение информации об оперативной памяти системы

Текущее состояние системной памяти позволяет получить команда **free**

По умолчанию все значения представлены в килобайтах. Значения в M позволяет получить опция –m.

1.1.3. Отображение информации об использовании дискового пространства

Команда df выводит данные об объеме доступного дискового пространства (в Кбайтах). Опция –h улучшает восприятие результатов.

Команда du дает возможность узнать объем дисковой памяти, занимаемой каталогами и файлами.

1.2 Файловая система /proc

Ядро Linux предоставляет механизм доступа к своим внутренним структурам и позволяет изменять установки ядра во время работы ОС посредством файловой системы /proc. Файловая система /proc является механизмом для ядра и его модулей, позволяющим посылать информацию процессам (отсюда и название /proc). С помощью этой виртуальной файловой системы можно работать с внутренними структурами ядра, получать полезную информацию о процессах и изменять установки (меняя параметры ядра) на лету. Файловая система /proc располагается в памяти в отличие от других файловых систем, которые располагаются на диске.

Файловая система /proc контролируется ядром. Из-за того, что она предоставляет информацию, контролируемую ядром, она располагается в памяти, контролируемой также ядром. Команда "ls -l" покажет, что большинство файлов в этой системе имеют нулевую длину, но посмотрев любой файл, Вы получите достаточно информации. Как это может быть? Все просто - файловая система /proc как любая другая файловая система регистрируется на уровне VFS (Virtual File System layer). Поэтому при запросе файлов/каталогов, файловая система /proc создает эти файлы/каталоги на основании информации, содержащейся в ядре.

В действительности многие программы собирают информацию из файлов в /proc, форматируют её своим собственным способом, а результат затем выводят на экран. Существует несколько программ, которые поступают именно так при выводе информации о процессах (top, ps и т. п.), /proc - это также хороший источник информации об аппаратном обеспечении, и по аналогии с

программами, показывающими процессы, некоторые другие программы являются просто интерфейсами к информации, находящейся в /proc.

Также существует специальный подкаталог /proc/sys. Он позволяет отображать параметры ядра и изменять их в режиме реального времени.

1.2.1 Информация о процессах

Каждый из каталогов содержит одинаковые пункты, краткое описание некоторых из них:

1. cmdline: этот (псевдо-) файл содержит полную командную строку, использованную для вызова процесса. Он не отформатирован: между программой и ее аргументами нет пробелов, а в конце строки нет разделителя строки. Чтобы просмотреть его, вы можете использовать: **perl -ple 's,\00, ,g' cmdline**.

2. cwd: эта символическая ссылка указывает на текущий рабочий каталог процесса (следует из имени).

3. environ: этот файл содержит все переменные окружения, определенные для этого процесса, в виде ПЕРЕМЕННАЯ=значение. Как и в cmdline вывод вообще не отформатирован: нет разделителей строк для отделения различных переменных, и в конце нет разделителя строки. Единственным решением для его просмотра будет: **perl -pl -e 's,\00,\n,g' environ**.

4. exe: эта символическая ссылка указывает на исполняемый файл, соответствующий запущенному процессу.

5. fd: этот подкаталог содержит список файловых дескрипторов, открытых в данный момент процессом.

6. maps: когда вы выводите содержимое этого именованного канала (при помощи команды **cat**, например), вы можете увидеть части адресного пространства процесса, которые в текущий момент распределены для файла. Вот эти поля (слева направо): адресное пространство, связанное с этим распределением; разрешения, связанные с этим распределеним; смещение от начала файла, где начинается распределение; старший и младший номера (в шестнадцатиричном виде) устройства, на котором находится распределенный файл; номер inode файла; и, наконец, имя самого файла.

7. root: эта символическая ссылка указывает на корневой каталог, используемый процессом. Обычно это будет /.

8. status: этот файл содержит разнообразную информацию о процессе: имя исполняемого файла, его текущее состояние, его PID и PPID, его реальные и эффективные UID и GID, его использование памяти и другие данные.

Если вывести список содержимого каталога fd для процесса 127, получим примерно следующее:

```
ls -l fd
total 0
lrwx------    1 root     root          64 Dec 16 22:04 0 -> /dev/console
l-wx------    1 root     root          64 Dec 16 22:04 1 -> pipe:[128]
l-wx------    1 root     root          64 Dec 16 22:04 2 -> pipe:[129]
l-wx------    1 root     root          64 Dec 16 22:04 21 -> pipe:[130]
lrwx------    1 root     root          64 Dec 16 22:04 3 -> /dev/apm_bios
lr-x------    1 root     root          64 Dec 16 22:04 7 -> pipe:[130]
lrwx------    1 root     root          64 Dec 16 22:04 9 -> /dev/console
```

На самом деле это список файловых дескрипторов, открытых процессом. Каждый открытый дескриптор представлен в виде символической ссылки, где имя - это номер дескриптора, который указывает на файл, открытый этим дескриптором.

6.2. Информация об аппаратном обеспечении

Кроме каталогов, связанных с различными процессами, в /proc также содержится значительный объём информации об аппаратном обеспечении ПК.

Список файлов каталога /proc, полученный с помощью команды ls -d [a-z]*
выглядит следующим образом:

acpi
asound
buddyinfo
bus
cmdline
cpuinfo
crypto
devices
diskstats
dma
driver
execdomains
fb
filesystems
fs

ide
interrupts
iomem
ioports
irq
kallsyms
kcore
keys
key-users
kmsg
loadavg
locks
mdstat
meminfo
misc
modules

mounts
net
partitions
schedstat
self
slabinfo
stat
swaps
sys
sysrq-trigger
sysvipc
tty
uptime
version
vmstat
zoneinfo

Например, каталог /proc/interrupts содержит список прерываний, используемых в данный момент системой, а также периферийные устройства, которые их используют.

Описание некоторых из файлов /proc:

cpuinfo: этот файл содержит, как видно из его имени, информацию о процессорах машины. Пример содержимого файла:

cat /proc/cpuinfo

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 8
model name      : Pentium III (Coppermine)
stepping        : 6
cpu MHz         : 1000.119
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
sep_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 mmx fxsr xmm
bogomips        : 1998.85
processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 8
model name      : Pentium III (Coppermine)
stepping        : 6
cpu MHz         : 1000.119
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
sep_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 mmx fxsr xmm
bogomips        : 1992.29
```

modules: этот файл содержит список модулей, используемых ядром в настоящий момент, вместе со счетчиком использования каждого из модулей. Эта информация используется командой lsmod, которая отображает её в более удобной для чтения форме,

meminfo: этот файл содержит информацию о загрузке памяти на момент вывода его содержимого. Команда free выведет ту же самую информацию, но уже в более удобном для чтения формате.

bus: этот подкаталог содержит информацию обо всех периферийных устройствах, найденных на различных шинах вашего компьютера. Информация обычно не удобна для чтения, и большая её часть переформатируется внешними утилитами.

acpi: некоторые файлы и каталоги, представленные в этом каталоге, особенно интересны для ноутбуков, которые позволяют вам выбирать различные варианты энергосбережения.

6.3. Отображение и изменение параметров ядра

Назначение подкаталога /proc/sys - сообщать о различных параметрах ядра, и позволить изменять некоторые из них в интерактивном режиме, В противоположность всем другим файлам каталога /proc, некоторые файлы из этого каталога могут быть открыты для записи, но только для root'a.

Содержимое этих каталогов зависит от системы, а большинство файлов будет полезно только для очень специализированных приложений.

МЕТОДИКА ВЫПОЛНЕНИЯ

1. Вывести список всех процессов системы.
2. Вывести дерево процессов.
3. С помощью команды top получить список 5 процессов, потребляющих наибольшее количество процессорного времени.
4. Найти 2 процесса, имеющих более ДВУХ потоков. Использовать состояние процесса l.
5. Используя команду top, изменить приоритеты 2 процессов.
6. Получить список открытых файлов пользователя aa
7. Получить текущее состояние системной памяти
8. Получить справку об использовании дискового пространства.
9. Вывести информацию о каком-либо процессе, используя содержимое каталога /proc
10. Вывести информацию о процессоре ПК, используя содержимое каталога /proc
11. Вывести список модулей, используемых в настоящий момент ядром ОС.

2. ОТЧЕТ О РАБОТЕ

Содержание отчета:

1. Результаты выполнения заданий 1- 11 (снимки экранов) и использованные команды ОС Linux.

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Команды вывода списка процессов.
2. Команда получения списка потоков
3. Команда для завершения приложений.
4. Состояния процесса Linux.
5. Получение информации о потоках процесса.
6. Примеры многопоточных процессов.
7. Необходимость использования потоков.
8. Процессы – зомби: как они появляются, как их найти и что с ними делать?
9. Содержимое вывода команды top.
10. Как получить информацию о процессах системы, используя файловую систему /proc?

11. Команды для получения информации об открытых файлах
12. Получение информации о состоянии системной памяти.
13. Получение информации об использовании дискового пространства.
14. Назначение файловой системы /proc

Лабораторная работа № 8 Обработка строк (работа с текстовыми данными) (4 часа)

Цель работы – практическое знакомство со способами эффективной обработки текста при помощи интерфейса командной строки и набора стандартных утилит

1.1 Ввод и вывод. Перенаправление ввода и вывода

Каждая программа работает с данными определенного типа: текстовыми, графическими, звуковыми и т. п. Основной интерфейс управления системой в Linux - это *терминал*, который предназначен для передачи текстовой информации от пользователя системе и обратно. Поскольку ввести с *терминала* и вывести на *терминал* можно только текстовую информацию, то ввод и вывод программ, связанных с терминалом, тоже должен быть текстовым. Однако необходимость оперировать с текстовыми данными не ограничивает возможности управления системой, а, наоборот, расширяет их. Пользователь может прочитать вывод любой программы и проанализировать, что происходит в системе, а разные программы оказываются совместимыми между собой, поскольку используют один и тот же вид представления данных - текстовый.

"Текстовость" данных - всего лишь договоренность об их формате. Никто не мешает выводить на экран нетекстовый файл, однако пользы в этом будет мало. Во-первых, раз уж файл содержит не текст, то не предполагается, что пользователь сможет что-либо понять из его содержимого. Во-вторых, если в нетекстовых данных, выводимых на *терминал*, случайно встретится *управляющая последовательность*, *терминал* ее выполнит. Например, если в скомпилированной программе записано некоторое число в виде четырех байтов: 27, 91, 49 и 74, соответствующий им **текст** состоит из четырех *символов* ASCII: "esc", "[", "1" и "J", и при *выводе* файла на виртуальную консоль Linux в этом месте выполнится очистка экрана, так как "^[[1J" - именно такая *управляющая последовательность* для виртуальной консоли. Не все *управляющие последовательности* столь безобидны, поэтому использовать нетекстовые данные в качестве текстов не рекомендуется.

Если содержимое нетекстового файла все-таки желательно просмотреть (то есть превратить в **текст**), можно воспользоваться утилитой **hexdump** с ключом **-C**, которая выдает содержимое файла в виде шестнадцатеричных ASCII-кодов, или **strings,** показывающей только те части файла, которые **могут** быть представлены в виде текста:

Пример 7.1. Использование hexdump

```
[student@localhost root]$ hexdump -C /bin/cat | less
 . . .
[student @localhost root]$ strings -n3 /bin/cat | less
```

В приведенном примере 7.1 утилита hexdump с ключом "-C" выводит в правой стороне экрана текстовое представление данных, заменяя непечатные *символы* точками (чтобы среди выводимого текста

не встретилось *управляющей последовательности*). Наименьшая длина строки передается strings ключом "-n".

Для того чтобы записать данные в файл или прочитать их оттуда, процессу необходимо сначала открыть этот файл (при открытии на запись, возможно, придется предварительно создать его). При этом процесс получает **дескриптор** (описатель) открытого файла - уникальное для этого процесса число, которое он и будет использовать во всех операциях записи. Первый открытый файл получит дескриптор 0, второй - 1 и так далее. Закончив работу с файлом, процесс закрывает его, при этом дескриптор освобождается и может быть использован повторно. Если процесс завершается, не закрыв файлы, за него это делает система. Строго говоря, только в операции открытия дескриптора указывается, какой именно файл будет задействован. В качестве "файла" используются и обычные файлы, и устройства (чаще всего - *терминалы*), и *каналы*. Дальнейшие операции - чтение, запись и закрытие - работают с дескриптором, как с потоком данных, а куда именно ведет этот поток, неважно.

Каждый процесс Linux получает при старте три "файла", открытых для него системой. Первый из них (дескриптор 0) открыт на чтение, это стандартный *ввод* процесса. Именно со стандартным *вводом* работают все операции чтения, если в них не указан дескриптор файла. Второй (дескриптор 1) - открыт на **запись**, это *стандартный вывод* процесса. С ним работают все операции записи, если дескриптор файла не указан в них явно. Наконец, третий поток данных (дескриптор 2) предназначается для *вывода диагностических сообщений*, он называется ***стандартный вывод ошибок***. Поскольку эти три дескриптора уже открыты к моменту запуска процесса, первый файл, открытый **самим** процессом, будет, скорее всего, иметь дескриптор 3.

Дескриптор - это описатель потока данных, открытого процессом. Дескрипторы нумеруются, начиная с 0. При открытии нового потока данных его дескриптор получает наименьший из неиспользуемых в этот момент номеров. Три заранее открытых дескриптора - стандартный *ввод* (0), *стандартный вывод* (1) и *стандартный вывод ошибок* (2) - выдаются при запуске.

Механизм копирования *окружения* подразумевает, в числе прочего, копирование всех открытых дескрипторов родительского процесса дочернему. В результате и родительский, и дочерний процесс имеют под одинаковыми дескрипторами одни и те же потоки данных. Когда запускается *стартовый командный интерпретатор*, все три заранее открытых дескриптора связаны у него с *терминалом* (точнее, с соответствующим устройством типа tty): пользователь вводит команды с клавиатуры и видит сообщения на экране. Следовательно, любая команда, запускаемая из командной оболочки, будет выводить на тот же *терминал*, а любая команда, запущенная интерактивно (не в фоне) - вводить оттуда.

Стандартный вывод

Стандартный вывод (standard output, stdout) - это поток данных, открываемый системой для каждого процесса в момент его запуска и предназначенный для данных, выводимых процессом.

Некоторые утилиты умеют выводить не только на *терминал*, но и в файл. Например, `info` при указании ключа "`-o`" с именем файла выведет текст руководства в файл, вместо того, чтобы отображать его на мониторе. Даже если разработчиками программы не предусмотрен такой ключ, известен и другой способ сохранить *вывод* программы в файле вместо того, чтобы выводить его на монитор: поставить знак ">" и указать после него имя файла.

Подмена *стандартного вывода* - задача командной оболочки (shell). Shell создает пустой файл, имя которого указано после знака ">", и дескриптор этого файла передается программе под номером 1

(*стандартный вывод*). Делается это очень просто. При запуске программы из оболочки после выполнения fork() появляется два одинаковых процесса, один из которых - дочерний - должен запустить вместо себя команду (выполнить exec()). Перед этим он **закрывает стандартный вывод** (дескриптор 1 освобождается) и **открывает** файл (с ним связывается первый свободный дескриптор, т. е. 1), а запускаемой команде ничего знать и не надо: ее *стандартный вывод* уже подменен. Эта операция называется **перенаправлением стандартного вывода**. В том случае, если файл уже существует, shell запишет его заново, полностью уничтожив все, что в нем содержалось до этого. Поэтому, чтобы продолжить записывать данные в textfile, потребуется другая операция - ">>":

Стандартный ввод

Стандартный ввод (standard input, stdin) - поток данных, открываемый системой для каждого процесса в момент его запуска и предназначенный для *ввода* данных.

Для передачи данных на вход программе может быть использован *стандартный ввод* (сокращенно - stdin). При работе с командной строкой *стандартный ввод* - это *символы*, вводимые пользователем с клавиатуры. *Стандартный ввод* можно **перенаправить** при помощи командной оболочки, подав на него данные из некоторого файла. *Символ* "<" служит для перенаправления содержимого файла на *стандартный ввод* программе. Например, если вызвать утилиту sort без параметра, она будет читать строки со *стандартного ввода*. Команда "sort < имя_файла" подаст на *ввод* sort данные из файла:

Результат действия этой команды аналогичен команде sort textfile - разница лишь в том, что когда используется "<", sort получает данные со *стандартного ввода* ничего не зная о файле "textfile", откуда они поступают. Механизм работы shell в данном случае тот же, что и при перенаправлении *вывода*: shell читает данные из файла "textfile", запускает утилиту sort и передает ей на *стандартный ввод* содержимое файла.

Необходимо помнить, что операция ">" **деструктивна**: она всегда создает файл нулевой длины. Поэтому для, допустим, сортировки данных **в файле** надо применять последовательно sort < файл > новый_файл и mv новый_файл файл. Команда вида команда < файл > тот_же_файл просто урежет его до нулевой длины!

Стандартный вывод ошибок

Для *диагностических сообщений*, информирующих пользователя о ходе выполнения работы: а также для сообщений об ошибках, возникших в ходе выполнения программы, в Linux предусмотрен *стандартный вывод ошибок* (сокращенно - stderr).

Стандартный вывод ошибок (standard error, stderr) - поток данных, открываемый системой для каждого процесса в момент его запуска и предназначенный для *диагностических сообщений*, выводимых процессом.

Использование *стандартного вывода ошибок* наряду со *стандартным выводом* позволяет отделить собственно результат работы программы от разнообразной сопровождающей информации, например, направив их в разные файлы. Стандартный вывод ошибок может быть перенаправлен так же, как и *стандартный ввод/вывод*, для этого используется комбинация *символов* "2>", например info cat > cat.info 2> cat.stderr

На терминал в этом случае ничего не попадет - *стандартный вывод* отправится в файл cat.info, *стандартный вывод ошибок* - в cat.stderr. Вместо ">" и "2>" можно было бы написать "1>" и "2>". Цифры

в данном случае обозначают номера дескрипторов открываемых файлов. Если некая утилита ожидает получить открытый дескриптор с номером, допустим, 4, то, для того чтобы ее запустить, **обязательно** потребуется использовать сочетание "4>".

Иногда, однако, требуется объединить *стандартный вывод* и *стандартный вывод ошибок* в одном файле, а не разделять их. В командной оболочке bash для этого имеется специальная последовательность "2>&1". Это означает "направить *стандартный вывод ошибок* туда же, куда и *стандартный вывод*":

Перенаправление в никуда

Иногда заведомо известно, что какие-то данные, выведенные программой, не понадобятся. Например, предупреждения со *стандартного вывода ошибок*. В этом случае можно перенаправить *стандартный вывод ошибок* на устройство, специально предназначенное для уничтожения данных - /dev/null. Все, что записывается в этот файл, просто будет выброшено и **нигде не сохранится**:

Пример 7.2. Перенаправление в /dev/null

```
[student@localhost root]$ info cat > cat.info 2> /dev/null
```

Точно таким же образом можно избавиться и от *стандартного вывода*, отправив его в /dev/null.

1.2. Обработка данных в потоке. Конвейер

Нередко возникают ситуации, когда нужно обработать *вывод* одной программы какой-то другой программой. Для решения подобной задачи в bash предусмотрена возможность перенаправления *вывода* можно не только в файл, но и **непосредственно** на *стандартный ввод* другой программе. В Linux такой способ передачи данных называется ***конвейер***.

В bash для перенаправления *стандартного вывода* на *стандартный ввод* другой программе служит *символ* "|". Самый простой и наиболее распространенный случай, когда требуется использовать *конвейер*, возникает, если *вывод* программы не умещается на экране монитора и очень быстро "пролетает" перед глазами, так что человек не успевает его прочитать. В этом случае можно направить *вывод* в программу просмотра (less).

Можно последовательно обработать данные несколькими разными программами, перенаправляя *вывод* на *ввод* следующей программе и организуя сколь угодно длинный *конвейер* для обработки данных. В результате получаются командные строки вида "cmd1 | cmd2 | ... | cmdN".

Организация *конвейера* устроена в shell по той же схеме, что и перенаправление в файл, но с использованием особого объекта системы - *канала*. Можно представить трубу, немедленно доставляющую данные от входа к выходу (английский термин - "pipe"). *Каналом* пользуются сразу два процесса: один пишет туда, другой читает. Связывая две команды *конвейером*, shell открывает *канал* (заводится **два** дескриптора - входной и выходной), подменяет по уже описанному алгоритму *стандартный вывод* первого процесса на входной дескриптор *канала*, а *стандартный ввод* второго процесса - на выходной дескриптор *канала*. После чего остается запустить по команде в этих процессах, и *стандартный вывод* первой попадет на *стандартный ввод* второй.

Канал (pipe) - неделимая пара дескрипторов (входной и выходной), связанных друг с другом таким образом, что данные, записанные во входной дескриптор, будут немедленно доступны на чтение с выходного дескриптора.

1.3 Фильтры

Если программа и вводит данные, и выводит, то ее можно рассматривать как трубу, в которую что-то входит и из которой что-то выходит. Обычно смысл работы таких программ заключается в том, чтобы определенным образом **обработать** поступившие данные. В Linux такие программы называют **фильтрами**: данные проходят через них, причем что-то "застревает" в *фильтре* и не появляется на выходе, а что-то изменяется, что-то проходит сквозь *фильтр* неизменным. *Фильтры* в Linux обычно по умолчанию читают данные со *стандартного ввода*, а выводят на *стандартный вывод*. Простейший *фильтр* - программа cat: собственно, никакой "фильтрации" данных она не производит, она просто копирует *стандартный ввод* на *стандартный вывод*.

Данные, проходящие через *фильтр*, представляют собой текст: в стандартных потоках *ввода-вывода* все данные передаются в виде *символов*, строка за строкой, как и в *терминале*. Поэтому могут быть состыкованы при помощи *конвейера ввод* и *вывод* любых двух программ, поддерживающих стандартные потоки *ввода-вывода*.

В любом дистрибутиве Linux присутствует набор стандартных утилит, предназначенных для работы с файловой системой и обработки текстовых данных. Это who, cat, ls, pwd, cp, chmod, id, sort и др. Каждая из этих утилит предназначена для исполнения какой-то **одной** операции над файлами или текстом: *вывод* списка файлов в каталоге, копирование, сортировка строк, хотя каждая утилита может выполнять свою функцию по-разному, в зависимости от переданных ей ключей и параметров. При этом все они ориентированы на работу с данными в текстовой форме, многие являются *фильтрами*, не имеют графического интерфейса, вызываются из командной строки и работают со стандартными потоками *ввода*/*вывода*, поэтому хорошо приспособлены для построения *конвейеров*.

1.4 Структурные единицы текста

Работу в системе Linux почти всегда можно представить как работу с текстами. Поиск файлов и других объектов системы - это получение от системы текста **особой** структуры - списка имен. Операции над файлами - создание, переименование, перемещение, а также сортировка, перекодировка и прочее - замену одних *символов* и строк другими либо в каталогах, либо в самих файлах. Работая с текстом в Linux, нужно принимать во внимание, что текстовые данные, передаваемые в системе, структурированы. Большинство утилит обрабатывает не непрерывный поток текста, а последовательность **единиц**. В текстовых данных в Linux выделяются следующие структурные единицы:

1. **Строки**

Строка - основная единица передачи текста в Linux. *Терминал* передает данные от пользователя системе строками (командная строка), множество утилит вводят и выводят данные построчно, при работе многих утилит одной строке соответствует один объект системы (имя файла, путь и т. п.), sort сортирует строки. Строки разделяются *символом* конца строки "\n" (newline).

2. **Поля**

В одной строке может упоминаться и больше одного объекта. Если понимать объект как последовательность *символов* из определенного набора (например, букв), то строку можно рассматривать как состоящую из слов и разделителей. В этом случае текст от начала строки до первого *разделителя* - это первое *поле*, от первого *разделителя* до второго - второе *поле* и т. д. В качестве *разделителя* можно рассматривать любой *символ*, который не может использоваться в объекте. Например, если в пути "/home/student" *разделителем* является *символ* "/", то первое *поле* пусто, второе содержит слово "home",

третье - "student". Некоторые утилиты позволяют выбирать из строк отдельные *поля* (по номеру) и работать со строками как с таблицей.

3. **Символы**

Минимальная единица текста - *символ*. ***Символ*** - это одна буква или другой письменный знак. Стандартные утилиты Linux позволяют заменять одни *символы* другими (производить транслитерацию), искать и заменять в строках *символы* и комбинации *символов*.

Символ конца строки в кодировке ASCII совпадает с *управляющей последовательностью* "^J" - "перевод строки", однако в других кодировках он может быть иным. Кроме того, на большинстве *терминалов* - но не на всех! - вслед за переводом строки необходимо выводить еще *символ* возврата каретки ("^M"). Это вызвало путаницу: некоторые системы требуют, чтобы в конце текстового файла стояли **оба этих символа** в определенном порядке. Чтобы избежать путаницы, в Linux было принято единственно верное решение: содержимое файла соответствует кодировке, а при *выводе* на *терминал* концы строки преобразуются в *управляющие последовательности* согласно настройке *терминала*.

В распоряжении пользователя Linux есть ряд утилит, выполняющих элементарные операции с единицами текста: поиск, замену, разделение и объединение строк, *полей*, *символов*. Эти утилиты, как правило, имеют одинаковое представление о том, как определяются единицы текста: что такое строка, какие *символы* являются *разделителями* и т. п. Во многих случаях их представления можно изменять при помощи настроек. Поэтому такие утилиты легко взаимодействуют друг с другом. Комбинируя их, можно автоматизировать довольно сложные операции по обработке текста.

1.5 Регулярные выражения

Регулярными выражениями называются особым образом составленные наборы символов, выделяющие из текста нужное сочетание слов или символов, которое соответствует признакам, отраженным в регулярном выражении. Иными словами, регулярное выражение — это **фильтр для текста**.

В Linux регулярные выражения используются командой grep, которая позволяет искать файлы с определенным содержанием либо выделять из файлов строки с необходимым содержимым (например, номера телефонов, даты и т. д.). Многие программы, так или иначе работающие с текстом, (текстовые редакторы), поддерживают *регулярные выражения*. К таким программам относятся два "главных" текстовых редактора Linux - Vim и Emacs. Однако нужно учитывать, что в разных программах используются разные диалекты языка *регулярных выражений*, где одни и те же понятия имеют разные обозначения, поэтому **всегда** нужно обращаться к руководству по конкретной программе.

1.5.1 Элементарные регулярные выражения

Элементарная структурная единица регулярного выражения — это символ. Текст можно искать по определенному набору букв и цифр. Рассмотрим пример, в котором с помощью регулярного выражения выделим из данных строк те, которые содержат сочетание букв bc (именно в этом порядке):

Исходный набор строк:
abc
abcd
dcba
adbc
Регулярное выражение:

bc
Результат:
abc
abcd
adbc

1.5.2. Конструкция вида [...]

Рассмотрим другой пример, заменив некоторые буквы в строках на заглавные:
Исходный набор строк:
abC
abcd
dcba
adBc
Регулярное выражение:
bc
Результат: abcd

Результат обработки строк с помощью регулярного выражения изменился. При использовании вышеуказанного регулярного выражения в большинстве случаев чаще всего различают строчные и прописные буквы, что логически обосновано, если не указан соответствующий параметр, предписывающий не различать их. Программы, которые работают с регулярными выражениями чаще всего различают строчные и прописные буквы, если не указана соответствующая опция, предписывающая не различать строчные и прописные буквы.

В результате получается всего одна строка — вторая. Для вывода трех строк, как в первом примере, понадобится **особая конструкция**. Рассмотрим ее.

В основе данной конструкции лежат две квадратные скобки (открывающая и закрывающая), внутри которых расположены символы либо конструкции (последний случай будет описан далее), один из которых может быть на месте этой конструкции в итоговом выражении. Изменим регулярное выражение в предыдущем примере. Теперь задачей является сделать это регулярное выражение более универсальным, чтобы с его помощью можно было найти в исходном наборе строк сочетание bc независимо от того, в каком регистре находятся буквы в конечных выражениях.

Исходный набор строк:
abC
abcd
dcba
adBc
Регулярное выражение:
[Bb][Cc]
Результат:
abC
abcd
adBc

Теперь рассмотрим такой пример: с помощью регулярного выражения необходимо выделить из указанных в предыдущем примере строк те, которые содержат некоторую букву английского алфавита в нижнем регистре и сразу за ней — букву с в нижнем или верхнем регистре. При применении способа перечисления для решения поставленной задачи получится следующее регулярное выражение:

[abcdefghijklmnopqrstuvwxyz] [Cc]

Это верно, но строка получилась длинной, что особенно неудобно при составлении больших регулярных выражений. В подобных случаях можно перечислить все эти 26 символов короче, используя интервалы, то есть указать начальный и конечный символы, поставив между ними знак «тире». Рассмотрим пример:

Исходный набор строк:
abC
abcd
dcba
adBc
Регулярное выражение:
[a-z][Cc]
Результат:
abC
abcd
dcba

Здесь a-z — это и есть нужный интервал. Можно изменить пример так, чтобы первый символ мог быть как строчным, так и прописным, для чего сразу после первого интервала указываем второй:

Исходный набор строк:
abC
abcd
dcba
adBc
Регулярное выражение:
[a-zA-Z][Cc]
Результат:
abC
abcd
dcba
adBc

То же касается и цифр. Границами интервала могут быть любые символы, но последовательности типа [z-a] и [5-1] смысла иметь не будут, так как ASCII-код первого символа должен быть меньше либо равен коду завершающего. По поводу интервалов с цифрами следует напомнить, что символ нуля идет раньше символов всех остальных цифр. Количество стоящих рядом последовательностей неограниченно.

В этой же конструкции можно обратить (или, как еще говорят, инвертировать) выбор символов, поставив после знака открывающей квадратной скобки символ ^, после чего на месте конструкции будут предполагаться все символы, кроме указанных в ней самой. Рассмотрим это на предыдущем примере.

Из данного набора строк выделим только те, в которых буква b стоит перед любым символом, кроме буквы c.

Исходный набор строк:
abC
abcd
dcba
adBc
Регулярное выражение:
[Bb][^Cc]
Результат:
dcba

1.5.3. Метасимволы

Не все символы можно использовать прямо по назначению. Посмотрите, например, на конструкцию, которая описывалась в предыдущем разделе. Допустим, требуется найти в каком-то файле строки, содержащие следующий набор символов: abc[def. Можно предположить, что регулярное выражение будет составлено по принципам, описанным выше, но это неверно. Открывающая квадратная скобка — это один из символов, который несет для программы, работающей с регулярными выражениями, особый смысл (который был рассмотрен ранее). Такие символы называются **метасимволами**.

Метасимволы бывают разными и служат для различных целей. Из предыдущего материала можно выделить метасимволы открывающей и закрывающей квадратных скобок, а также символ ^. Символ «тире» не рассматривается как метасимвол, так как он имеет особое значение только внутри конструкции с квадратными скобками, а вне такой конструкции специально не применяется.

Рассмотрим те метасимволы, которые предполагают, что в конечном выражении на их месте будет стоять какой-либо символ или символы (табл. 7.1).

Таблица 7.1. Знакозаменяющие метасимволы

Метасим-вол	Описание метасимвола
.(точка)	Предполагает, что в конечном выражении на ее месте будет стоять любой символ. Продемонстрируем это на примере набора английских слов: **Исходный набор строк:** wake make machine cake maze **Регулярное выражение:** ma.e **Результат:** make maze
\w	Замещает любые символы, которые относятся к буквам, цифрам и знаку подчеркивания. Пример: **Исходный набор строк:** abc a$c a1c a c **Регулярное выражение:** a\wc **Результат:** abc a1c
\W	Замещает все символы, кроме букв, цифр и знака подчеркивания (то есть является обратным метасимволу \w). Пример: **Исходный набор строк:** abc a$c a1c a c **Регулярное выражение:** a\Wc

	Результат: a$c a c
\d	Замещает все цифры. Продемонстрируем его действие на том же примере: **Исходный набор строк:** abc a$c a1c a c **Регулярное выражение:** a\dc **Результат:** a1c
\D	Замещает все символы, кроме цифр, например: **Исходный набор строк:** abc a$c a1c a c
\D	**Регулярное выражение:** a\Dc **Результат:** abc a$c a c
[\b]	Замещает символ перевода курсора на один влево (возврат курсора)
\r	Замещает символ перевода курсора в начало строки
\n	Замещает символ переноса курсора на новую строку
\t	Замещает символ горизонтальной табуляции
\v	Замещает символ вертикальной табуляции
\f	Замещает символ перехода на новую страницу
\s	Равнозначен использованию пяти последних метасимволов, то есть вместо метасимвола \s можно написать [\r\n\t\v\f], что, однако, не так удобно
\S	Является обратным метасимволу \ s

Для лучшего понимания рассмотрим, что такое символ перевода курсора в начало строки, переноса курсора на новую строку и т. д. В конце каждой строки находятся один или два символа, которые указывают на необходимость перехода на новую строку. Начиная с операционной системы MS-DOS и до настоящего времени в операционных системах Microsoft используются два символа для обозначения переноса строки — сам символ переноса и символ возврата курсора в начало строки (заменяются метасимволами \n и \r соответственно), хотя в отдельности эти символы почти не применяются. В Linux используется только символ перехода на новую строку (заменяется метасимволом \n). Это следует учесть при составлении регулярных выражений.

Рассмотрим интересную группу символов, для чего поставим следующую задачу. Количество символов, которые должны быть в конечном тексте, не всегда известно (примером может быть распознавание имени веб-сайта), поэтому при помощи вышеописанных конструкций и метасимволов написать действительно универсальные регулярные выражения невозможно. Рассмотрим еще одну группу символов, которые помогут решить подобные проблемы. Они используются сразу после символа, метасимвола либо конструкции, количество вхождения которых они должны описать (табл. 7.2).

Таблица 7.2. Метасимволы количества повторений

Символ	Описание метасимвола

?	Указывает обработчику регулярных выражений на то, что предыдущий символ, метасимвол или конструкция могут вообще не существовать в конечном тексте либо присутствовать, но иметь не более одного вхождения. Рассмотрим пример. Из данного набора строк требуется найти только те, в которых символу с может (но не обязательно) предшествовать один символ a, перед чем должен стоять символ b*:* **Исходный набор строк:** acbd aabc caab ecad bcde abac **Регулярное выражение:** b[Aa]?c **Результат:** aabc bcde abac
*****	Означает, что впереди стоящие символ, метасимвол либо конструкция могут как отсутствовать, так и быть в конечном выражении, причем количество вхождений неограниченно. **Пример**: из данного набора строк выделим те, в которых есть по крайней мере две буквы a, между которыми может быть либо отсутствовать некоторое количество цифр: **Исходный набор строк:** a123a a12a al23b alc3b b12a aaa **Регулярное выражение:** [Aa]\d*[Aa] **Результат:** al23a al2a aaa
+	Действие схоже с действием предыдущего символа с тем отличием, что впередистоящие метасимвол, символ или конструкция должны повторяться как минимум один раз. Рассмотрим предыдущий пример, но чтобы между буквами a была хотя бы одна цифра: **Исходный набор строк:** a123a a12a al23b alc3b b12a aaa **Регулярное выражение:** [Aa]\d+[Aa] **Результат:** a123a a12a
{min, max}	Иногда первых трех способов указания количества вхождений бывает недостаточно, так как они описывают количество не детально. Решить проблему можно следующим способом. Для указания количества вхождений символа либо конструкции после них ставят открывающие фигурные скобки и пишут минимальное количество вхождений. Если это количество фиксированное (то есть должно быть не больше и не меньше вхождений символа), то скобку закрывают; если должно быть не меньше указанного количества вхождений, то ставят запятую и закрывают скобку; если существует предельное количество вхождений, то после запятой указывают его и закрывают скобку. Так, эквивалентом знаку вопроса является конструкция {0,1}, знаку звездочки — { 0, }, знаку «плюс» — {1, }. Рассмотрим пример: **Исходный набор строк:** a123a

	a12a al23b alc3b b12a aaa **Регулярное выражение:** [Aa]\d{2,}[Aa] **Результат:** a123a a12a

Символ \b

Описание метасимвола

Играет большую роль при разборе выражений. Его функция заключается в следующем. Обычно программы, которые работают с регулярными выражениями (в том числе и grep), ищут сходные выражения в тексте, не определяя, является ли выражение словом и может ли быть расположено конечное выражение в начале или конце слова. Однако часто требуется найти именно конкретное слово, что позволяет сделать данный метасимвол. Он ставится на том месте, где слово должно начинаться или заканчиваться. Рассмотрим пример, в котором попытаемся в данном наборе слов найти начинающиеся на букву s и заканчивающиеся на букву r (для сравнения здесь будут приведены примеры регулярного выражения с использованием метасимвола /b и без него):

Исходный набор строк:
starfish
starless
stellar
ascender
sacrifice
scalar
Регулярное выражение:
[Ss]\w*[Rr]
Результат:
starfish
starless
stellar
a**scender**
sacrifice
scalar

Были выбраны слова, которые не соответствуют условиям. Изменим регулярное выражение, добавив метасимвол \b:

Регулярное выражение:
\b[Ss]\w*[Rr]\b
Результат:
stellar
scalar

1.5.4. Группировка выражений

Особым приемом при составлении регулярных выражений является группировка нескольких его составляющих в одну единицу. Рассмотрим пример, в котором требуется выделить из текста обычный телефонный номер в его записи без кода города, когда весь номер разбивается на три части, между которыми ставится дефис. Для этого подойдет такое регулярное выражение:

\d{l,3}-\d{l,3}-\d{l,3}

Это регулярное выражение можно сократить. В нем есть два идентичных блока текста, стоящих подряд. В этом случае всегда можно сделать выражение короче, заключив повторяющиеся фрагменты в круглые скобки и указав после них количество повторений. Это можно сделать любым способом. Вот один из вариантов:

(\d{1,3}-){2}\d{l,3}

Теперь все выражение, заключенное в скобки, считается единым целым.

Внутри скобок также можно использовать прием, который позволяет выбирать между несколькими выражениями. Вот простой пример. Дано несколько чисел. Необходимо с помощью регулярного выражения выделить из них те, в которых есть цифра 7, после которой находится одна из цифр — 1, 3 или 5. Задачу легко решить с помощью конструкции с квадратными скобками, однако сделаем по-другому. Для выбора между несколькими выражениями требуется заключить их в круглые скобки и поставить между каждыми двумя выражениями символ вертикальной черты. Посмотрим, как таким способом решить поставленную задачу.

Исходный набор строк:
123
178
176
755
713
873
Регулярное выражение:
(7(1|3|5))
Результат:
755
713
873

Данную задачу было бы правильнее решить с помощью конструкции с квадратными скобками, при использовании которой регулярное выражение получилось бы короче. Следует отметить, что в скобках можно также выполнять операцию группировки выражений неограниченное по глубине количество раз, то есть выражение типа (a | (b | (c | d))) будет верным.

1.5.5. Использование зарезервированных символов

Выше упоминалось, что некоторые символы имеют для программы, работающей с регулярными выражениями, особый смысл. Это, например, косая черта, точка, круглая, фигурная и квадратная скобки, звездочка и т. д. Однако не исключено, что в целевом выражении также могут быть эти символы, и их наличие нужно будет определить в регулярном. В данном случае эти символы нужно указать особым образом («защитить» их). При этом перед нужным символом ставят косую черту, то есть, чтобы указать наличие в конечном тексте символа звездочки, в регулярном выражении на соответствующем месте следует написать *. Рассмотрим пример.

С помощью регулярного выражения требуется найти строки, где между некоторыми буквами или цифрами в круглые или квадратные скобки заключено несколько цифр.

Исходный набор строк:
ab(123)cd
a[12]d
al23]d
a(12d

l[123]d
Регулярное выражение:
\w+(\[|\()\d+(\] |\))\w+
Результат:
ab(123)cd
a[12]d
1[123]d

Это регулярное выражение несколько нелогично, так как позволяет сделать открывающей круглую скобку, а закрывающей — квадратную, и наоборот. Попробуйте придумать свое, более универсальное регулярное выражение, которое исправило бы этот недостаток.

1.5.6. Примеры использования регулярных выражений

Рассмотрим несколько примеров на основе изученного материала. В первом примере попытаемся из текста выделить IP-адреса. Следует напомнить, что правильным IP-адресом являются четыре числа, в каждом из которых содержится не более трех цифр, причем эти числа разделены точками. Регулярное выражение будет таким:

\d{l,3}\.\d{l,3}\.\d{l,3}\.\d{l,3}\

В нем содержится три одинаковых блока, стоящих один за другим. Их можно сгруппировать в один блок следующим образом:

(\d{l,3}\.){3}\d{l,3}

Однако это выражение также нельзя назвать верным: ни одно из чисел, входящих в состав IP-адреса, не может быть больше 255. По этой причине составить регулярные выражения не так просто. Следует разбирать каждое число посимвольно. Рассмотрим возможные варианты и регулярные выражения, соответствующие им (табл. 7.3).

Таблица 7.3. Перебор регулярных выражений для определения IP-адреса

Описание	Регулярное выражение
Один или два символа цифр. В таком случае эти цифры могут быть любыми	\d{l,2}
Три символа цифр, причем первая — нуль или единица. В таком случае две другие цифры могут быть любыми	(0|l)\d{2)
Три символа цифр, причем первая двойка, а вторая меньше пяти. В таком случае третья цифра может быть любой	2[0-4]\d
Три символа цифр, причем вторая — пятерка. Третья цифра должна быть меньше или равна пяти	25[0-5]

Осталось сгруппировать выражения, приведенные в таблице, в одно, которое позволило бы выделить из текста число, меньшее либо равное 255:

((\d{l,2})I{(0|l)\d{2}) | (2[0-4]\d) | (25[0-5]))

В итоге получаем регулярное выражение для поиска IP-адреса:

(((\d{1,2}) | ((0|1) \d{2}) | (2[0-4]\d) | (25[0-5]))\.) {3}
((\d{l,2}) | ((0|l)\d{2}) | (2[0-4]\d) | (25 [0-5]))

Продемонстрируем работу этого регулярного выражения на примере.
Пример

Исходный набор строк:
127.0.0.1
255.255.255.255
12.34.56
123.256.0.0
1.23.099.255
Регулярное выражение:
(((\d{1,2}) | ((0 | l)\d{2}) | (2[0-4]\d) | (25 [0-5])) \.) {3}\d{l,3}
Результат:
127.0.0.1
255.255.255.255
1.23.099.255

Рассмотрим пример, связанный с выделением из текста электронных почтовых адресов (то есть адресов e-mail). Для начала определим все возможные варианты, которые должны быть отражены в регулярном выражении. Существует мнение, что не так сложно учесть все варианты, как не допустить, чтобы регулярное выражение соответствовало неправильному конечному тексту. Давайте определим все возможные варианты.

Самым обычным считается адрес типа: username@foobarwebsite.com.

Может показаться, что для выделения адреса e-mail будет достаточно такого регулярного выражения:

\w+@\w+\.\w+

Этого достаточно, чтобы распознать вышеуказанный формат адреса, но данное регулярное выражение не универсально. Следующий адрес корректен, однако вышеуказанное регулярное выражение распознать его не сможет:

my.user.name@sub.foobar.website.com.

В качестве тренировки подумайте, каким будет регулярное выражение. Ответ прост: чтобы регулярное выражение в данном случае было универсальным, следует учесть возможное наличие точек в имени пользователя и домена. Решение будет следующим:

(\w+\ .) *\w+@ (\w+\ .)+\w+

Это не совсем правильный вариант. В имени домена первого уровня могут быть только буквы, но не цифры или другие символы. В связи с этим можно изменить регулярное выражение следующим образом:

(\w+\.)*\w+@(\w+\.) + [A-Za-z]+

Продемонстрируем работу регулярного выражения на примере.

Исходный набор строк:
my@email.com
another.my@email.com
not.my@email.address.com
wrong.address.com
another.wrong.address.com
Регулярное выражение:
(\w+\.)*\w+@(\w+\.)+[A-Za-z]+
Результат:
my@email.com
another.my@email.com
not.my@email.address.com

Закончим еще одним классическим примером — распознаванием адреса страницы в Интернете. Определим критерии, по которым будем искать адрес:

- в начале выражения может быть **http://, https://** или **www.,** причем последнее выражение, если оно существует, должно быть позже двух первых, а http:// и https:// не могут присутствовать одновременно;
- составными частями выражения могут быть буквы, цифры и знаки подчеркивания, причем эти составные части разделяются косыми чертами;
- в конце выражения может стоять косая черта или имя файла (выделяем только адрес страницы и не учитываем случай, когда передаются какие-либо параметры).

На основе этого попытаемся составить регулярное выражение. Разобьем его на составляющие и посмотрим, что описывает каждая его часть (табл. 7.4).

Таблица 7.4. Части, составляющие адрес страницы

Выражение	Описание
(https?://)?	Здесь предположим, что в конечном тексте может быть http:// либо https://
(www\.)?	Предполагаем, что в конечном тексте может быть www
(\w+\.) +	С помощью этой фразы выделяются имена доменов второго и последующих уровней
[A-Za-z]+	Здесь выделяем имя домена первого уровня. Это не совсем верное решение, так как на данный момент существуют только домены первого уровня, длина которых не превышает четырех символов, и при пользовании этой фразой есть шанс вместе с адресами сайтов выделить «мусор»
(/+\w+)*	Предполагаем, что может быть также указан путь к определенному каталогу
(\.\w+)?	Здесь предполагаем, что в адресе может стоять имя файла. В этой фразе учитываем только возможность наличия расширения, так как само имя файла будет учтено предыдущей

В результате получим регулярное выражение:

(https?://) ?(www\.)?(\w+\.) + [A-Za-z] + (/ + \w+)* (\.\w+) ?

Теперь, применив выражение к некоторому тексту, вы сможете выделить из него адреса страниц.

1.6. Примеры задач

Этот раздел посвящен нескольким примерам использования стандартных утилит для решения разных типичных (и не очень) задач. Примеры не следует воспринимать как исчерпывающий список возможностей, они приведены просто для демонстрации того, как можно организовать обработку данных при помощи *конвейера*. Чтобы освоить их, нужно читать руководства и экспериментировать.

1.6.1. Подсчет

Часто бывает необходимо подсчитать количество определенных элементов текстового файла. Для подсчета строк, слов и символов служит стандартная утилита - wc (от англ. "word count" - "подсчет слов"). Используя текстовый вывод утилит, можно посчитать свои файлы:

Пример 7.3. Подсчет файлов при помощи find и wc

```
[student@localhost root]$ find . | wc -l
42
```

Для подсчета файлов в примере использована команда find - инструмент поиска нужных файлов в системе. Команда find вызвана с одним параметром - каталогом, с которого надо начинать поиск. find выводит список найденных файлов по одному на строку, а поскольку критерии поиска в данном случае не уточнялись, то find просто вывела список всех файлов во всех подкаталогах текущего каталога. Этот

список передан утилите wc для посчета количества полученных строк "-l". В ответ wc выдала искомое число - "42".

Задав find критерии поиска, можно посчитать и что-нибудь менее тривиальное, например, файлы, которые создавались или были изменены в определенный промежуток времени, файлы с определенным режимом доступа, с определенным именем и т. п. Узнать обо всех возможностях поиска при помощи find и подсчета при помощи wc можно из руководств по этим программам.

1.6.2. Отбрасывание ненужного

Иногда пользователя интересует только часть из тех данных, которые собирается выводить программа. Утилита head нужна, чтобы вывести только первые несколько строк файла. Не менее полезна утилита tail (англ. "хвост"), выводящая только последние строки файла. Если же пользователя интересует только определенная часть каждой строки файла - поможет утилита cut.

Допустим, пользователю потребовалось получить список всех файлов и подкаталогов в "/etc", которые принадлежат группе "adm". При этом ему нужно, чтобы найденные файлы в списке были представлены не полным путем, а только именем файла (это требуется для последующей автоматической обработки полученного списка):

Пример 7.4. Извлечение отдельного поля

```
[student@localhost root]$ find /etc -maxdepth 1 -group adm 2> /dev/null \
> | cut -d / -f 3
syslog.conf
anacrontab
```

Если команда получается такой длинной, что ее неудобно набирать в одну строку, можно разбить ее на несколько строк, не передавая системе: для этого в том месте, где нужно продолжить набор со следующей строки, достаточно поставить *символ* "\" и нажать Enter. При этом в начале строки bash выведет *символ* ">", означающий, что команда еще не передана системе и набор продолжается. Вид этого приглашения определяется значением переменной *окружения* "PS2". Для системы безразлично, в сколько строк набрана команда, возможность набора в несколько строк нужна только для удобства пользователя.

Пользователь получил нужный результат, задав параметры find - каталог, где нужно искать и параметр поиска - наибольшую допустимую глубину вложенности и группу, которой должны принадлежать найденные файлы. Ненужные *диагностические сообщения* о запрещенном доступе он отправил в /dev/null, а потом указал утилите cut, что в полученных со *стандартного ввода* строках нужно считать *разделителем символ* "/" и вывести только третье *поле*. Таким образом, от строк вида "/etc/filename" осталось только "filename" Как уже указывалось, первым *полем* считается текст от начала строки до первого *разделителя*; в приведенном примере первое *поле* - пусто, "etc" - содержимое второго *поля*, и т. д.

1.6.3. Выбор нужного. Поиск

Зачастую пользователю нужно найти только упоминания чего-то конкретного среди данных, выводимых утилитой. Обычно эта задача сводится к поиску строк, в которых встречается определенное слово или комбинация *символов*. Для этого подходит стандартная утилита grep, которая может искать строку в файлах, а может работать как *фильтр*: получив строки со *стандартного ввода*, она выведет на

стандартный вывод только те строки, где встретилось искомое сочетание *символов*. В следующем примере анализируются процессы bash, которые выполняются в системе:

Пример 7.5. Поиск строки в выводе утилиты

```
[student@ localhost root]$ ps aux | grep bash
student  3459 0.0 3.0 2524 1636 tty2  S  14:30  0:00 -bash
student  3734 0.0 1.1 1644  612 tty2  S  14:50  0:00 grep bash
```

Первый аргумент команды grep - та строка, которую нужно искать в *стандартном вводе*, в данном случае это "bash", а поскольку ps выводит сведения по строке на каждый процесс, на экран будут выведены только процессы, в имени которых есть "bash". Однако неожиданно в списке выполняющихся процессов получены две строки, в которых встретилось слово "bash", т. е. два процесса: один - искомый - командный интерпретатор bash, а другой - процесс поиска строки "grep bash", запущенный **после** ps. Это произошло потому, что после разбора командной строки bash запустил **оба** дочерних процесса, чтобы организовать *конвейер*, и на момент выполнения команды ps процесс grep bash уже был запущен и тоже попал в *вывод* ps. Чтобы в этом примере получить правильный результат, необходимо добавить в *конвейер* еще одно звено: | grep -v grep, эта команда **исключит** из конечного *вывода* все строки, в которых встречается "grep".

1.6.4. Поиск по регулярному выражению

Очень часто точно не известно, какую именно комбинацию *символов* нужно будет найти. Точнее, известно только то, как примерно должно выглядеть искомое слово, что в него должно входить и в каком порядке. Так обычно бывает, если некоторые фрагменты текста имеют строго определенный формат. Например, в руководствах, выводимых программой info, принят такой формат ссылок: "*Note название_узла::". В этом случае нужно искать не конкретное сочетание *символов*, а "Строку "*Note", за которой следует название узла (одно или несколько слов и пробелов), оканчивающееся *символами* "::"". Утилита grep может выполнить такой запрос, если его сформулировать на языке регулярных выражений.

Пример 7.6. Поиск ссылок в файле info

```
[student@ localhost root]$ info grep > grep.info 2> /dev/null
[student@ localhost root]$ grep -on "\*Note[^:]*::" grep.info
324:*Note Grep Programs::
684:*Note Invoking::
[student@ localhost root]$
```

Первый параметр grep, который взят в кавычки - это и есть *регулярное выражение* для поиска ссылок в формате info, второй параметр - имя файла, в котором нужно искать. Ключ "-o" заставляет grep выводить строку не целиком, а только ту часть, которая совпала с *регулярным выражением* (шаблоном поиска), а "-n" - выводить номер строки, в которой встретилось данное совпадение.

В регулярном выражении большинство *символов* обозначают сами себя, как если бы мы искали обыкновенную текстовую строку, например, "Note" и "::" в регулярном выражении соответствуют строкам "Note" и "::" в тексте. Однако некоторые *символы* обладают специальным значением, самый главный из таких *символов* - звездочка ("*"), поставленная после элемента регулярного выражения, обозначает, что могут быть найдены тексты, где этот элемент повторен любое количество раз, в том числе и ни одного, т. е. просто отсутствует. В нашем примере звездочка встретилась дважды: в первый раз

нужно было включить в регулярное выражение именно *символ* "звездочка", для этого потребовалось лишить его специального значения, поставив перед ним "\" (см. раздел 1.5).

Вторая звездочка обозначает, что стоящий перед ней элемент может быть повторен любое количество раз от нуля до бесконечности. В нашем случае звездочка относится к выражению в квадратных скобках - "[^:]", что означает "любой *символ*, кроме ":"". Целиком регулярное выражение можно прочесть так: "Строка "*Note", за которой следует ноль или больше любых *символов*, кроме ":", за которыми следует строка "::"". Особенность работы "*" состоит в том, что она пытается выбрать совпадение максимальной длины. Именно поэтому элемент, к которому относилась "*", был задан как "не ":"". Выражение "ноль или более **любых символов**" (оно записывается как ".*") в случае, когда, например, в одной строке встречается **две** ссылки, вбирает подстроку от конца **первого** "*Note" до начала **последнего** "::" (*символы* ":", поместившиеся внутри этой подстроки, распознаются как "любые").

Регулярные выражения позволяют резко повысить эффективность работы, хорошо интегрированы в рабочую среду в системе Linux.

1.5.5. Замены

Удобство работы с потоком не в последнюю очередь состоит в том, что можно не только выборочно передавать результаты работы программ, но и автоматически заменять один текст другим прямо в потоке.

Для замены одних *символов* другими предназначена утилита tr (сокращение от англ. "translate" - "преобразовывать, переводить"), работающая как *фильтр*. В примере 7.7 утилита применена прямо по назначению – с ее помощью выполнена транслитерация - замена латинских *символов* близкими по звучанию русскими:

Пример 7.7. Замена символов (транслитерация)

```
 [student@ localhost root]$ cat cat.info | tr
abcdefghijklmnopqrstuvwxyz абцдефгхийклмнопкрстуввсиз \
> | tr ABCDEFGHIJKLMNOPRSTUVWXYZ АБЦДЕФГХИЙКЛМНОПКРСТУВВСИЗ | head -4
Филе: цореутилс.инфо, Ноде: цат инвоцатион, Нест: тац инвоцатион,
Тп: Оутпут оф ентире филес
'цат': Цонцатенате анд врите филес
……
[student@ localhost root]$
```

Два параметра для утилиты tr: соответствия латинских букв кириллическим. Первый *символ* из первого параметра tr заменяет первым *символом* второго, второй - вторым и т. д. Пользователь обработал поток *фильтром* tr дважды: сначала чтобы заменить строчные буквы, а затем - прописные. Он мог бы сделать это и за один проход (просто добавив к параметрам прописные после строчных), но не захотел выписывать столь длинные строки. Полученному на выходе тексту вряд ли можно найти практическое применение, однако транслитерацию можно употребить и с пользой. Если не указать tr второго параметра, то все *символы*, перечисленные в первом, будут заменены на "ничто", т. е. попросту удалены из потока. При помощи tr можно также удалить дублирующиеся *символы* (например, лишние пробелы или переводы строки), заменить пробелы переводами строк и т. п.

1.6. **Потоковый редактор sed**

Помимо простой замены отдельных *символов*, возможна замена последовательностей (слов). Специально для этого предназначен потоковый редактор sed (сокращение от англ. "stream editor"). Он работает как фильтр и выполняет редактирование поступающих строк: замену одних последовательностей *символов* другими, причем можно заменять и регулярные выражения.

Например, с помощью sed можно сделать более понятным для непривычного читателя список файлов, выводимый ls:

Пример 7.8. Замена по регулярному выражению

```
[student@ localhost root]$ ls -l | sed s/^-[-rwx]*/Файл:/ | sed s/^d[-rwx]*/Каталог:/
итого 124
Файл: 1 student 2693 Ноя 15 16:09 cat.info
Файл: 1 student   69 Ноя 15 16:08 cat.stderr
Каталог: 2 student 4096 Ноя 15 12:56 Documents
Каталог: 3 student 4096 Ноя 15 13:08 examples
Файл: 1 student 83459 Ноя 15 16:11 grep.info
Файл: 1 student    26 Ноя 15 13:08 loop
Файл: 1 student   23 Ноя 15 13:08 script
Файл: 1 student    33 Ноя 15 16:07 textfile
Каталог: 2 student 4096 Ноя 15 12:56 tmp
Файл: 1 student    32 Ноя 15 13:08 to.sort
[student @ localhost root]$
```

У sed очень широкие возможности, но довольно непривычный синтаксис, например, замена выполняется командой "s/что_заменять/на_что_заменять/". Чтобы в нем разобраться, нужно обязательно прочесть руководство sed(1) и знать регулярные выражения.

1.7. Упорядочивание

Для того чтобы разобраться в данных, нередко требуется их упорядочить: по алфавиту, по номеру, по количеству употреблений. Основной инструмент для упорядочивания - утилита sort. Рассмотрим ее использование в сочетании с несколькими другими утилитами:

Пример 7.9. Получение упорядоченного по частотности списка словоупотреблений

```
 [student@ localhost root]$ cat grep.info | tr "[:upper:]" "[:lower:]" | tr "[:space:][:punct:]" "\n" \
> | sort | uniq -c | sort -nr | head -8
 15233
  720 the
  342 of
```

251 to
244 a
213 and
180 or
180 is
[student@ localhost root]$

В примере 7.9 выполняется подсчет, сколько раз какие слова были употреблены в файле "grep.info" и вывод самых часто употребляемых с указанием количества употреблений в файле. Для этого потребовалось сначала заменить все большие буквы маленькими, чтобы не было разных способов написания одного слова, затем заменить все пробелы и знаки препинания концом строки (*символ* "\n"), чтобы в каждой строке было ровно по одному слову (пользователь всюду взял параметры tr в кавычки, чтобы bash не понял их неправильно). Потом список был отсортирован, все повторяющиеся слова заменены одним словом с указанием количества повторений ("uniq -c"), затем строки снова отсортированы по убыванию чисел в начале строки ("sort -nr") и выведены первые 8 строк ("head -8").

2. МЕТОДИКА ВЫПОЛНЕНИЯ

1. Используя утилиты hexdump и strings, вывести на экран содержимое одного из перечисленных ниже файлов из каталога /bin. Позиция файла для распечатки определяется номером бригады. Имена файлов для выполнения задания 1:

tar, sort, sed, ping, vi, unlink, uname, touch, sleep, sty.

2. Подсчитать общее количество файлов (каталогов) в одном из перечисленных ниже каталогов. Каталог для подсчета количества определяется номером бригады. Имена каталогов для выполнения задания 2:

/bin, /etc, /lib, /proc, /usr, /var, /dev, /sbin, /sys, /root

3. Найти общее количество процессов, выполняющихся в системе в данный момент.

4. Вывести список выполняющихся процессов, в именах которых присутствует слово manager и отсутствует слово grep

5. Создать текстовый файл, содержащий набор строк вида:

123
178
176
755
713
873

С помощью утилиты grep найти строки, в которых есть цифра 7, после которой находится одна из цифр — 1, 3 или 5.

6. Создать текстовый файл, содержащий набор строк вида:

starfish
starless
samscripter
stellar
microsrar
ascender

sacrifice

scalar

С помощью утилиты grep найти строки, начинающиеся на букву s и заканчивающиеся на букву r

7. Создать текстовый файл, содержащий простейшие адреса электронной почты вида username@website.com.

С помощью утилиты grep найти строки, содержащие правильные простейшие адреса. Проверить возможность использования более сложного регулярного выражения для распознавания адресов, содержащих другие допустимые символы.

8. На произвольном примере продемонстрировать работу утилиты tr

Создать текстовый файл, содержащий допустимые и недопустимые IP-адреса, например 127.0.0.1

255.255.255.255

12.34.56

123.256.0.0

1.23.099.255

0.79.378.111

С помощью утилиты grep и руководства man найти строки, содержащие допустимые четырехбайтовые IP адреса.

9. Создать текстовый файл, содержащий корректные и некорректные номера телефонов ведомственной АТС объемом 399 номеров, номера с 000 до 399 – корректные, 0, 400, 900 –некорректные.

С помощью утилиты grep и руководства man найти строки, содержащие допустимые номера телефонов.

3. ОТЧЕТ О РАБОТЕ

Содержание отчета:

1. Командные строки, использованные при выполнении заданий 1 - 9 и результаты выполнения заданий (в текстовом виде или в виде снимков экрана).
2. Письменный ответ на контрольный вопрос

4. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Вывод на экран содержимого нетекстового файла с помощью утилит hexdump и strings.
2. Стандартный ввод, вывод, стандартный вывод ошибок.
3. Конвейер и канал.
4. Фильтры.
5. Структурные единицы текста. Подсчет количества единиц текста.
6. Элементарные регулярные выражения.
7. Знакозаменяющие метасимволы.
8. Метасимволы количества повторений в регулярных выражениях.
9. Группировка выражений в регулярных выражениях.
10. Использование зарезервированных символов в регулярных выражениях.
11. Подсчет количества элементов текстового файла.
12. Назначение утилит head, tail, cut.
13. Назначение и использование утилиты grep.
14. Выполнение транслитерации.
15. Назначение потокового редактора sed.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению практических работ по дисциплине
«Операционные системы»
для студентов специальности 10.05.01 «Компьютерная безопасность»
(специализация «Информационно-аналитическая и техническая экспертиза компьютерных систем»)

Ставрополь
2025

Содержание

Практическое занятие №1. Интерфейс командной строки Windows

Практическое занятие №2. Исследование интерфейса командной строки Windows

Практическое занятие № 3. Исследование программы Debug.

Практическое занятие №4 Мультипрограммные вычислительные процессы

Практическое занятие № 5. Детальное исследование мультипрограммного вычислительного процесса

Практическое занятие № 6. Создание журналов трассировки и оповещений

Практическое занятие № 7.Исследование проблем мультипрограммных вычислительных процессов.

Практическое занятие № 8 Создание потоков в приложении

Практическое занятие №1. Интерфейс командной строки Windows

Цель работы: Изучить интерфейс командной строки Windows

Рассматриваемые вопросы:
1 Состав команд управления
2 Команды справочной системы
3 Команды файловой системы
4 Команды управления работой ОС

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.

Объяснением этого факта, по-видимому, служит следующее. Unix – системы начинались и развивались еще от мощных и дорогих вычислительных машин (mainframes) и ориентировались на работу профессионально подготовленных пользователей (операторов, программистов, системных инженеров). Здесь интерфейс командной строки всегда был и остается традиционно богатым и мощным.

Напротив, Windows – ориентированные системы ведут свое развитие от простых персональных компьютеров. Корпорация Microsoft при разработке операционных систем для них ориентировалась, в первую очередь, на пользователей-непрофессионалов и закладывала принцип “нулевого администрирования”. Согласно ему, в различных версиях ОС Windows предусматривается лишь минимальное участие пользователей в управлении и распределении ресурсов систем. Выполнение этих функций стало прерогативой программ самой операционной системы. Пользователи же довольствуются в основном сервисом графического интерфейса (Graphical Unit Interface, GUI), т.е. пользуются рубриками определенных меню, а значит, ограничивают себя набором стандартных ситуаций управления. Интерфейс командной строки (Command Prompt Interface) в ОС Windows присутствует и играет вспомогательную роль. В свое время он формировался как некое подмножество Unix’а и особого развития не получил. Однако следует сказать, что интерфейс командной строки во многих нештатных ситуациях остается единственным средством определения рассогласований и “тонкой настройки” аппаратно-программных средств. Поэтому в новых версиях операционных систем Windows с учетом роста сложности аппаратной и программной частей компьютерных систем добавлен ряд команд, позволяющих проводить подобные настройки. Часть команд, заимствованных из MS DOS, получили дополнительные возможности. Например, такие команды как dir, copy, xcopy, rename и др., в новых редакциях Windows могут работать с длинными именами файлов.

Рассмотрим особенности построения и применения режима командной строки под управлением операционной системы Windows.

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

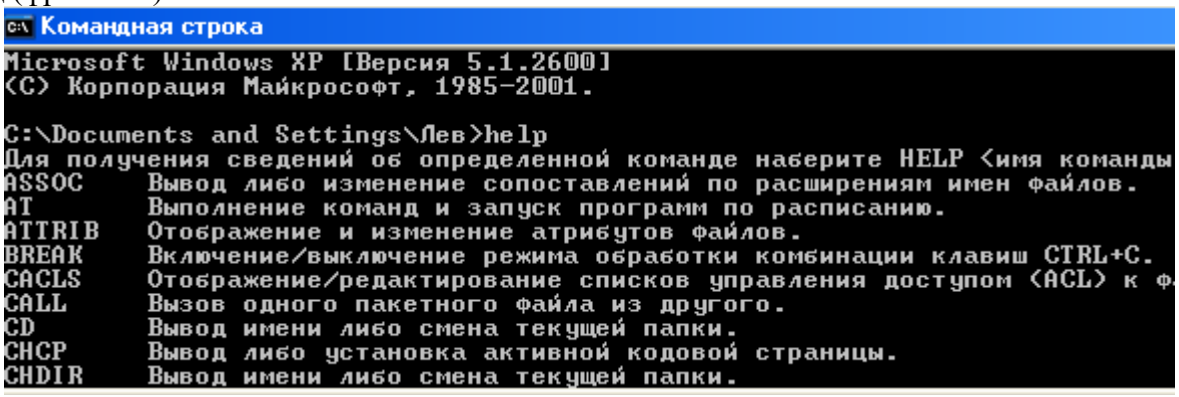
Пуск → Выполнить, затем набрать”cmd” (команда Windows – сокращение от слова commander).

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

Здесь указываются версия работающей операционной системы и строка приглашения с активным диском и рабочим каталогом (папкой).

Выключение режима обеспечивается набором в строке приглашения команды “exit” и ее выполнением при нажатии клавиши Enter.

В состав внутренних команд, составляющих “подножие Windows”, входит примерно семь десятков команд (в системах Unix – более трехсот, уже от этого идут более богатые методы управления ресурсами). Весь перечень этих команд можно просмотреть, если в командной строке выполнить команду “**help**”. Ниже представлено начало списка этих команд (фрагмент).



Поскольку весь перечень команд перекрывает размер экрана дисплея, то для ознакомления с каждым элементом перечня следует использовать полосу прокрутки окна или вызов на экран частей перечня постранично. Для этого следует набрать более сложную команду, состоящую из конвейера двух команд “help|more”. Правила построения конвейеров будут изложены ниже. Так как список команд включает почти семь десятков команд, то обычно их рассматривают по группам:

Команды справочной системы;
Команды файловой системы;
Команды управления работой ОС;
Команды пакетных (командных) файлов.
Ниже они приведены по группам.

2 вопрос

Команды справочной системы

HELP	Выводит справочную информацию о системе команд Windows
HELP имя_команды	Выводит справочную информацию для набранной команды
Имя_команды /?	Выводит справочную информацию для набранной команды

3 вопрос

Команды файловой системы

ATTRIB	Отображение и изменение атрибутов файлов.

CD	Вывод имени либо смена текущей папки.
CHDIR	Вывод имени либо смена текущей папки.
CHKDSK	Проверка диска и вывод статистики.
CLS	Очистка экрана.
COMP	Сравнение содержимого двух файлов или двух наборов файлов.
COPY	Копирование одного или нескольких файлов в другое место.
DEL	Удаление одного или нескольких файлов.
DIR	Вывод списка файлов и подпапок из указанной папки.
DISKCOMP	Сравнение содержимого двух гибких дисков
DISKCOPY	Копирование содержимого одного гибкого диска на другой.
ERASE	Удаление одного или нескольких файлов.
FC	Сравнение двух файлов или двух наборов файлов и вывод различий между ними.
FIND	Поиск текстовой строки в одном или нескольких файлах.
FINDSTR	Поиск строк в файлах.
FORMAT	Форматирование диска для работы с Windows 2000/XP.
LABEL	Создание, изменение и удаление меток тома для дисков.
MD	Создание папки.
MKDIR	Создание папки.
MOVE	Перемещение одного или нескольких файлов из одной папки в другую.
PUSHD	Сохранение значения текущей активной папки и переход к другой папке.
POPD	Восстановление предыдущего значения текущей активной папки, сохраненного с помощью команды PUSHD.
PRINT	Вывод на печать содержимого текстовых файлов.
RD	Удаление папки.
REN	Переименование файлов и папок.
RENAME	Переименование файлов и папок.
REPLACE	Замещение файлов.
RMDIR	Удаление папки.
SORT	Сортировка ввода.
TREE	Графическое отображение структуры папок заданного диска или заданной папки.
TYPE	Вывод на экран содержимого текстовых файлов.
VERIFY	Установка режима проверки правильности записи файлов на диск
VOL	Вывод метки и серийного номера тома для диска.
XCOPY	Копирование файлов и дерева папок.

4 вопрос

Команды управления работой ОС

ASSOC	Вывод либо изменение сопоставлений по расширениям имен файлов.
AT	Выполнение команд и запуск программ по расписанию.
BREAK	Включение/выключение режима обработки комбинации клавиш CTRL+C.
CACLS	Отображение/редактирование списков управления доступом (ACL) к

	файлам.
CHCP	Вывод либо установка активной кодовой страницы.
CHKNTFS	Отображение или изменение выполнения проверки диска во время загрузки.
CMD	Запуск еще одного интерпретатора командных строк Windows 2000/XP.
COLOR	Установка цвета текста и фона, используемых по умолчанию.
COMPACT	Отображение/изменение сжатия файлов в разделах NTFS.
CONVERT	Преобразование дисковых томов FAT в NTFS. Нельзя выполнить преобразование текущего активного диска.
DATE	Вывод либо установка текущей даты.
DOSKEY	Редактирование и повторный вызов командных строк; создание макросов.
FTYPE	Вывод либо изменение типов файлов, используемых при сопоставлении по расширениям имен файлов.
GRAFTABL	Позволяет Windows 2000/XP отображать расширенный набор символов в графическом режиме.
MODE	Конфигурирование системных устройств.
MORE	Последовательный вывод данных по частям размером в один экран.
PATH	Вывод либо установка пути поиска исполняемых файлов.
PROMPT	Изменение приглашения в командной строке Windows 2000/XP.
RECOVER	Восстановление читаемой информации с плохого или поврежденного диска.
SET	Вывод, установка и удаление переменных среды Windows 2000/XP.
START	Запуск программы или команды в отдельном окне.
SUBST	Сопоставляет заданному пути имя диска.
VER	Вывод сведений о версии Windows 2000/XP.

Практическое занятие №2. Исследование интерфейса командной строки Windows (часть2)

Цель работы: Исследовать интерфейс командной строки Windows

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

Одной из интересных команд управления файловой системой является команда **start.** Она служит для запуска указанной в качестве параметра программы или команды в отдельном окне. Правда, запуск отдельных программ можно осуществлять и минуя эту команду. Например, следующие варианты команд

C:>start Document.txt<Enter> или

C:>start notepad Document.txt <Enter> или

C:> notepad Document.txt <Enter>

приводят к одному и тому же эффекту – вызову программы Блокнот с последующей загрузкой в него файла Document.txt, если такой файл существует. Она интересна тем, что позволяет управлять приоритетами программ. Это очень важное качество команды, так как очередь готовых программ упорядочивается по убыванию их приоритетов, а значит, команда позволяет управлять последовательностью запуска программ. Все современные компьютеры являются многопрограммными. Из очереди готовых программ сначала запускаются программы с высоким приоритетом, затем с более низким. Имеются четыре стандартных приоритета: самый низкий (low) – 4; нормальный (normal) – 8; высокий (high) – 13; реального времени (realtime) – 24. Последний вид приоритета пользователям не рекомендуется использовать, так как это может дестабилизировать систему. Значение приоритета задается в виде параметра сразу после команды start перед именем запускаемой программы, например:

C:> start /low notepad

C:> start /high test.bat.

Команда **dir** – команда вывода на экран дисплея содержимого каталогов (папок). Команда дает пользователю достаточно большой объем информации. Она выдает полный перечень всех папок и файлов в исследуемом каталоге. Содержимое каталога (папки) включает данные об имеющихся файлах и дочерних каталогах, их именах, размерах файлов, времени и датах их создания, а также сводные данные:

общее число всех файлов, размещаемых в каталоге, и их суммарный объем;

общее число папок в каталоге;

общий объем свободного пространства диска, доступный пользователю.

Две первые строки содержимого каждого каталога вместо имен помечены точкой и двумя точками, что обозначает соответственно вход и выход в данном каталоге (папке).

Формат этой команды

dir *диск:путь\имя_файла.расш ключи.*

Наиболее употребительными вариантами применения этой команды служат:

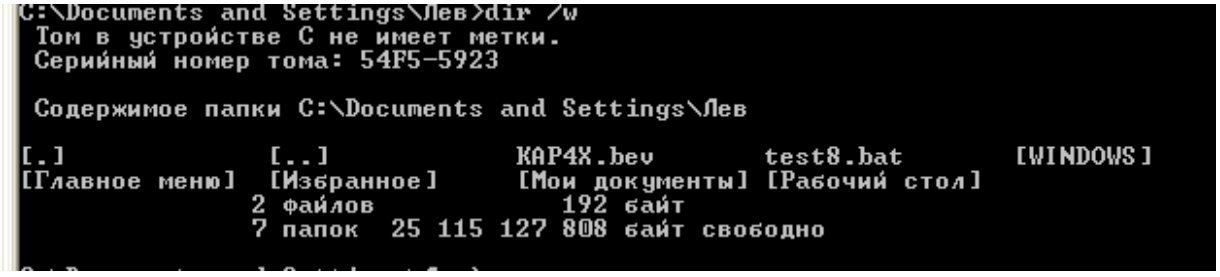
исследоав) ание Исосдпеорлжьизомвоагноите т.е. подразумевается

кукщоимханкдаыталог**d**о**ir**в и

дбиезсковп.аНр**а**мпреитрмоевр,,

Использование команды **dir** с параметром /p. Используется в случаях, когда количество объектов в каталоге превышает количество строк экрана. Осуществляется вывод страницами, для перехода на следующую страницу следует нажать любую клавишу. Практически аналогом этого варианта применения команды является использование конвейера двух команд **dir|more;**

Использование команды **dir** с параметром /w. В этом случае на экран выдается краткая информация о содержимом каталога в виде пяти столбцов с именами файлов и сводные данные.



Задание 1:

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

3. Кроме команды **dir,** формирующей информацию о содержимом каталогов и файлов, существует еще одна команда **tree**, строящая дерево каталогов и выводящая его на экран дисплея. Команда позволяет представить структуру каталогов. Формат команды:

Tree *диск:\маршрут* ./f

Ключ ./f означает, что список файлов каждого каталога выводится на дисплей.

Если в команде tree не указаны параметры *диск:* и *маршрут,* то по умолчанию, считается, что требуется вывести структуру подкаталогов и файлов текущего каталога.

Команда

**tree **

предполагает вывод всей структуры дерева каталогов текущего диска.

Изучение дерева каталогов большого объема можно проводить постранично, используя конвейер команд, например,

Tree d:\ /f |*more*

Результат выполнения команды ***tree*** можно сохранить в любом текстовом файле, указав его имя, например:

Tree d:\ /f tree.lst,

или путем переспециальности операции вывода (используя знаки > и >>).

4. Команда **find** используется для поиска фрагмента текстовой строки (определенного набора символов – шаблона поиска) в одном или нескольких файлах.
Формат команды

find [/v] [/c] [/n] [/i] “строка символов” [[диск:][путь]имя файла[…]].

Если путь к файлу и его имя в команде не заданы, то поиск выполняется в тексте, вводимом с клавиатуры или переданном по конвейеру другой командой. Команда имеет четыре ключа, определяющие режимы поиска:

/c – поиск с определением общего числа строк в файле, содержащих шаблон поиска; /

i – поиск строк, содержащих шаблон поиска, без учета регистра символов (прописных и строчных букв);

/n – поиск строк, содержащих шаблон, с отображением порядковых номеров строк; /v – вывод на экран всех строк файла, не содержащих шаблон поиска.

Задание 2:

Создать текстовый файл с некоторым фрагментом текста;

Набрать и выполнить команду find с различными ключами и шаблонами поиска элементов этого текста;

Оценить и объяснить результаты поиска. ***Задание 3:***

Создать несколько текстовых файлов с наличием в них одних и тех же элементов текста;

Организовать поиск элементов текста в группе файлов, используя шаблоны имен файлов (например, *.txt, s*.lst) и их перечисление.

Команда find в конвейерах с другими командами служит мощным средством получения итоговых характеристик исследования каталогов (папок). Например, если внимательно изучить структуру и содержание информации, выдаваемой по команде dir, то можно получать следующие характеристики:

dir|find “файлов” – количество файлов в каталоге и их суммарный объем; ***dir|find*** “байт” – то же самое, так как внутри строки информация не разделяется; ***dir|find*** “папок” – количество папок в каталоге и объем свободного пространства диска;

dir|find “.txt” – весь перечень строк, характеризующих текстовые файлы. Меняя тип расширения файлов, можно получить аналогичную характеристику по другим типам файлов;

dir|find /c “.doc” – подсчет общего числа файлов с заданным расширением.

Обратим внимание, что информация, получаемая перечисленными конвейерами, не может быть определена с помощью рубрик меню Windows.

Задание 4:

Проверить действие перечисленных конвейеров команд ***dir*** и ***find*** для собственного каталога, меняя режимы поиска и шаблоны;

Оценить и объяснить результаты поиска.

5. Следует иметь в виду, что при формировании файловой структуры желательно иметь дерево каталогов “широким, но не глубоким”. Это позволяет уменьшить трудоемкость записи маршрута при доступе к нужному файлу, а значит, сократить число ошибок в наборе требуемых команд. Но это требование не всегда удается выполнить. В этом случае полезно использовать команду **subst**, заменяющую длинный маршрут доступа к

объектам каталога одной буквой или более простым идентификатором. По существу команда создает виртуальный, логический диск с более простым доступом. Формат команды

Subst *диск1: диск2:\маршрут*

Здесь параметр *диск1:* несет указание буквы (идентификатора) виртуального диска, которой будет обозначаться требуемый каталог. Параметр *диск2:\маршрут* указывает местонахождение (наименование диска и маршрут следования) требуемого каталога.

Например, команда

Subst q: a:\letters\pavlov

Формирует виртуальный диск *q:* вместо маршрута *a:\letters\pavlov.*

Удаление виртуального диска производится командой

Subst диск1: /d

6. Очень похожие действия по уменьшению трудоемкости использования “длинных” путей к файлам можно выполнить с помощью команды set. Эта команда предназначается для вывода, создания и удаления переменных среды интерпретатора команд cmd.exe. Она имеет формат

Set [*переменная*=[*строка*]],

где *переменная* задается именем переменной среды,

строка – это строка символов, присваиваемая указанной переменной.

По этой команде формируется символьная переменная окружения в резервируемой памяти операционной системы, доступной для команд ОС и программам пользователей. Следует помнить, что имя создаваемой переменной не должно совпадать с ключевыми резервируемыми словами. Команда **set** без параметра выводит содержимое всех используемых ОС переменных окружения, а команда

Set *имя*= (без значения)

удаляет соответствующую строку из окружения.

Например, команда **Set** TMP=E:\GARBAGE устанавливает значение переменной окружения TMP, которую прикладные программы фирмы Microsoft (Word, Excel и др.) используют для определения каталога TMP, в котором они будут создавать временные файлы. Файлы, имеющие малый период существования, можно помещать в этот каталог. Это исключает рассеивание этих файлов по дисковым накопителям, их выявление и удаление.

Команда **set** в последних версиях Windows достаточно сложна и имеет много дополнительных возможностей. Так, например, с ключом /a (**set** /a *переменная=выражение*) она, по существу, является совершенно новой командой. Она позволяет обрабатывать арифметические и алгебраические выражения, заданные в правой части равенства. Обработчик формул очень прост и поддерживает следующие операции над десятичными, восьмеричными и шестнадцатеричными переменными:

() – группировка;

! ~ – унарные операторы;

* / % – арифметические операторы;

+ – арифметические операторы;

<<>> – двоичный сдвиг;

& – двоичное И;

^ – двоичное исключающее ИЛИ;

| – двоичное ИЛИ;

= *= /=%= += -= – присвоение;

&= ^= |= <<=>> = – присвоение;

, – разделитель операторов.

Ниже представлен пример задания целочисленной переменной **d** и с помощью нее подсчет некоторых целочисленных констант, новые значения которых выводятся сразу после выполнения набранных команд.

```
C:\Documents and Settings\Лев>set /a d=0
0
C:\Documents and Settings\Лев>set /a d=d+2
2
C:\Documents and Settings\Лев>set /a d=d+5
7
C:\Documents and Settings\Лев>set /a d=d+10
17
```

В следующем параграфе при разработке командных (пакетных) файлов будет показано, как эта команда в данной конструкции будет использоваться для подсчета некоторых характеристик проводимых вычислений.

Задание 5:

1) Создать логический диск для одного из своих каталогов и убедиться, что механизм логических дисков упрощает работу пользователя;

2) С помощью команды set создать текстовую (вые) переменную (ые) окружения и привести пример её (их) использования;

3) Создать целочисленную переменную окружения и убедиться в возможности использования ее при обработке данных.

4) Команда mkdir или md. Предназначается для создания новых каталогов (папок). Формат команды

>mkdir (или md) диск:путь\имя_каталога.

Например, команда

C:\>md a:\practice

обозначает, что, работая в корневом каталоге диска C:, создаем в корневом каталоге дискеты A: 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 уже указанного правила: дерево создаваемых каталогов должно быть “широким, но не глубоким”. В противном случае трудоемкость записи пути к требуемым каталогам и файлам резко возрастает.

6. Команда **chdir** (или **cd**) – команда смены каталога. Команда имеет формат

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 лучше всего применять сокращенную форму, например,

cd..

Для перехода из каталога нижнего уровня в корневой каталог диска лучше всего применять сокращенную форму, например,

cd

7. Команда **copy** (копирования) – достаточно сложная команда. Помимо основного своего назначения – копирования файлов, она используется для их объединения

(конкатенации), создания новых текстовых файлов, передачи параметров на отдельные устройства и т.д.

Для копирования команда имеет следующий формат

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

В команде сначала указывается входная спецификация Идиск: путь\имя_файла.расш (И – источника), затем – выходная Пдиск:путь\имя_файла.расш (П-приемника). В спецификациях источника и приемника допускается применение шаблонов групповых операций. В шаблонах используются символы: * – произвольное сочетание символов, ? – промзвольный одиночный символ.

Примеры:

Команда >copy a:s*.txt c:\proba выполняет копирование с дискеты всех текстовых файлов с именами, начинающимися с буквы s и расширением txt;

Команда >copy a:*.* nul осуществляет 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 *И1 диск:путь\имя_файла.расш + И2 диск: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….

Примеры:

1) copy ch1.doc+ch2.doc+ch3.doc book.doc /v

Данная команда производит объединение трех файлов ch1.doc, ch2.doc и ch3.doc в один файл book.doc в текущем каталоге текущего диска с проверкой записи.

Copy ch*.doc book.doc /v

Данная команда производит объединение всех файлов ch*.doc с произвольными номерами в один файл book.doc в текущем каталоге текущего диска с проверкой записи.

Copy con+trans.bat trans.bat

По данной команде строки, вводимые с клавиатуры, добавляются в начало уже существующего файла trans.bat. Ввод завершается нажатием клавиш <Ctrl>+Z и <Enter>.

Все варианты команды **copy** могут использовать ключи /a, /b, /v. Ключ /a рассматривает обрабатываемые файлы, как текстовые в коде ASCII (Американский код передачи информации). Ключ /b обозначает, что файлы рассматриваются 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.

Интересна следующая модификация команды **copy**

>copy con prn

Команда расшифровывается: копировать информацию, набираемую на клавиатуре (**con** – консоли) на принтер (**prn** – ключевое слово, обозначающее принтер). Она 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. Выполните задание, записанное в конце этого пункта. Оно как раз и иллюстрирует передачу параметров на принтер.

Создание текстовых файлов можно выполнить следующей командой

>copy *диск:путь\имя_файла.расш.*

Например, для создания файла **f1.txt** в текущем каталоге необходимо выполнить команду

>copy con f1.txt

Этой командой ввод текста с клавиатуры (консоли con) осуществляется в новый, создаваемый этой операцией файл f1.txt. После набора каждой строки следует нажимать клавишу <Enter>. Окончание набора файла должно заканчиваться нажатием клавиш <Ctrl>+Z или F6 (признаком окончания 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. Таким образом, команда **copy** использует простейший однострочный редактор.

Задание 6:

Создайте небольшой текстовый файл, например f1.txt.

Передайте его на принтер командой **>copy f1.txt prn.** Обратите внимание, каким шрифтом файл был распечатан.

Измените значение параметра – размер шрифта на принтере следующей последовательностью команд:

>copy con prn

<Ctrl>+O (установка уплотненного шрифта)

<Ctrl>+Z или **F6** **(**завершение файла)

Здесь **<Ctrl>+O** заставляет принтер перейти на уплотненный шрифт, а следующая строка заканчивает файл (установку).

Выполните действия п.2, т.е. снова повторите печать файла.

Сравните обе распечатки файла.

6. Более привычным для пользователей будет использование внешнего редактора, вызываемого командой

>**edit** *диск:путь\имя_файла.расш.*

Этот редактор имеет собственное, достаточно подробное меню. С помощью этого редактора удобно оперативно вносить изменения и создавать новые версии исследуемых примеров.

Задание 7:

Создайте новый текстовый файл, используя команду:

edit ff.txt.

Просмотрите рубрики меню встроенного редактора **edit**.

Попробуйте средствами редактора создать несколько текстовых файлов, выполнить их редактирование и сохранение.

9. Команда **type** – команда просмотра содержимого текстовых файлов. Формат команды

type *диск:путь\имя_файла.расш*

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 8:

Создайте новый рабочий каталог.

Создайте различными методами несколько текстовых файлов.

Запишите в конец одного из файлов текущую дату и время.

Используя команды **dir** и **type**, убедитесь в правильности проведенных операций.

10. Атрибуты файлов. Каждый файл и каталог, находящиеся в компьютере, могут иметь атрибуты – характеристики, используемые операционной системой для их защиты. Атрибутами файла (файлов) могут быть:

- R – “только для чтения”, то есть нельзя модифицировать файл и уничтожить его;
- A – “архивный”, то есть +A обозначает, что снимались копии данного файла, соответственно –A – файл является вновь созданным;
- H – “скрытый”, скрытые файлы не показываются командой dir и некоторыми программными оболочками;
- S – “системный”, этот атрибут показывает, что файл является принадлежностью операционной системы.

Изменение атрибутов файлов осуществляется командой **attrib**. Формат команды

Attrib +|-*атрибут* ***диск:****маршрут**имя файла* ./s

Установка любого атрибута производится знаком “+”, отмена – знаком “-“. Можно задавать изменение сразу нескольких атрибутов в любой последовательности. Ключ /s,стоящий в конце формата, указывает, что процесс изменения атрибутов файла (файлов) распространяется не только на текущий каталог, но и на все каталоги, подчиненные текущему.

Примеры:

1) команда

Attrib +A +H +R primer.txt

присваивает файлу сразу три атрибута:”только для чтения”,”архивный” и ”скрытый” файлу **primer.txt**.

2) команда

Attrib -R a:*.* /s

с использованием шаблона снимает атрибут ”только для чтения” у всех файлов диска A, то есть находящихся на дискете.

11. Команда **del –** команда удаления одного или нескольких файлов. Аналогом этой команды служит еще одна команда **erase.** Формат команд

del (erase) – *диск:путь\имя_файла.расш*

Особенностями выполнения команд служат следующие ситуации:

Если в команде не указан , то подразумевается, что удалению подлежит (подлежат) файл или файлы текущего каталога.

В спецификации файла допускается использование групповых операций. Спецификация *.* обозначает, что удалению подлежат все файлы выбранного каталога.

Если в командах отсутствует спецификация файла, то командный процессор выдает предупреждение “Ошибка в синтаксисе команды”.

Команды **del** не удаляют каталоги (подкаталоги). Они используются только для удаления в них файлов.

12. Команда **rmdir** (или **rd)** – команда удаления каталога. Формат команды

rmdir (rd) – *диск:путь*

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

Задание 9:

Самостоятельно изучите особенности применения и использования команд: **Sort, Mode, More** и др.

Практическое занятие № 3. Исследование программы Debug.

Цель работы: Изучить программу Debug

Рассматриваемые вопросы:

1 Использование программы Debug

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

1 вопрос

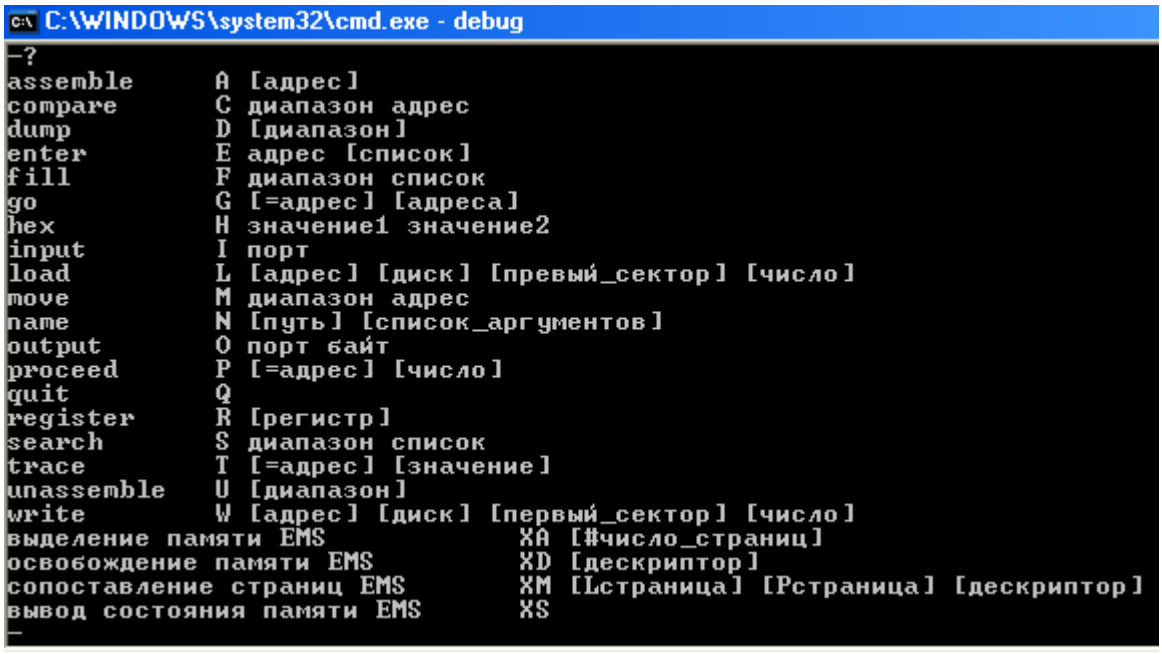
В составе любой операционной системы, а также в любой среде разработки программных комплексов имеются так называемые программы-отладчики.

У операционной системы MS DOS таким отладчиком была программа **debug**, в последних версиях OS Windows такой программой служит **ntsd.exe**. Первая из этих программ используется для однопрограммных однопользовательских режимов работы и 16-разрядных приложений. Вторая программа очень сложна, она предназначена для работы в многопрограммном многопользовательском режиме работы вычислительной системы с 32-разрядными приложениями.

Программу **debug** целесообразно использовать для обучения студентов при начальном знакомстве с операционными системами и принципами их построения и работы. Она удобна для демонстрации процессов, происходящих в микропроцессоре во время работы и принципов управления различными устройствами компьютера.

Вызов программы **debug** осуществляется путем набора в командной строке её имени и нажатием клавиши <Enter>. Компьютер переходит в режим эмуляции DOS. Программа вызывается и размещается в памяти компьютера, начиная с ячейки 100h (шестнадцатеричный адрес). В следующей строке появляется мигающий минус “-“ – приглашение к вводу нужной команды. Выход из программы **debug** осуществляется набором буквы **q (Quit** – завершение работы).

Список всех команд, выполняемых программой debug можно получить, если в строке приглашения набрать: -?, как показано на рисунке.

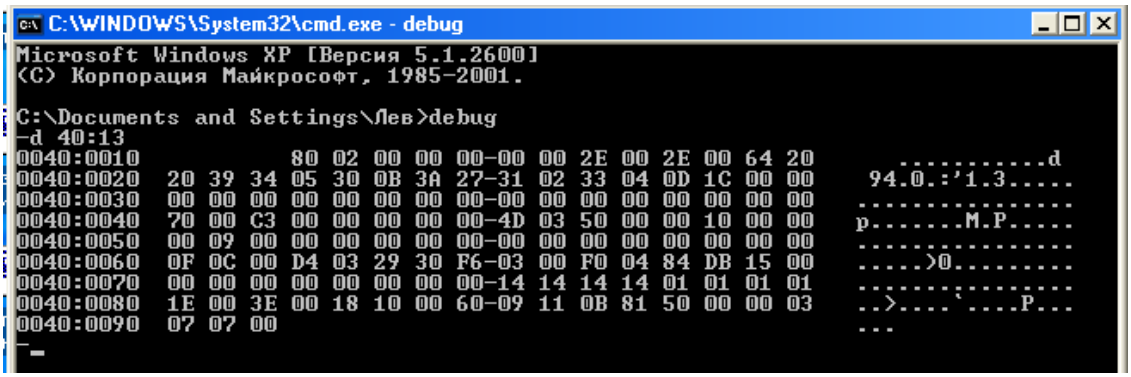


Просмотр памяти

Проверка объема основной памяти DOS

Эта информация после загрузки DOS находится в 413h и 414h байтах памяти (адреса младшего и старшего байтов в шестнадцатеричном представлении). Зададим адрес сегмента: 400 (последний нуль следует “отбросить” и этим самым указываем начало параграфа 40, умноженное на 16) и смещения: 13.

Набираем команду _d (dump – диапазон) 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.

Как и должно получиться – в первом (413) и втором (414) байтах распечатки содержатся цифры 80 и 02, то есть с учетом “вращения байтов” получаем число 0280h Кбайт в шестнадцатеричном представлении. Переводим в десятичное представление

2*16*16+8*16+0=640 Кб.

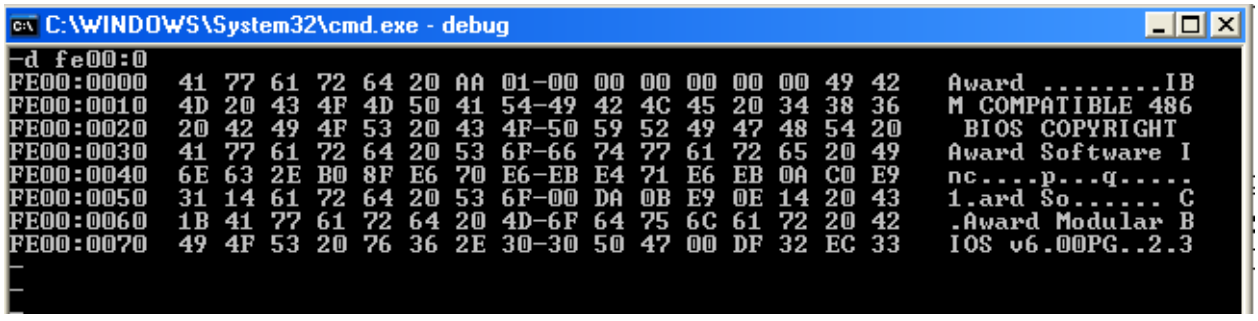
Что и должно быть!

Проверка “серийного номера и даты копирайта”

Эта информация находится в ПЗУ, начиная с адреса FE000h. Здесь находятся: семизначный номер компьютера, дата копирайта, фирма изготовитель. Данная информация является символьной.

Аналогично наберем адрес памяти:_d FE00:0 <Enter>.

На эту команду получаем следующее:

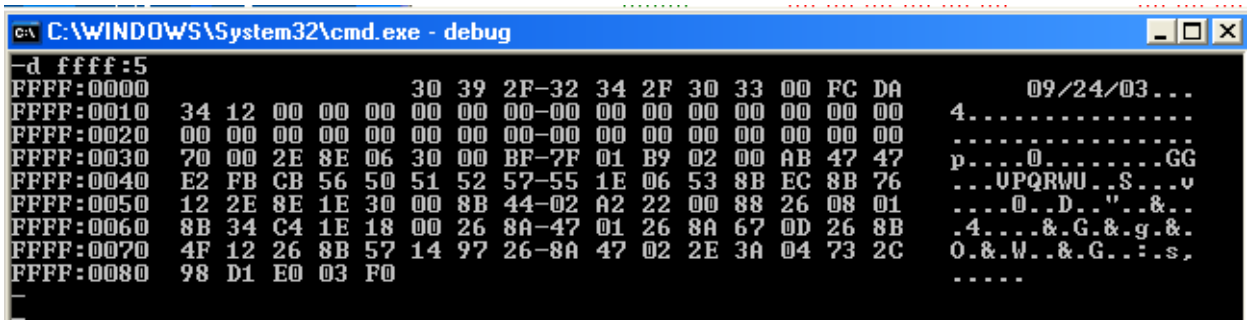


Проверка даты “прошивки” ROM BIOS

Информация находится по адресу: FFFF5h. Формат даты – американский, то есть mm/dd/yy.

Вводим адрес и выполняем команду:_d FFFF:5 <Enter>.

По этой команде должна быть выдана информация о дате “прошивки” ПЗУ (ROM BIOS).



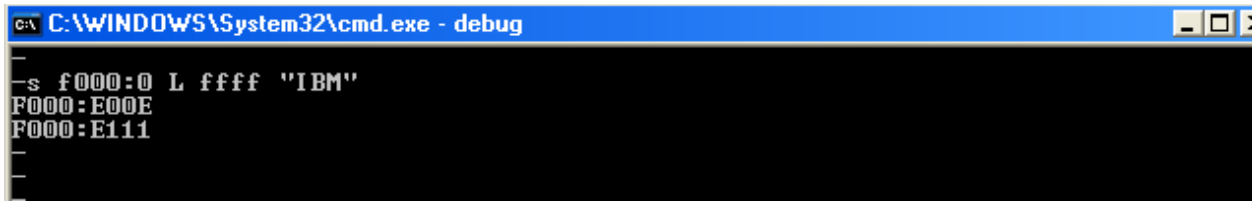
В верхней строке в символьном представлении читаем интересующую нас дату.

Еще одна полезная операция!

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:

_s F000:0 L FFFF ”IBM” <Enter>



Ниже выполненной команды следует перечень адресов, в которых содержится искомое ключевое слово. Проверьте этот факт, набрав эти адреса памяти.

Команду S (Search) можно использовать, например, для поиска вирусов, если известны их “следы” и т.п.

Выводы:

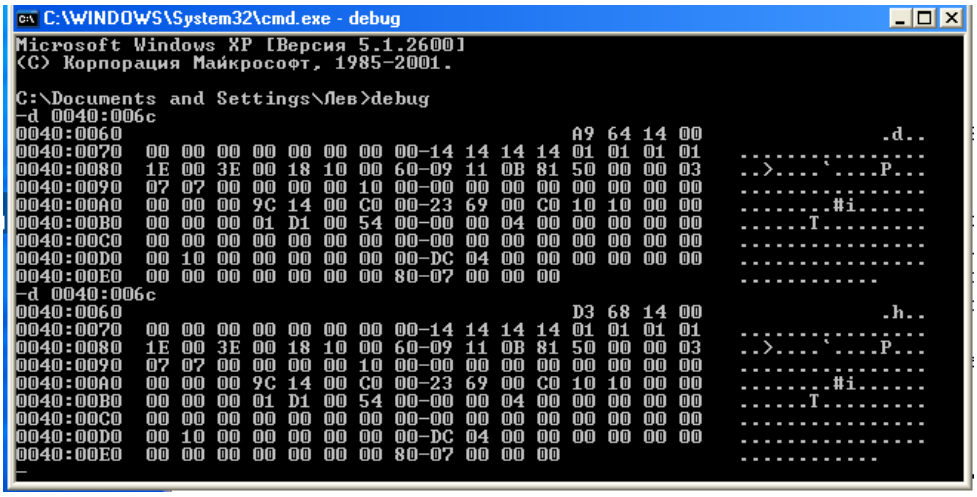
1). Эти три операции объективно позволяют определить “модель и возраст компьютера”.

2). Пользуясь этими командами можно просмотреть содержимое любой ячейки памяти.

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 $f = 14{,}31818$ Мгц, и составляет 1/12 этой частоты.

Определим значение счетчика с помощью программы debug.



Изменение значений байтов счетчика наглядно показывает, что время неотвратимо “бежит вперед”.

Задание. Запишите два показания счетчика примерно 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 вопрос

Задача. Пусть имеется следующая программа:

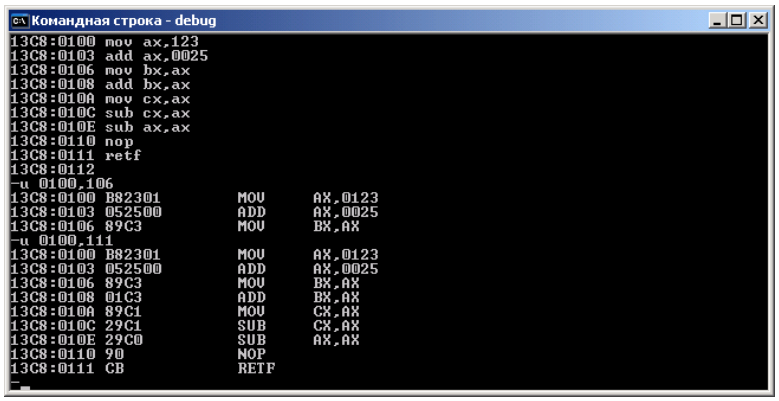
Адреса	Ассемблерный код	Машинный код	Содержание
0100	MOV AX,123	B8 2301	(AX):= 123h
0103	ADD AX,0025	05 2500	(AX):= 123h + 25h =148h
0106	MOV BX,AX	89 C3	(BX):= AX
0108	ADD BX,AX	01 C3	(BX):= BX + AX=290h
010A	MOV CX,AX	89 C1	(CX):= BX
010C	SUB CX,AX	29 C1	(CX):= CX – AX=148h
010E	SUB AX,AX	29 C0	(AX):= 0
010F	NOP	90	
	RETF	CB	

Обратить внимание при выполнении следующих заданий на следующие моменты:

1). Как вычисляются адреса команд?

2). Как числовая информация размещается в памяти?

3). Как должно меняться и меняется содержимое регистров AX, BX, CX?



Ввод программы в память ПК.

Прежде всего 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>

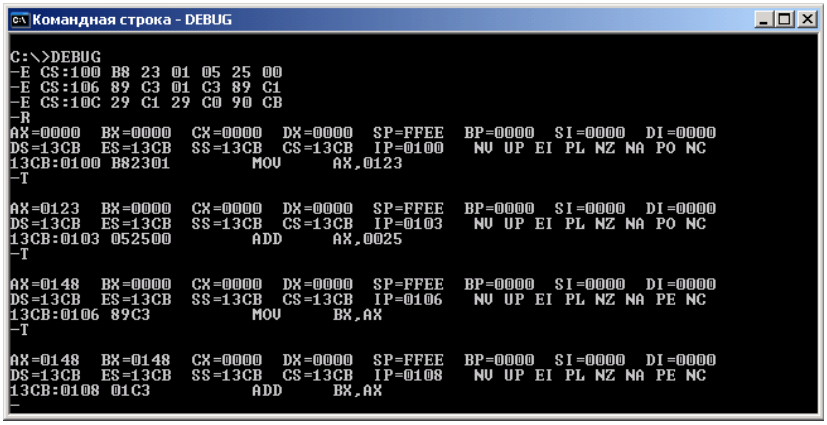
Затем подадим команду _R (Registr) – команду чтения содержимого регистров, включая и регистр флагов. Это позволит просмотреть исходное состояние всех регистров общего назначения перед выполнением программы. Содержимое регистров CS и IP определяет адрес очередной выполняемой команды MOV AX,123, адрес, машинный код и содержание которой приведено после перечня регистров.

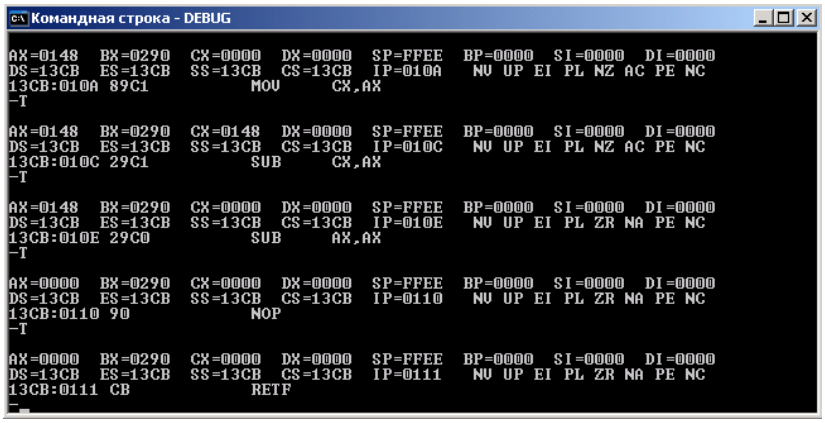
-r

-

Если теперь подать команду T (трассировка, то есть покомандное выполнение программы с остановками), то можно увидеть последовательное изменение содержания регистров в соответствии с командами программы.

Повторять выполнение команды _T и анализировать выполнение каждой очередной команды до тех пор, пока не будет выполнена последняя команда программы.





Обратить внимание:

Как и почему меняется содержимое регистров после выполнения очередной команды программы?

Как и почему меняется содержимое регистра флагов?

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

набираем и выполняем команду _R IP, для того, чтобы просмотреть содержимое регистра IP;

вводим 0100 и нажимаем клавишу <Enter>. Этим самым регистр IP вновь получает значение 0100, и можно снова повторить трассировку.

Пример выполнения программы BIOS

Рассмотрим фрагмент программы BIOS, определяющей объем основной памяти DOS компьютера (так же, как это было в примере 1.1.), но не вручную, а по прошитой программе. Программа работает по прерыванию INT 12h.

В ранних версиях DOS это прерывание обрабатывалось обработчиком прерываний по следующей программе:

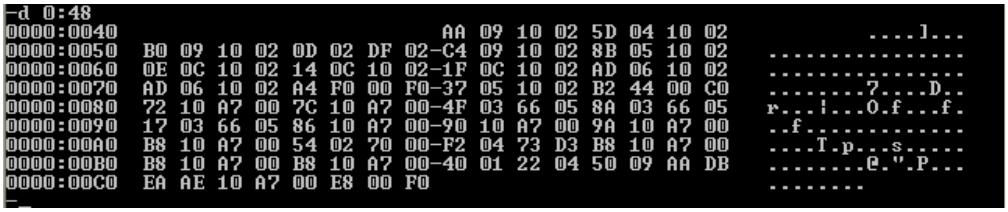
```
STI
PUSH DS
MOV AX,0040
MOV DS, AX
MOV AX, [0013]
POP DS
IRET
```

В этой программе обратим внимание на начало и конец, где командой PUSH DS обеспечивается сохранение старого значения регистра DS, а командой POP DS восстанавливается это значение. Командой IRET возвращается управление прерванной программе.

Рассмотрим, какой последовательностью команд под управлением Windows реализуется теперь обработка этого прерывания. Рассмотрим и сам механизм обращения к обработчику прерывания по его вектору.

Известно, что в IBM PC возможно возникновение 256 видов прерываний с номерами от 0 до 255 (или в шестнадцатеричной системе от 0h до FFh). Они занимают младшие адреса оперативной памяти с 0h до 3FFh (1Кбайт). Каждый вектор имеет размер четыре байта, первые два байта определяют значение IP, другие – значение CS программы – обработчика прерывания. При возникновении прерывания любого типа определяется его номер, а затем по его вектору определяется адрес программы обработки этого вида прерывания. Продемонстрируем этот процесс.

Поскольку каждый вектор занимает четыре байта, то, увеличив его двоичный номер в четыре раза, получим адрес его вектора. Адрес вектора INT 12h будет равным 100102*4=10010002 или 48h. Посмотрим содержимое этого вектора, послав команду –d 0:48.



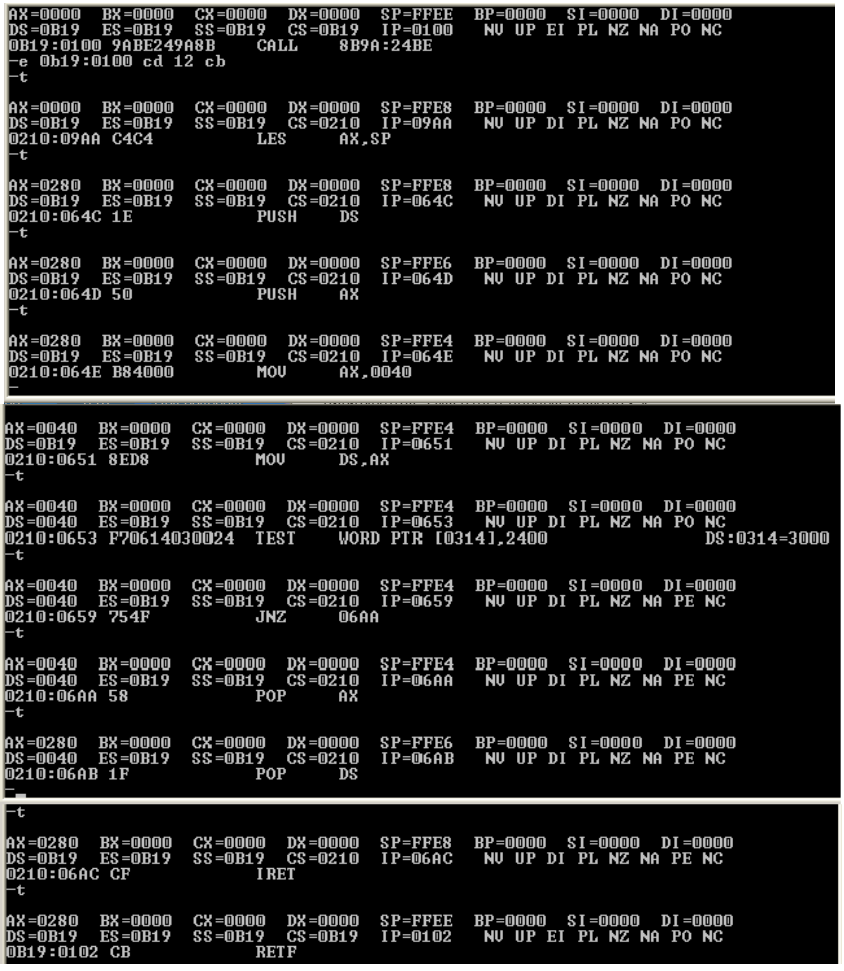
Определяем, что программа-обработчик этого прерывания имеет адрес IP=09AA, CS=0210. Именно по этому адресу должно быть передано управление. Посмотрим, как это происходит.

Запишем в любую область памяти команду, содержащую INT 12h. Она подразумевает вызов обработчика 12h (100102 или 1810) прерывания. Подадим команду R для того, чтобы определить значение регистра CS, а затем введем команду:

_E CS:0100 CD 12 CB <Enter>,

которая должна обеспечить выполнение 12h ($10010_2 = 18_{10}$) прерывания. Проследим последовательность обработки этого прерывания, выполняя трассировку.

-r



Данный фрагмент программы показывает, что после выполнения команды по адресу 0b19:0100, содержащей INT 12h, произошла передача управления на обработку прерывания.

После выполнения этой программы регистр AX будет содержать значение (0280) объема памяти DOS. Это уже известные байты с адресами 0413 и 0414 памяти DOS в шестнадцатеричном формате (см. пример 1.1).

Обратите внимание:

на адреса местонахождения команд программы и передачу управления им;

возможно некоторое расхождение в полученной последовательности команд программы, связанное с тем, что запуск программы Debug осуществлялся из другой версии Windows. Но результат выполнения программы полностью остается тем же.

Ввод ассемблерных программ в память ПК и перевод их в машинный код

Для ввода и перевода программ используются команды Debug' а: _A (Assemble – ассемблирование) и _U (Unassemble – дисассемблирование).

Введем команду

_A 100 <Enter>

По этой команде будет установлен начальный адрес в виде xxxx:0100 (xxxx – означает произвольное, текущее значение регистра CS). После этого можно вводить покомандно всю программу, например:

```
xxxx:0100    MOV AL,25   <Enter>
xxxx:0102    MOV BL,32   <Enter>
xxxx:0104    ADD AL,BL   <Enter>
xxxx:0106    RET         <Enter>
```

После окончания ввода всей программы следует еще раз нажать на клавишу <Enter> и операция ввода будет прекращена.

Введем команду дисассемблирования

_U 0100,106 ,

где 100 и 106 обозначают соответственно начальный и конечный адреса фрагмента программы, подлежавшего переводу в машинный код.

На эту команду будет выдана информация, содержащая ассемблерный и машинный коды программы:

```
xxxx:0100    B025   MOV AL,25
xxxx:0102    B332   MOV BL,32
xxxx:0104    00D8   ADD AL,BL
xxxx:0106    C3     RET
```

Наберем и выполним команду

_D xxxx:0100 <Enter>

Теперь можно рассмотреть соответствие ассемблерных и машинных кодов команд программы, а также их представление в памяти ПК.

Практическое занятие №4 Мультипрограммные вычислительные процессы

Цель работы: Исследовать мультипрограммные вычислительные процессы

Рассматриваемые вопросы:

1 Обобщенная информация о компонентах вычислительного процесса

2 Просмотр и анализ информации о заданиях, процессах и потоках

1 вопрос

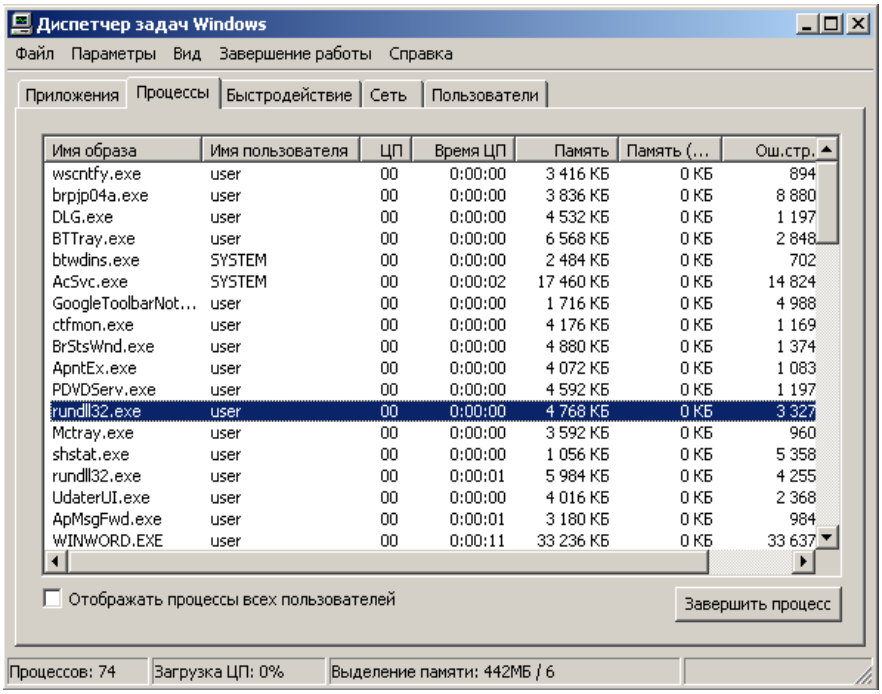
Обобщенная информация о компонентах вычислительного процесса

Современные операционные системы содержат встроенные средства, позволяющие

получить достаточно полную информацию о компонентах вычислительного процесса. Диспетчер задач (Task Manager) операционных систем Windows позволяет получить обобщенную информацию об организации вычислительного процесса с детализацией до выполняющихся прикладных программ (приложений) и процессов. Однако диспетчер задач не позволяет отслеживать потоки.

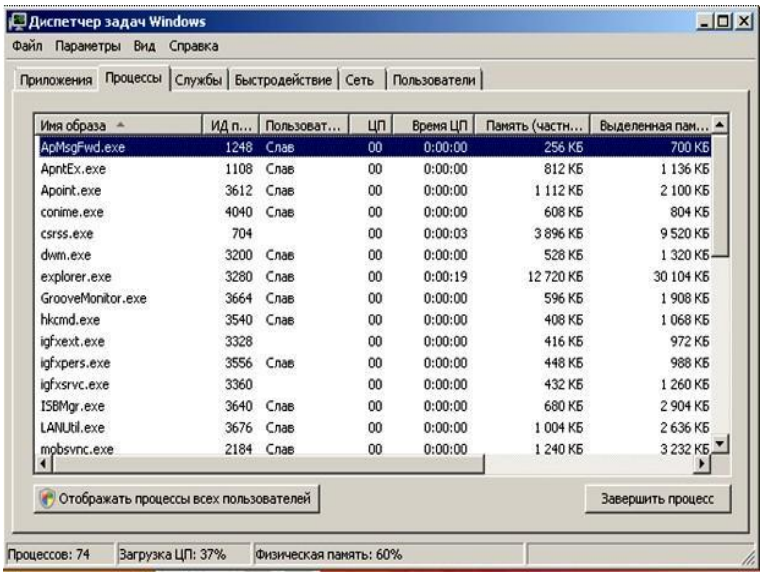
Для запуска диспетчера задач и просмотра компонентов вычислительного процесса нужно выполнить следующие действия.

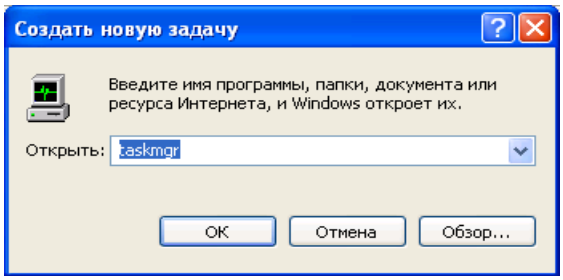
1. Щелкнуть правой кнопкой мыши по панели задач и выбрать строку Диспетчер задач, или нажать клавиши Ctrl + Alt + Del, или нажать последовательно Пуск → Выполнить → taskmgr (рис. 2.1-а Windows XP, рис. 2.1-б Windows 7 и Vista).



2. Для просмотра приложений перейти на вкладку Приложения. Здесь можно завершить приложение (кнопка Снять задачу), переключиться на другое приложение (кнопка Переключиться) и создать новую задачу (кнопка Новая задача). В последнем случае после нажатия кнопки Новая задача в появившемся окне (рис. 2.2) нужно ввести имя задачи.

3. Просмотр (мониторинг) процессов осуществляется переходом на вкладку Процессы, (развернуть окно на весь экран для удобства просмотра). Таблица процессов включает все процессы, запущенные в собственном адресном пространстве, в том числе все приложения и системные сервисы. Обратите внимание на процесс Бездействие системы – фиктивный процесс, занимающий процессор при простое системы.



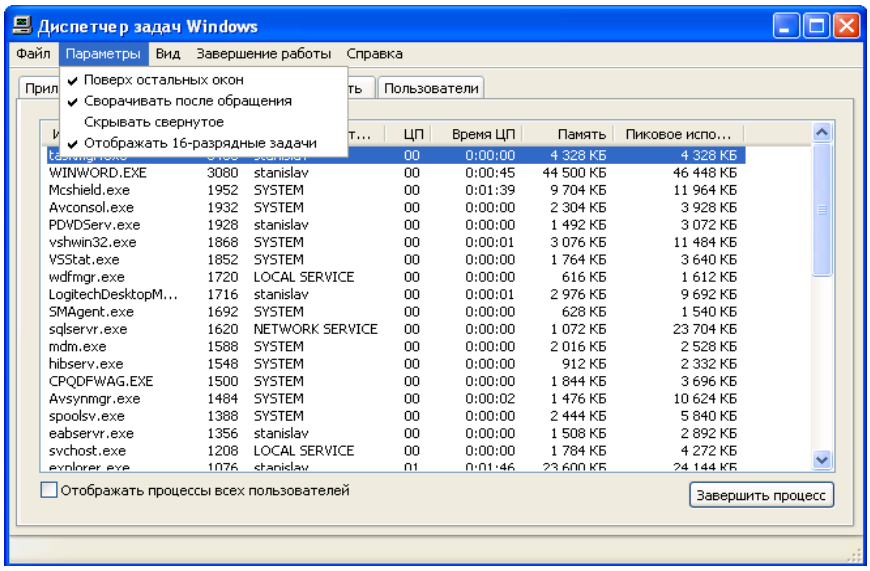


4. Если требуется просмотреть 16-разрядные процессы, то в меню Параметры выбрать команду Отображать 16- разрядные задачи (рис. 2.3).

Для выбора просматриваемых показателей (характеристик) с помощью команды Выбрать столбцы (меню Вид) установить флажки рядом с показателями, которые требуется отображать.

В качестве примера рассмотрим процессы приложения Word. Для этого нужно выполнить следующие действия.

1. Запустить MS Word. Щелкнуть правой клавишей мыши по названию приложения, и в появившемся контекстном меню выбрать строку Перейти к процессам. Произойдет переход на вкладку Процессы. Можно просмотреть число потоков и другие характеристики процесса.



2. Изменить приоритета процесса. На вкладке Процессы щелкнуть правой клавишей мыши по названию процесса и выбрать в контекстном меню строку Приоритет (рис. 2.4). Изменив приоритет, можно увидеть в колонке Базовый приоритет его новое значение (обратить внимание на предупреждение).

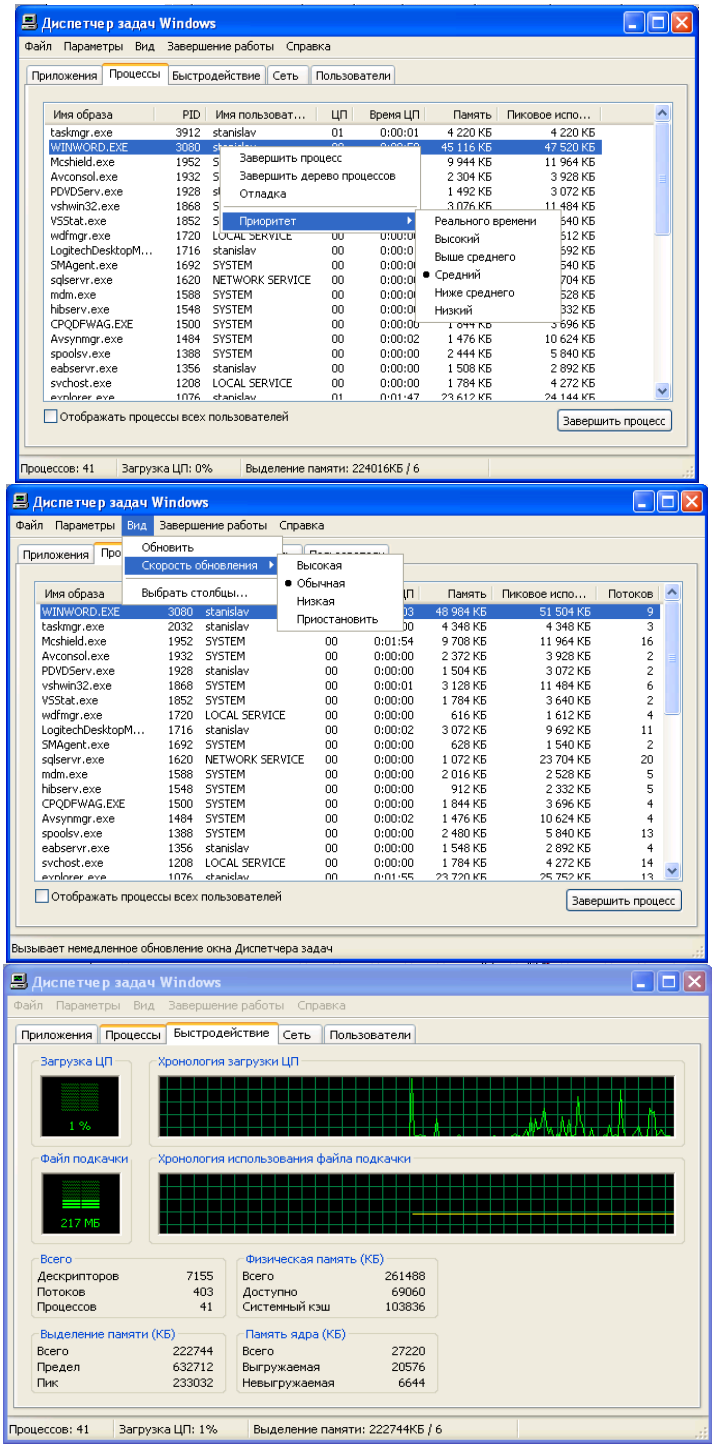
3. Изменить скорости обновления данных. Войти в меню Вид и выбрать команду Скорость обновления (рис. 2.5). Установить требуемую скорость (высокая – каждые полсекунды, обычная – каждую секунду, низкая – каждые 4 секунды, приостановить – обновления нет). Следует иметь ввиду, что с повышением скорости мониторинга возрастают затраты ресурсов компьютера на работу операционной системы, что в свою очередь вносит погрешность в результаты мониторинга.

Диспетчер задач позволяет получить обобщенную информацию об использовании основных ресурсов компьютера. Для этого необходимо:

1. Перейти на вкладку Быстродействие (рис. 2.6). Верхние два окна показывают интегральную загрузку процессора и хронологию загрузки. Нижние два окна те же показатели, но по использованию памяти.

2. Для просмотра использования процессора в режиме ядра (красный цвет) войти в меню Вид и щелкнуть на строке Вывод времени ядра.

В нижней части окна вкладки Быстродействие отображается информация о количестве процессов и потоков, участвующих в мультипрограммном вычислительном процессе, общем количестве дескрипторов (описателей) объектов, созданных операционной системой, а также информация о доступной и выделенной памяти для реализации приложений. Кроме того, приводятся сведения о выделении памяти под ядро операционной системы с указанием выгружаемой и невыгружаемой памяти ядра и объеме системного кэша.



Задание для самостоятельной работы

1. Исследовать мультипрограммный вычислительный процесс на примере выполнения самостоятельно разработанных трех задач (например, заданий по курсу программирования).

2. Для одной из задач определить PID, загрузку ЦП, время ЦП, базовый приоритет процесса, использование памяти, хронологию использования ЦП в режиме ядра. Изменить приоритет процесса и установить влияет ли это на время выполнения приложения.

3. Монопольно выполнить каждую из трех задач, определить время их выполнения. Запустить одновременно (друг за другом) три задачи, определить время выполнения пакета.

Ответить на следующие вопросы:

1. В каком случае суммарное время выполнения задач больше? При последовательном выполнении или одновременном выполнении?

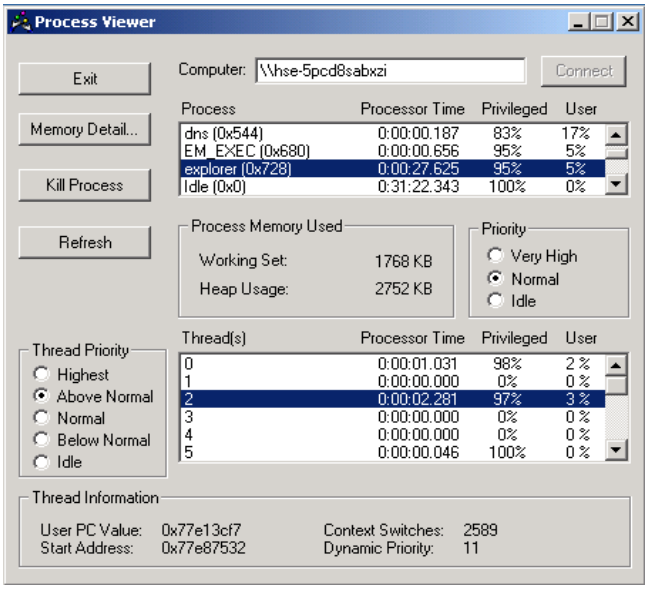
2. Как изменилось время выполнения каждой отдельной задачи?

3. Как изменится время выполнения отдельной задачи при изменении ее приоритета? Окажет ли влияние изменение приоритета одной задачи на время выполнения другой задачи? Объяснить результаты.

2 вопрос

Просмотр и анализ информации о заданиях, процессах и потоках

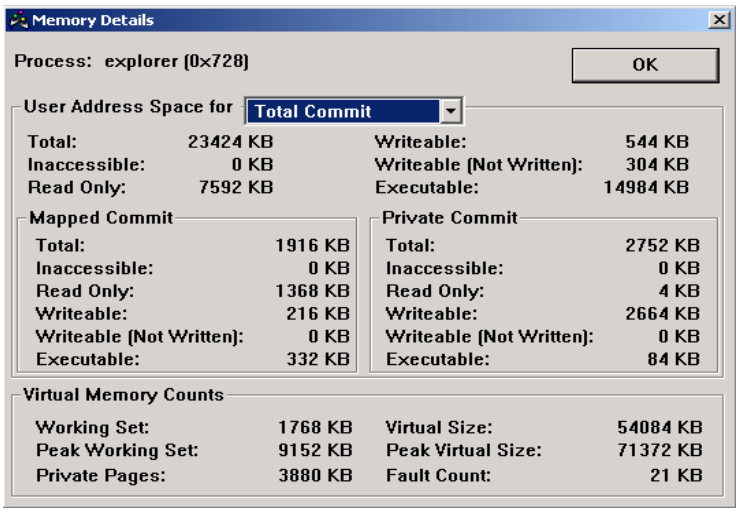
Ряд программ, как производителей операционных систем, так и сторонних фирм позволяют получить более детальную информацию о компонентах вычислительного процесса. [6] рассмотрена программа Process Viewer (Просмотр процесса), которая позволяет увидеть процессы и потоки простым одноступенчатым способом, как на локальном, так и на удаленном компьютере сети.



В верхней части окна перечисляются все процессы, выполняемые на выбранном компьютере. Статистические данные включают количество общего 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.

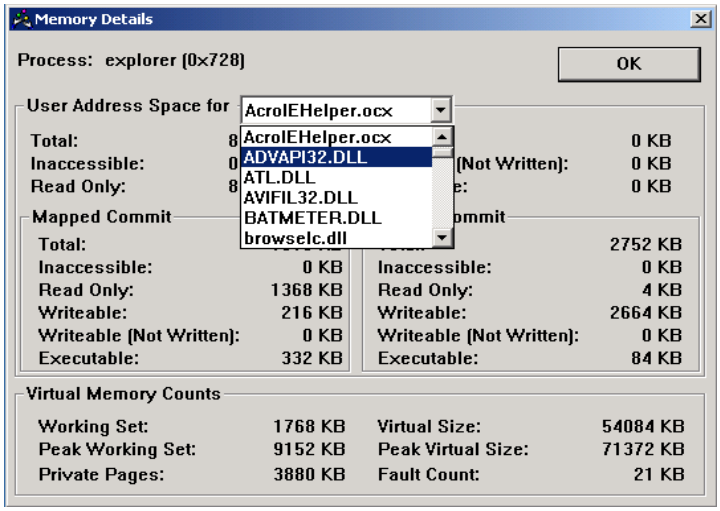
В нижней части окна перечисляются потоки выделенного процесса. Здесь также выводятся статистические данные по использованию процессора, но уже потоками. Слева от перечня потоков даются текущие значения их приоритетов. Поток получает тоже значение приоритета, что и связанный с ним процесс, однако это 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 (рис. 2.8). 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 (рис. 2.9). Выбрав один из этих элементов, можно увидеть, какая память используется данной частью приложения.



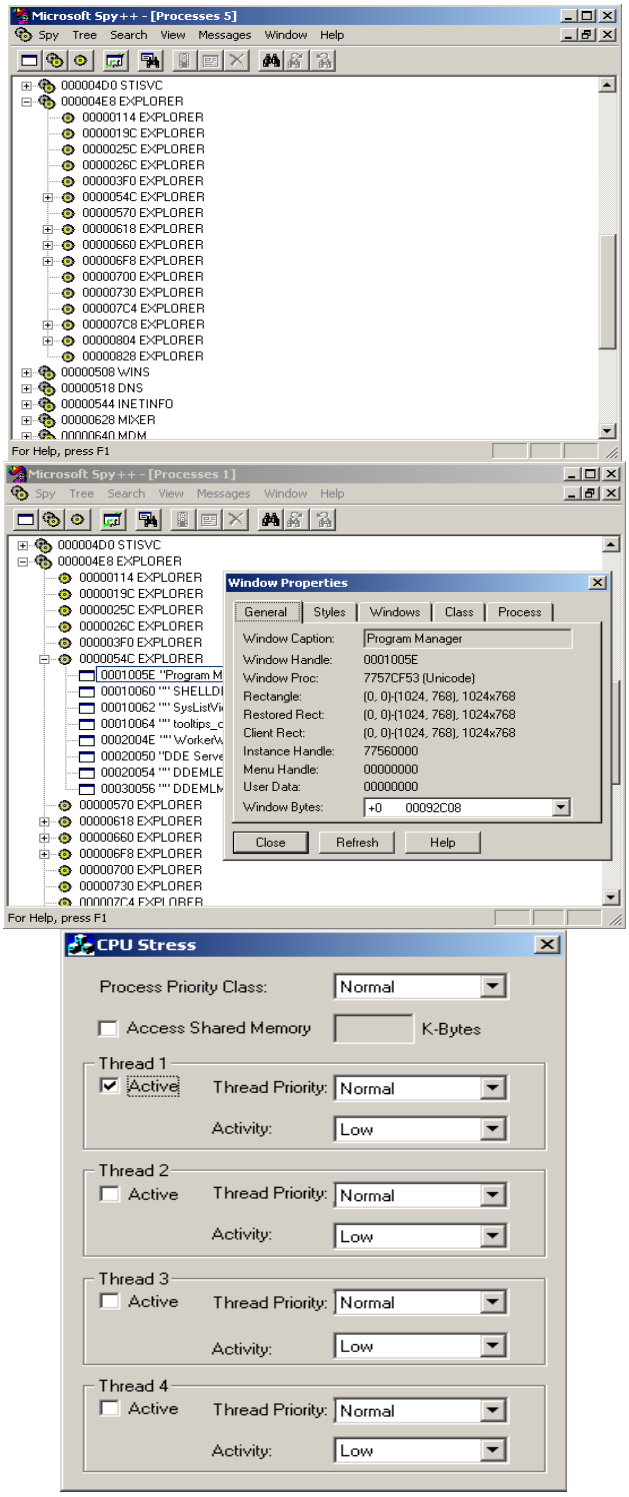
Запустив программу, можно просмотреть с ее помощью текущие характеристики процессов и потоков, в том числе свои задачи.

Производитель операционных систем Windows, компания Microsoft, предлагает утилиту Microsoft Spy++. Программа обладает большими возможностями и поставляется комплектно с операционной системой. Ее можно использовать для просмотра процессов, потоков и ресурсов, используемых ими, в том числе потоков пользовательского интерфейса (UI – потоков).

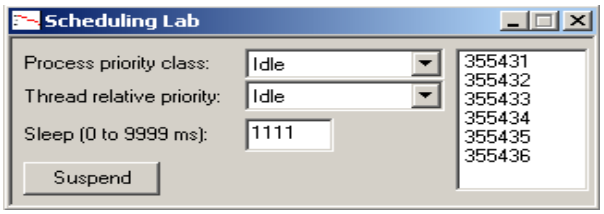


Проиллюстрируем возможности утилиты для просмотра потоков пользовательского интерфейса. Для этого запустим программу в режиме Process (меню Spy -> Processes). Найдем процесс Explorer и щелкнем по значку + слева от него. На экране высветятся потоки процесса. Около потоков UI расположен значок + (рис. 2.10), 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.

Программа CPU Stress, рассмотренная в [6], нагружает процессор, чтобы проконтролировать его работу в заданных условиях. Показано начальное окно программы. 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.



Интересные возможности по изучению вычислительного процесса предоставляет программа Scheduling Lab[1] [11]. Программа позволяет проводить эксперименты с классами приоритетов процессов и относительными приоритетами потоков и исследовать их влияние на общую производительность системы. После запуска программы открывается окно, показанное на рис. 2.13. 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.



.

Для анализа влияния изменения приоритета потока нужно запустить несколько экземпляров (3 – 4) программы и понаблюдать за нагрузкой процессора, используя оснастку Производительность (см. ниже). 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, 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).

Наиболее крупной единицей работы для операционной системы является задание (см. [8]). Программа JobLab [11] позволяет проводить эксперименты по созданию, выполнению, расширению и завершению заданий, состоящих из нескольких процессов (их число определяет пользователь) и потоков. После запуска программы открывается окно, показанное на рис. 2.14.

Запущенная программа создает объект задание с именем JobLab, который можно наблюдать в оснастке Производительность и Диспетчере задач. Программа также создает порт завершения ввода-вывола и связывает с ним объект-задание. Это позволяет отслеживать уведомления от задания и отображать их в списке в нижней части окна.

Изначально в задании нет процессов, и никаких ограничений для него не установлено. Поля в верхней части окна позволяют задавать базовые и расширенные ограничения путем ввода соответствующих значений. Пустое поле означает отсутствие ограничения. Кроме базовых ограничений можно задавать ограничения по пользовательскому интерфейсу. Пометив флажок Preserve Job Time When Applying Limits, можно изменять ограничения, не сбрасывая значений элементов Period Time: User = xx , Kernel = xx . Этот флажок становится недоступным при наложении ограничений на процессорное время для отдельных заданий.

Кнопка Terminate Processes уничтожает все процессы в задании. Кнопка Spawn CMD in Job запускает командный процессор, сопоставляемый с заданием. Из этого процесса можно запускать дочерние процессы (в том числе, предложенные в этом разделе: CPU Stress и Scheduling Lab) и наблюдать, как они ведут себя, став частью задания. Кнопка Put PID in Job позволяет связать существующий свободный процесс (например, свою задачу) с заданием, т.е. включить его в задание.

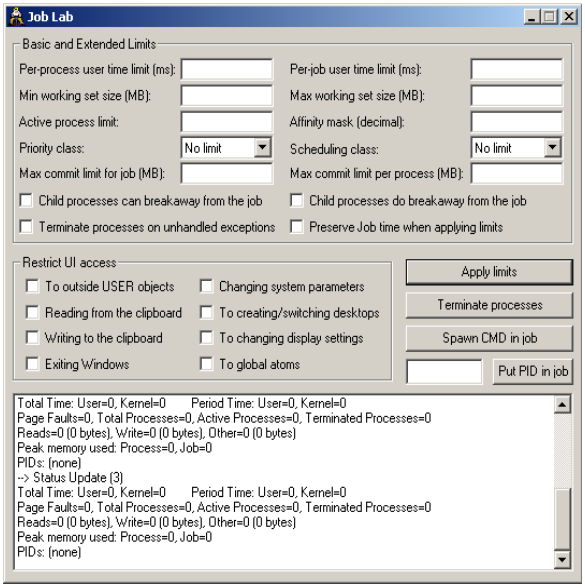
Текст в нижней части окна каждые 10 секунд обновляет информацию о статусе задания: базовые и расширенные сведения, статистику ввода-вывода, а также пиковые объемы памяти, занимаемые процессом и заданием.

Задание для самостоятельной работы

1. Исследовать процесс выполнения задания пользователя, сформированного с помощью программы JobLab, добавляя в задание процессы, запуская кнопкой Spawn CMD in Job программы CPU Stress и Scheduling Lab. Предварительно создать журнал счетчиков (см. ниже п. 2.4), выбрав счетчики, характеризующие использование процессора и основные характеристики процессов.

2. Провести запись журналов в формате CVS, а затем в двоичном формате. Построить диаграммы и графики, используя оснастку Производительность и Excel.

3. Создать журнал Оповещения, выбрав счетчики по своему усмотрению, обосновав решение.


Job Lab
Basic and Extended Limits
Per-process user time limit (ms):
Per-job user time limit (ms):
Min working set size (MB):
Max working set size (MB):
Active process limit:
Affinity mask (decimal):
Priority class:
No limit
Scheduling class:
No limit
Max commit limit for job (MB):
Max commit limit per process (MB):
Child processes can breakaway from the job
Child processes do breakaway from the job
Terminate processes on unhandled exceptions
Preserve Job time when applying limits
Restrict UI access
To outside USER objects
Changing system parameters
Reading from the clipboard
To creating/switching desktops
Writing to the clipboard
To changing display settings
Exiting Windows
To global atoms
Apply limits
Terminate processes
Spawn CMD in job
Put PID in job
Total Time: User=0, Kernel=0 Period Time: User=0, Kernel=0
Page Faults=0, Total Processes=0, Active Processes=0, Terminated Processes=0
Reads=0 (0 bytes), Write=0 (0 bytes), Other=0 (0 bytes)
Peak memory used: Process=0, Job=0
PIDs: (none)
--> Status Update (3)
Total Time: User=0, Kernel=0 Period Time: User=0, Kernel=0
Page Faults=0, Total Processes=0, Active Processes=0, Terminated Processes=0
Reads=0 (0 bytes), Write=0 (0 bytes), Other=0 (0 bytes)
Peak memory used: Process=0, Job=0
PIDs: (none)

Практическое занятие № 5. Детальное исследование мультипрограммного вычислительного процесса

Цель работы: Исследование, запись и представление результатов анализа вычислительного процесса

Рассматриваемые вопросы:

1 Детальное исследование мультипрограммного вычислительного процесса.

2 Запись и представление результатов анализа вычислительного процесса

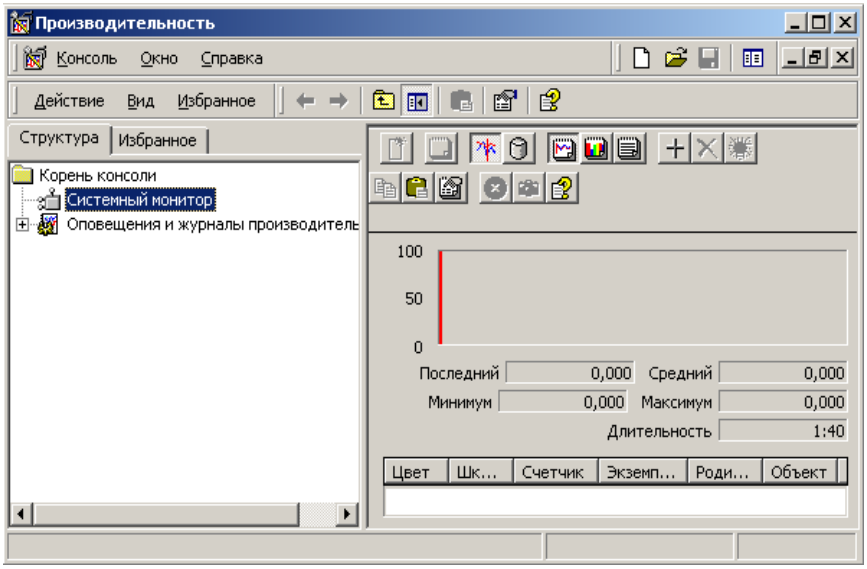
1 вопрос

В операционной системах Windows имеются средства, позволяющие детально анализировать вычислительные процессы. К таким средствам относится Системный монитор и Оповещения и журналы производительности. Для доступа к этим средствам нужно выполнить последовательность действий

Пуск -> Программы -> Администрирование -> Производительность

Откроется окно Производительность, содержащее две оснастки: Системный монитор и Оповещения и журналы производительности (рис. 2.15). 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 Консоль 1, в котором можно выбрать нужную оснастку.

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



Для просмотра и выбора объектов мониторинга и настройки счетчиков нужно выполнить следующие действия.

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 сек.;

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 (рис. 2.16).

Для удобства работы предусмотрена настройка вида отображаемой информации. 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 (рис. 2.17).

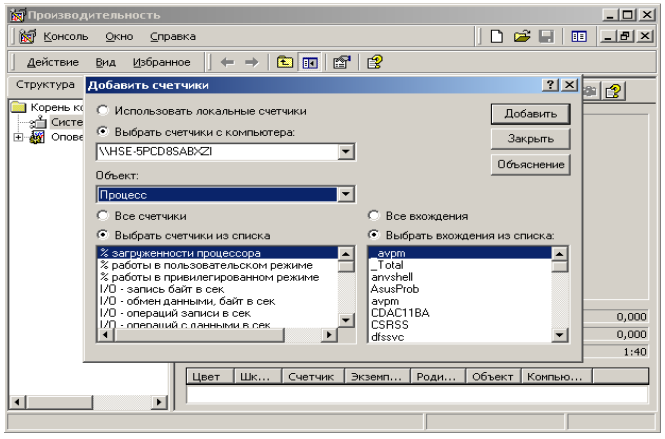
На вкладке Общие можно задать вид информации (график, гистограмма, отчет), отображаемые элементы (легенда, строка значений, панель инструментов), данные отчета и гистограммы (максимальные, минимальные и т. д), период обновления данных и др.

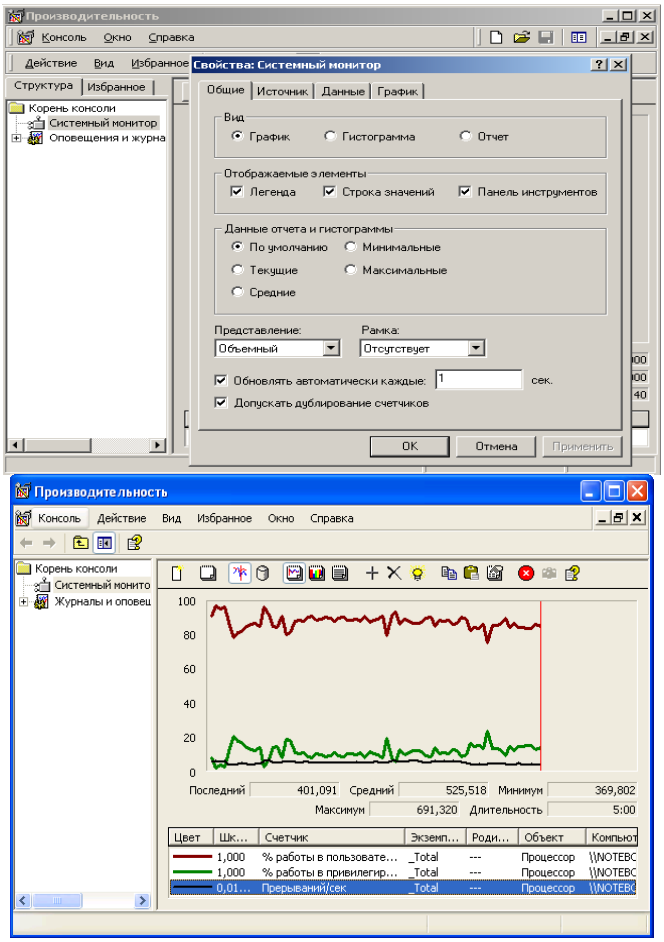
На вкладке Источник задается источник данных. На вкладке Данные можно для каждого счетчика задать цвет, ширину линии, масштаб и др.

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.

Полученная с помощью Монитора производительности информация позволяет наглядно произвести экспресс-анализ функционирования нужного компонента вычислительного процесса или устройства компьютера. Например, на рис. 2.18 показана следующая информация об использовании процессора: процент работы процессора в пользовательском и привилегированном режимах и количество прерываний в секунду.





Задание для самостоятельной работы

1. Запустить предварительно разработанную программу. Используя возможности оснастки Производительность, получить диаграммы, характеризующие использование процессора при его нагрузке различным количеством потоков, меняя их активность и уровни приоритета.

2. Исследовать свои задачи (приложения). Определить характеристики процессов: % загрузки процессора (в пользовательском и привилегированном режиме), % времени прерываний, количество прерываний, базовый приоритет, обращения к диску, время выполнения процесса.

2.4. Запись и представление результатов анализа вычислительного процесса

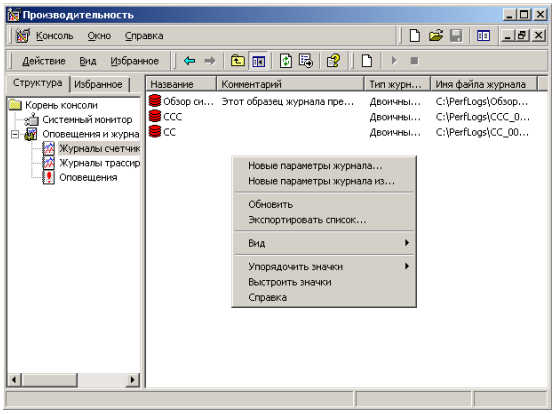
Оснастка Оповещения и журналы производительности содержит три компонента: Журналы счетчиков, Журналы трассировки и Оповещения, которые можно использовать для записи и просмотра результатов исследования вычислительного процесса. Данные, созданные при помощи оснастки можно просматривать как в процессе сбора, так и после его окончания.

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

запустить оснастку Производительность;

дважды щелкнуть на значке Оповещения и журналы производительности;

выбрать значок Журналы счетчиков, щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт Новые параметры журнала (рис. 2.19);



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

в новом окне (рис. 2.20) на вкладке Общие добавить нужные счетчики и установить интервал съема данных;

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 для самостоятельной работы

1. Исследовать свои приложения с записью результатов в Журнал счетчиков, выбрав следующие счетчики: % загруженности, работы процессора в привилегированном и пользовательском режимах, % времени прерываний, % использования выделенной памяти, частота обращений к диску, скорость обмена с диском.

2. Выполнить следующие действия:

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

Запустить исследуемую программу.

Через 2 – 3 мин. Остановить журнал.

Просмотреть Результаты, открыв файл журнала в Excel (для удобства просмотра нужно, используя меню Формат, дать Перенос по словам для заголовков и растянуть ширину ячеек с числовыми данными, пример на рис. 2.21). Объяснить полученные результаты.

Исследовать программу еще раз, указав тип журнала – двоичный (чтобы потом можно было просмотреть диаграммы).

3. Просмотреть собранную информацию в консоли Производительность. Для этого выполнить следующие действия:

дважды щелкнуть по значку Системный монитор;

щелкнуть правой клавишей мыши в правом поле и выбрать в контекстном меню строку Свойства (рис. 2.22);

перейти на вкладку Источник;

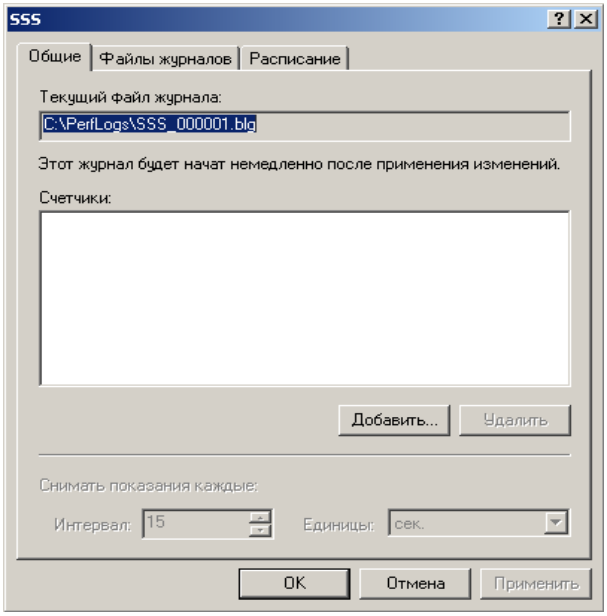
щелкнуть на кнопке Файл журнала и указать его размещение, используя кнопку Обзор;

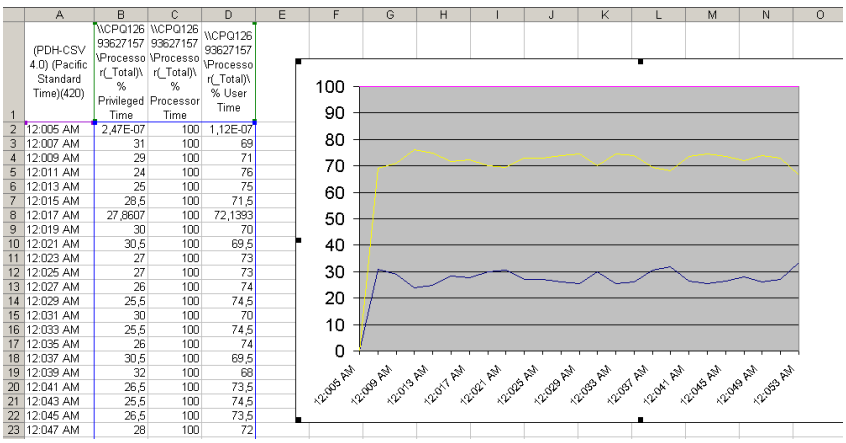
нажать кнопку Диапазон времени и выбрать диапазон представления результатов, передвигая левую и правую планки;

нажать кнопку OK;

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

просмотреть полученные диаграммы. Объяснить полученные результаты.





	A	B	C	D
1	(PDH-CSV 4.0) (Pacific Standard Time)(420)	\\CPQ12693627157\Processor(_Total)\% Privileged Time	\\CPQ12693627157\Processor(_Total)\% Processor Time	\\CPQ12693627157\Processor(_Total)\% User Time
2	12:005 AM	2,47E-07	100	1,12E-07
3	12:007 AM	31	100	69
4	12:009 AM	29	100	71
5	12:011 AM	24	100	76
6	12:013 AM	25	100	75
7	12:015 AM	28,5	100	71,5
8	12:017 AM	27,8607	100	72,1393
9	12:019 AM	30	100	70
10	12:021 AM	30,5	100	69,5
11	12:023 AM	27	100	73
12	12:025 AM	27	100	73
13	12:027 AM	26	100	74
14	12:029 AM	25,5	100	74,5
15	12:031 AM	30	100	70
16	12:033 AM	25,5	100	74,5
17	12:035 AM	26	100	74
18	12:037 AM	30,5	100	69,5
19	12:039 AM	32	100	68
20	12:041 AM	26,5	100	73,5
21	12:043 AM	25,5	100	74,5
22	12:045 AM	26,5	100	73,5
23	12:047 AM	28	100	72

2 вопрос

Запись и представление результатов анализа вычислительного процесса

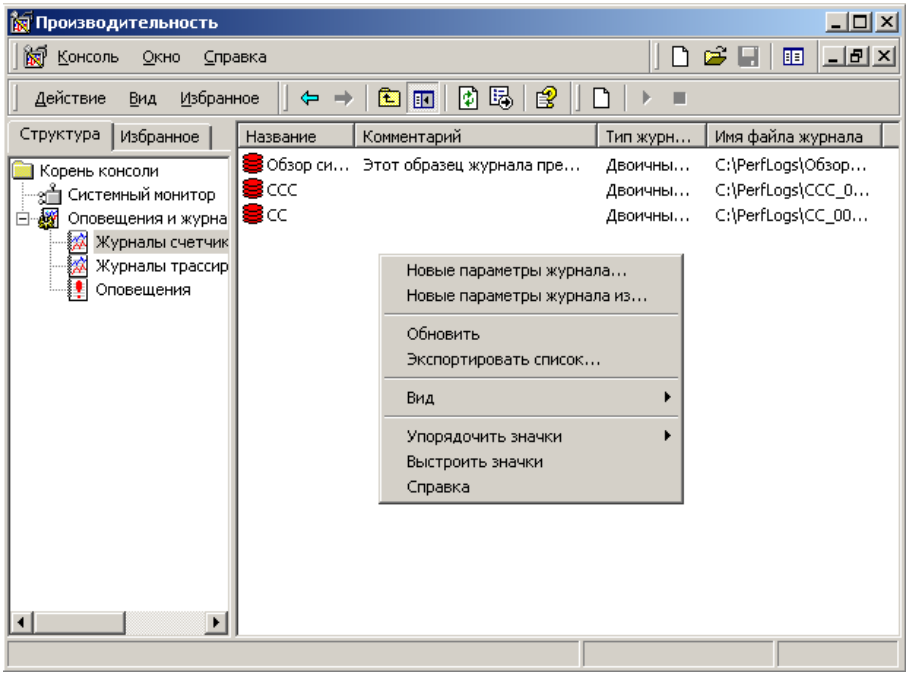
Оснастка Оповещения и журналы производительности содержит три компонента: Журналы счетчиков, Журналы трассировки и Оповещения, которые можно использовать для записи и просмотра результатов исследования вычислительного процесса. Данные, созданные при помощи оснастки можно просматривать как в процессе сбора, так и после его окончания.

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

запустить оснастку Производительность;

дважды щелкнуть на значке Оповещения и журналы производительности;

выбрать значок Журналы счетчиков, щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт Новые параметры журнала (рис. 2.19);



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

в новом окне (рис. 2.20) на вкладке Общие добавить нужные счетчики и установить интервал съема данных;

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 для самостоятельной работы

1. Исследовать свои приложения с записью результатов в Журнал счетчиков, выбрав следующие счетчики: % загруженности, работы процессора в привилегированном и пользовательском режимах, % времени прерываний, % использования выделенной памяти, частота обращений к диску, скорость обмена с диском.

2. Выполнить следующие действия:

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

Запустить исследуемую программу.

Через 2 – 3 мин. Остановить журнал.

Просмотреть Результаты, открыв файл журнала в Excel (для удобства просмотра нужно, используя меню Формат, дать Перенос по словам для заголовков и растянуть ширину ячеек с числовыми данными, пример на рис. 2.21). Объяснить полученные результаты.

Исследовать программу еще раз, указав тип журнала – двоичный (чтобы потом можно было просмотреть диаграммы).

3. Просмотреть собранную информацию в консоли Производительность. Для этого выполнить следующие действия:

дважды щелкнуть по значку Системный монитор;

щелкнуть правой клавишей мыши в правом поле и выбрать в контекстном меню строку Свойства (рис. 2.22);

перейти на вкладку Источник;

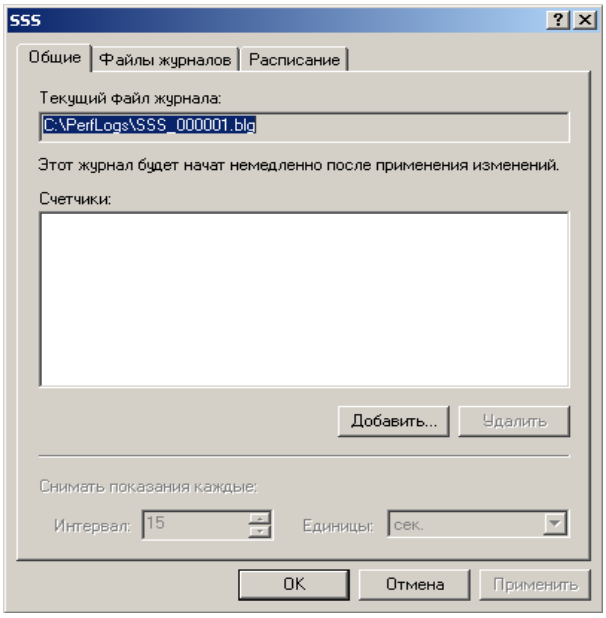
щелкнуть на кнопке Файл журнала и указать его размещение, используя кнопку Обзор;

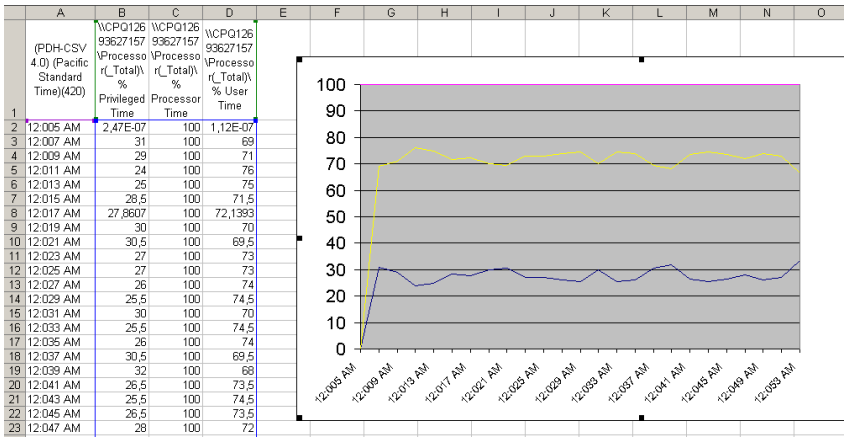
нажать кнопку Диапазон времени и выбрать диапазон представления результатов, передвигая левую и правую планки;

нажать кнопку OK;

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

просмотреть полученные диаграммы. Объяснить полученные результаты.





	A	B	C	D
1	(PDH-CSV 4.0) (Pacific Standard Time)(420)	\\CPQ12693627157\Processor(_Total)\% Privileged Time	\\CPQ12693627157\Processor(_Total)\% Processor Time	\\CPQ12693627157\Processor(_Total)\% User Time
2	12:005 AM	2,47E-07	100	1,12E-07
3	12:007 AM	31	100	69
4	12:009 AM	29	100	71
5	12:011 AM	24	100	76
6	12:013 AM	25	100	75
7	12:015 AM	28,5	100	71,5
8	12:017 AM	27,8607	100	72,1393
9	12:019 AM	30	100	70
10	12:021 AM	30,5	100	69,5
11	12:023 AM	27	100	73
12	12:025 AM	27	100	73
13	12:027 AM	26	100	74
14	12:029 AM	25,5	100	74,5
15	12:031 AM	30	100	70
16	12:033 AM	25,5	100	74,5
17	12:035 AM	26	100	74
18	12:037 AM	30,5	100	69,5
19	12:039 AM	32	100	68
20	12:041 AM	26,5	100	73,5
21	12:043 AM	25,5	100	74,5
22	12:045 AM	26,5	100	73,5
23	12:047 AM	28	100	72

Практическое занятие № 6. Создание журналов трассировки и оповещений

Цель работы: Научиться создавать журналы трассировки и оповещений

Рассматриваемые вопросы:

1.Создание журналов трассировки оповещений.

1 вопрос

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:

запустить оснастку Производительность;

щелкнуть на значке Журналы трассировки;

щелкнуть правой кнопкой мыши в панели результатов и выбрать в контекстном меню пункт Новые параметры журнала (рис. 2.23);

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

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

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

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

на вкладке Файлы журналов выбрать тип журнала:

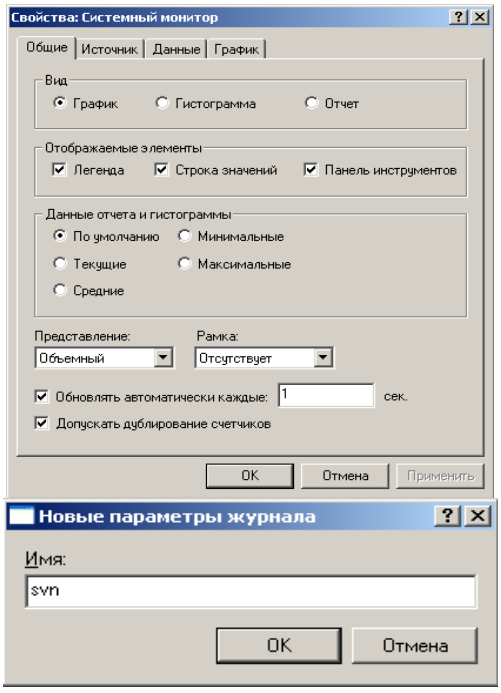
- файл циклической трассировки (журнал с перезаписью событий, расширение etl);

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

на этой же вкладке выбрать и размер файла;

на вкладке Дополнительно можно указать размер буфера журнала;

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



В ряде случаев для обнаружения неполадок в организации вычислительного процесса удобно использовать оповещения. С помощью этого компонента можно установить 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:

щелкнуть на значке Оповещения;

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

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

в появившемся окне (рис. 2.24) на вкладке Общие можно задать комментарий к оповещению и выбрать нужные счетчики;

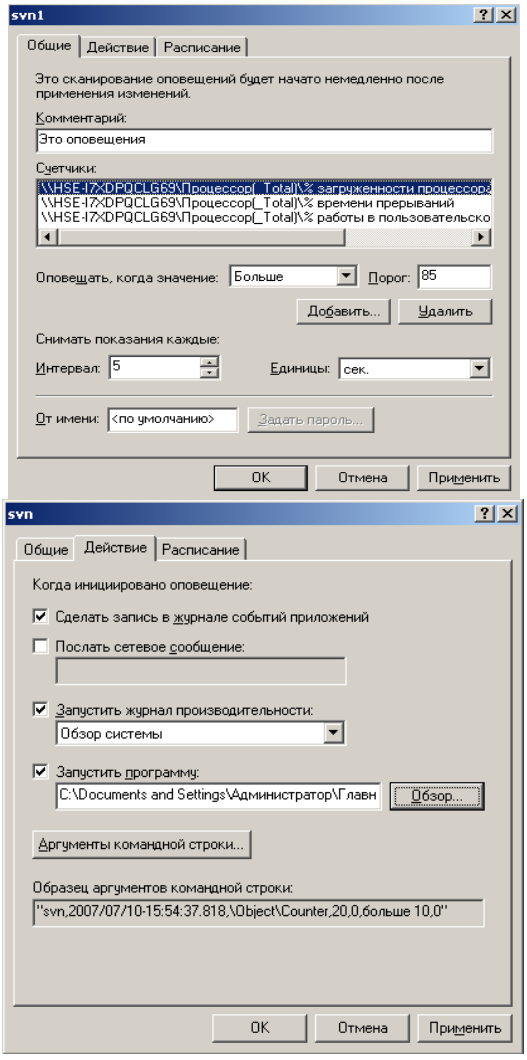
в поле Оповещать, когда значение выбрать предельные значения для счетчиков;

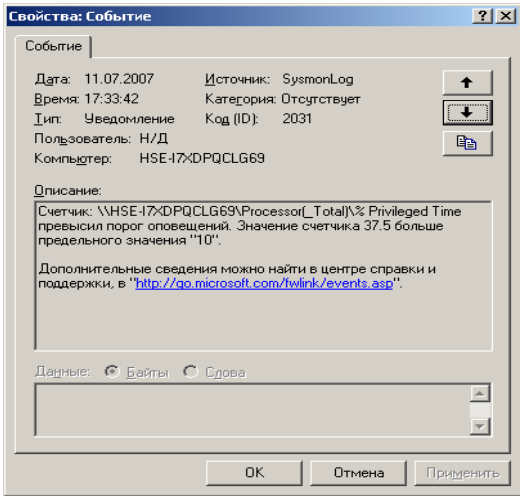
в поле Снимать показания каждые выбрать период опроса счетчиков;

на вкладке Действие (рис. 2.25) можно выбрать действие, которое будет происходить при запуске оповещения, например, Послать сетевое сообщение и указать имя компьютера;

на вкладке Расписание выбрать режим запуска и остановки наблюдения.

Если в компьютере произойдет событие, предусмотренное в Оповещениях, в журнал событий Приложение, будет сделана соответствующая запись. Для ее просмотра нужно зайти в оснастку Просмотр событий, где и можно увидеть сведения о событии, например такие, как показано на рис. 2.26.





Базовый уровень

1. Создать журнал трассировки для исследования своего приложения.
2. Создать Оповещения по выбранным счетчикам для своего приложения.

Повышенный уровень

3. Просмотреть журнал событий.
4. Объяснить полученные результаты.

Практическое занятие № 7.Исследование проблем мультипрограммных вычислительных процессов.

Цель работы: Изучение принципов работы режимов First Come – First Served, Round Robin и Shortest Job First.

В учебных материалах данной лабораторной работы используется программа «Autoplanner.exe», разработанная в 2015 году студентом группы КМБ-с-о-121 Карасевым Иваном, которая позволяет моделировать следующие алгоритмы распределения времени центрального процессора очереди выполняемых процессов:

- Round Robin (RR) – «круговая дисциплина обслуживания»;
- First Come – First Served (FCFS) – «первый пришел – первый обслужен»;
- Shortest Job First (SJF) – «короткая задача – первая».

Окно программы после ее запуска показано на рис. 2.30. В этом окне необходимо выбрать, какой алгоритм мы будем исследовать.

Рис. 2.30

После выбора необходимого алгоритма, например, First Come – First Served, открывается окно (рис. 2.31), в котором необходимо задать необходимые параметры для моделирования работы планировщика.

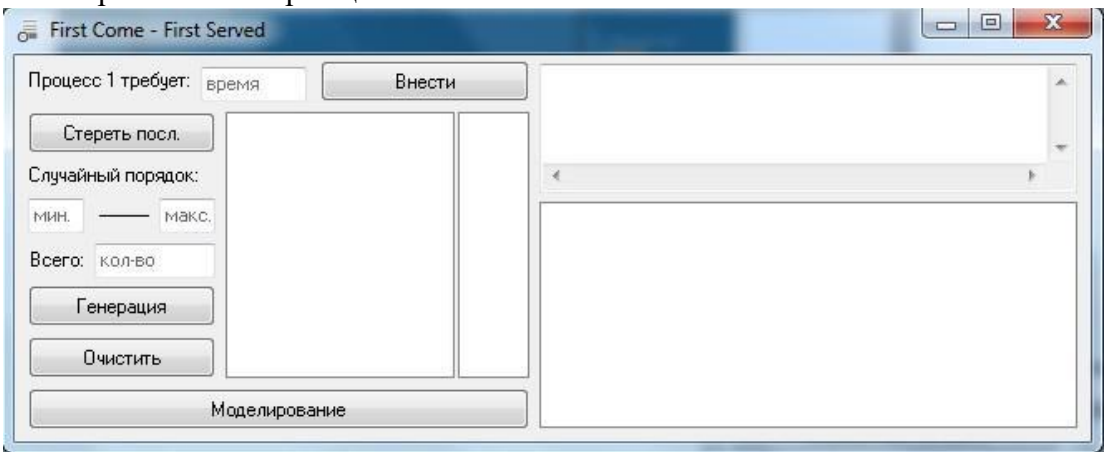


Рис. 2.31

Здесь, помимо возможности ввода времени выполнения процессов вручную есть возможность автоматической генерации. Для начала попробуем ввести 3 процесса вручную. Допустим, мы будем моделировать в секундах и нашему первому процессу для выполнения требуется 7 секунд, второму – 3 секунды, а третьему – 5 секунд. Для этого в поле «время» вводим число 7 и нажимаем кнопку «Внести». Тоже самое делаем с 3 и 5 (рис. 2.32).

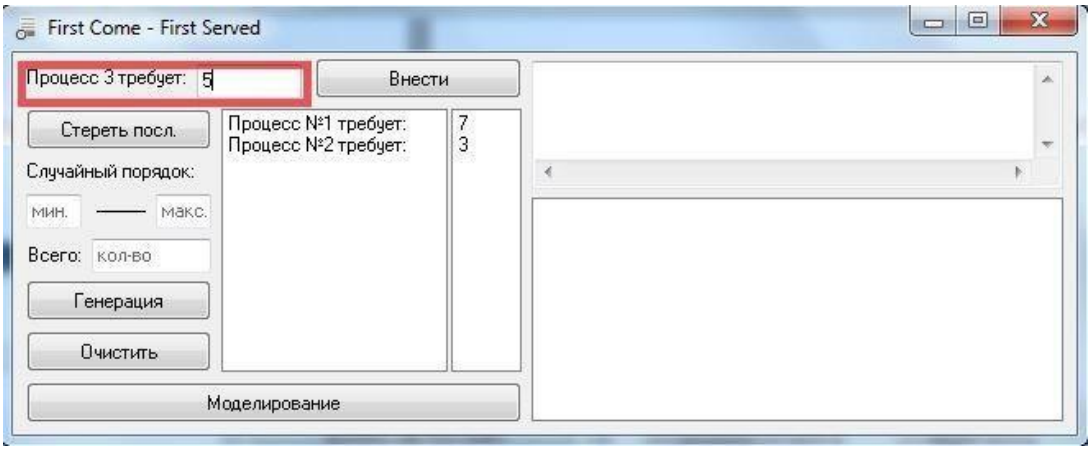


Рис. 2.32

Убедитесь, что данные введены правильно и нажмите на кнопку «Моделирование». В правой части окна будет выведен результат моделирования с расчетом дополнительных параметров (Рис. 2.33).

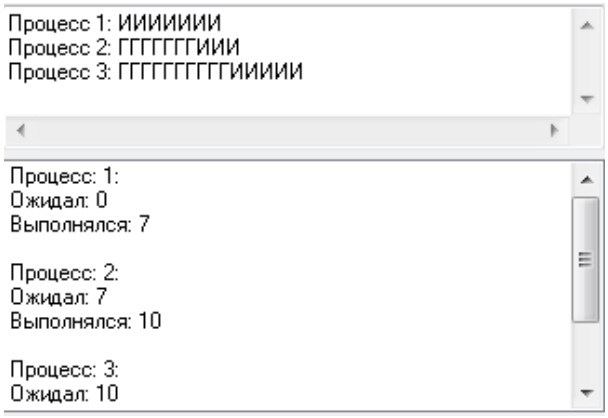


Рис. 2.33

В окне на рисунке 2.33 буква И означает, что процесс исполняется, а буква Г – находится в режиме готовности. Таким образом, получаем следующую картину: первые 7 секунд выполнялся процесс 1, а процессы 2 и 3 находились в режиме готовности. После этого процесс 1 завершился и больше не нуждался в системном времени и свою работу начал процесс 2 и выполнялся ровно 3 секунды, после чего запустился последний – процесс 3 на 5 секунд.

В этом окне ниже рассчитано время ожидания каждого процесса, полное время выполнения (сумма времени ожидания и выполнения), а также среднее время ожидания процессов (среднее арифметическое всех времен ожидания) и среднее полное время выполнения (среднее арифметическое всех полных времен выполнения.

Теперь воспользуемся автогенерацией времен процессов, для этого нажимаем кнопку «Очистить», чтобы удалить заданные в прошлом примере процессы. В поля «Случайный порядок» вводим минимальное и максимальное значения времен, то есть диапазон, в котором хотели бы видеть наши параметры. В поле «Всего» вводим общее количество процессов для генерации.

Например, мы хотим сгенерировать 10 процессов со временем выполнения от 5 секунд – до 15. Для этого заполняем поля так, как это показано на Рис. 2.34 и нажимаем на кнопку «Генерация».

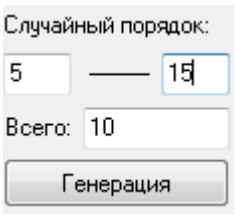


Рис. 2.34

Далее нажимаем на кнопку «Моделирование» и получаем результат, подобно прошлому примеру (Рис. 2.35).

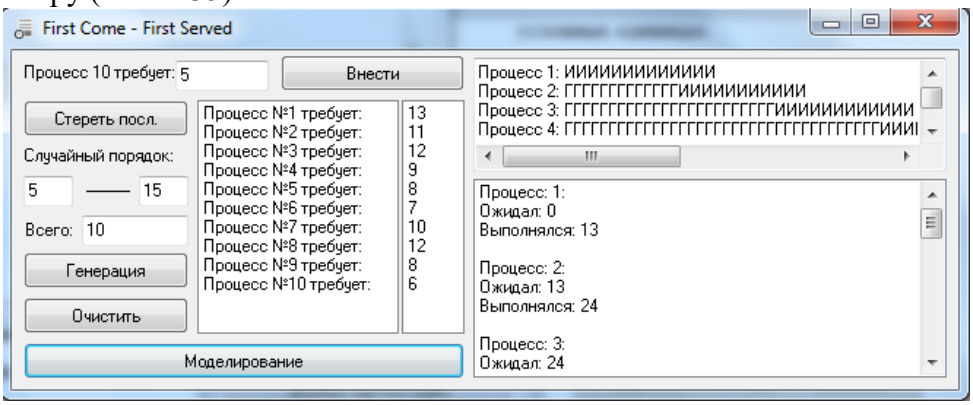


Рис. 2.35

Таким образом, нам удалось разобраться, как работать внутри данной программы!

После завершения работы с алгоритмом рекомендую нажать на кнопку «Очистить» и закрывать окно алгоритма, после чего выбираем следующий (Round Robin показан на Рис. 2.36, Shortest Job First на Рис. 2.37).

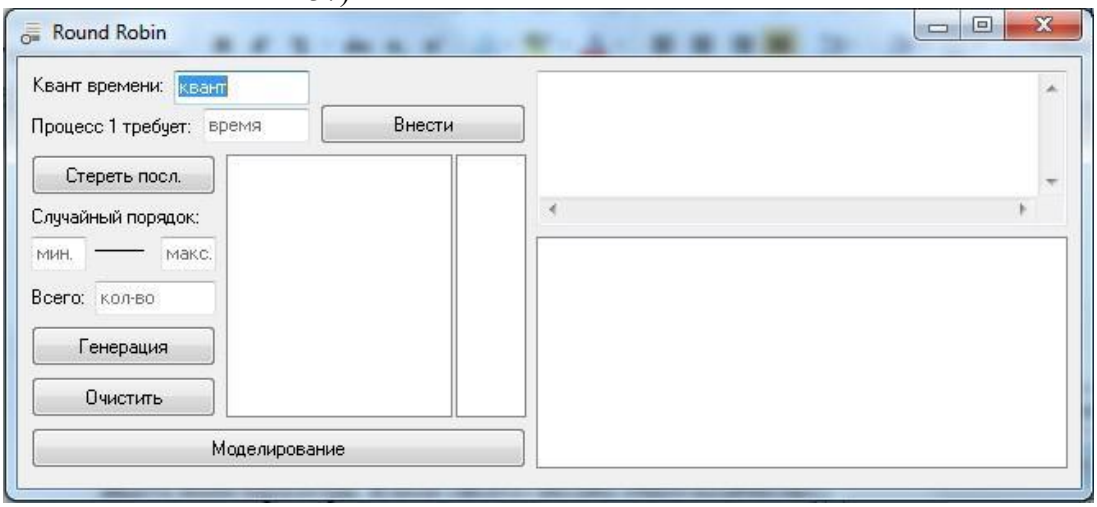


Рис. 2.36

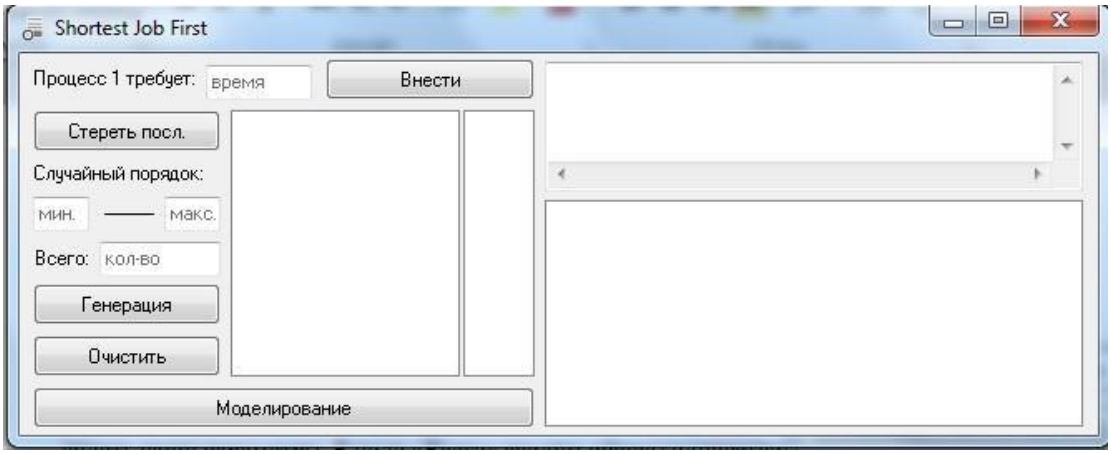


Рис. 2.37

Как мы видим, задаваемые параметры в Shortest Job First ничем не отличаются от рассмотренного примера. В Round Robin добавлено новое поле «Квант времени», в которое мы указываем, квант времени, который будет выделяться для выполнения процессов в циклическом алгоритме.

Задание для самостоятельной работы

1. Смоделировать ситуацию в соответствии со своим вариантом и объяснить полученные результаты.

Вариант №	Модель 1	Модель 2
1	**Режим:** FCFS Процесс 1: 7 Процесс 2: 3 Процесс 3: 5	**Режим:** RR (квант – 2) Процесс 1: 4 Процесс 2: 3 Процесс 3: 2
2	**Режим:** FCFS Процесс 1: 12 Процесс 2: 1 Процесс 3: 3	**Режим:** SJF Процесс 1: 7 Процесс 2: 3 Процесс 3: 5
3	**Режим:** RR (квант – 3) Процесс 1: 11 Процесс 2: 2 Процесс 3: 9	**Режим:** FCFS Процесс 1: 2 Процесс 2: 7 Процесс 3: 11
4	**Режим:** SJF Процесс 1: 3 Процесс 2: 3 Процесс 3: 15	**Режим:** FCFS Процесс 1: 11 Процесс 2: 1 Процесс 3: 2
5	**Режим:** RR (квант – 1) Процесс 1: 9 Процесс 2: 5 Процесс 3: 7	Режим: SJF Процесс 1: 7 Процесс 2: 3 Процесс 3: 5
6	**Режим:** SJF Процесс 1: 3 Процесс 2: 3 Процесс 3: 4	**Режим:** FCFS Процесс 1: 7 Процесс 2: 9 Процесс 3: 2
7	**Режим:** SJF Процесс 1: 12 Процесс 2: 4 Процесс 3: 8	**Режим:** RR (квант – 4) Процесс 1: 11 Процесс 2: 3 Процесс 3: 5
8	**Режим:** SJF Процесс 1: 2 Процесс 2: 6 Процесс 3: 14	**Режим:** FCFS Процесс 1: 1 Процесс 2: 7 Процесс 3: 9
9	**Режим:** SJF Процесс 1: 4 Процесс 2: 9 Процесс 3: 12	**Режим:** RR (квант – 2) Процесс 1: 5 Процесс 2: 6 Процесс 3: 11
10	**Режим:** RR (квант – 1) Процесс 1: 1 Процесс 2: 7 Процесс 3: 15	**Режим:** FCFS Процесс 1: 2 Процесс 2: 6 Процесс 3: 10

2. Смоделировать ситуацию в соответствии со своим вариантом (с использованием генератора) и объяснить полученные результаты.

Вариант №	**Модель 1**	**Модель 2**
1	Режим: FCFS	Режим: RR (квант – 5)

	Количество: 3 Диапазон: 10 – 20	Количество: 3 Диапазон: 5 – 15
2	Режим: RR (квант – 8) Количество: 7 Диапазон: 2 – 7	Режим: SJF Количество: 11 Диапазон: 3 – 8
3	Режим: SJF Количество: 5 Диапазон: 10 – 15	Режим: FCFS Количество: 12 Диапазон: 2 – 4
4	Режим: RR (квант – 4) Количество: 6 Диапазон: 3 – 8	Режим: FCFS Количество: 10 Диапазон: 2 – 10
5	Режим: SJF Количество: 8 Диапазон: 2 – 10	Режим: (квант – 2) Количество: 12 Диапазон: 2 – 4
6	Режим: SJF Количество: 2 Диапазон: 5 – 10	Режим: RR (квант – 4) Количество: 5 Диапазон: 10 – 15
7	Режим: RR (квант – 3) Количество: 9 Диапазон: 7 – 9	Режим: FCFS Количество: 11 Диапазон: 2 – 4
8	Режим: RR (квант – 6) Количество: 3 Диапазон: 4 – 12	Режим: SJF Количество: 8 Диапазон: 7 - 13
9	Режим: SJF Количество: 5 Диапазон: 2 - 8	Режим: RR (квант – 7) Количество: 12 Диапазон: 7 - 11
10	Режим: FSFS Количество: 4 Диапазон: 10 – 20	Режим: SJF Количество: 6 Диапазон: 5 – 20

3. (ПОВЫШЕННЫЙ УРОВЕНЬ) Смоделировать случайную ситуацию (на свой выбор) во всех трех режимах работы и сформировать отчет о быстродействии работы того или иного алгоритма. В отчете должно отражаться среднее полное время выполнения каждого процесса в каждом алгоритме, а также среднее время ожидания. Какой алгоритм в вашем случае использовать выгоднее и почему?

4. Разработать программу, с помощью которой можно сравнить эффективность обработки потоков на основе квантования и дисциплины FIFO..

Контрольные вопросы

1. Поясните принцип работы режима First Come – First Served.
2. Поясните принцип работы режима Round Robin.
3. Что такое квант времени в режиме Round Robin?
4. Поясните принцип работы режима Shortest Job First.
5. В чем заключается отличие работы алгоритма First Come – First Served от Shortest Job First?
6. Пусть на входе стека находятся 3 процесса: первый требует 7 секунд системного времени, второй – 3 секунды, третий – 5 секунд. В каком порядке будут выполняться процессы в режиме работы First Come – First Served?
7. Пусть на входе стека находятся 3 процесса: первый требует 9 секунд системного времени, второй – 6 секунды, третий – 7 секунд (квант времени неважен). В каком порядке будут выполняться процессы в режиме работы Round Robin?
8. Пусть на входе стека находятся 3 процесса: первый требует 7 секунд системного времени, второй – 3 секунды, третий – 5 секунд. В каком порядке будут выполняться процессы в режиме работы Shortest Job First?

Практическое занятие № 8 Создание потоков в приложении

Цель работы: Исследовать состояния вычислительных потоков

Рассматриваемые вопросы:

1 Создание потоков в приложении 2 Приоритеты потоков

3 Состояния потоков

4 Порядок выполнения потоков

5 Последовательное выполнение потоков 1 вопрос

Все потоки мультипрограммного вычислительного процесса должны иметь доступ к системным ресурсам – кучам, портам, файлам, окнам и т. д. Потоки взаимодействуют друг с другом в двух основных случаях: совместно используя разделяемый ресурс, и уведомляя друг друга о завершении каких-либо операций. С вопросами синхронизации потоков связана обширная тематика, которую мы частично рассмотрим. Предварительно нужно научиться создавать многопоточные процессы. Наиболее просто это сделать, используя современные системы программирования, например, MS Visual Studio 2005 и языковые средства (например, C#), предоставляемые этой системой.

Для создания нового потока в среде .NET нужно создать экземпляр класса Thread. Конструктор этого класса принимает один параметр – экземпляр делегата ThreadStart. Этот делегат представляет метод, который будет вызываться для выполнения операций созданного потока. Шаблон для создания потока выполнения выглядит следующим образом [10]:

```
public static void Method ( )
        {
           {
                // Тело метода
            }
        }
public static void Main ( )
{
     Thread t2 = new Thread (new ThreadStart (Method ( ) );
     . . .
// Пример создания потоков
```

```
using System;
using System.Threading;

namespace ConsoleApplication1
{
class Create_Thred
{
        // Метод Coundown считает от 1000 до 1
        public static void Coundown()
        {
                for (int counter = 1000; counter > 0; counter--)
                        Console.Write(counter.ToString() + " ");
        }
        static void Main()
        {
```

```
            // Создание второго потока
            Thread t2 = new Thread(new ThreadStart(Coundown));
            // Запуск потока
            t2.Start();
            //
            // В это же время метод Coundown вызывается в
               // основном     потоке
            Coundown();
            Console.ReadLine();
        }
}
}
```

В этой программе создается поток с именем t2 и вызывается метод Start. После того как метод Start вызван, начинается выполнение метода Countdown. В тоже время метод Countdown вызывается из основного потока. Основная идея этого примера – заставить два метода Countdown выполняться одновременно.

На рис. 2.33 приведен возможный результат выполнения этой программы. Если внимательно рассмотреть рисунок, то можно заметить, что информация, выводимая разными потоками, перемешана. Это происходит потому, что хотя процессор выполняет потоки раздельно, они используют одни и те же ресурсы – дисплей компьютера. Если выполнить программу несколько раз, можно получить различные результаты. На самом деле невозможно предсказать, когда процессор переключится с одного потока на другой. Это связано с тем, что потоки имеют один и тот же приоритет, т.е. равноправны для системы.

2 Вопрос

Приоритеты потоков

Один из основных принципов многопоточной архитектуры заключается в том, что не все потоки имеют одинаковые приоритеты. Несмотря на то, что все потоки создаются одинаковыми (с приоритетом Normal), не всегда целесообразно оставлять приоритеты потоков, установленные при их создании.

Например, если производится фоновая печать одновременно с редактированием документа, то ясно, что отдавать половину вычислительных ресурсов процессу печати невыгодно. Намного лучше отдавать большее количество ресурсов основному приложению – в этом случае оно будет быстрее реагировать на запросы пользователя, а печать все равно закончится, может быть несколько позже.

Чтобы выполнить подобное перераспределение ресурсов, используется свойство Priority. Это свойство имеет пять допустимых значений:

Lowest (низший),

BelowNormal (ниже нормального),
Normal (нормальный),
AboveNormal (выше нормального),
Highest (наивысший).

Нужно иметь в виду, что среда .NET изначально проектировалась в расчете на перенос на несколько платформ, поэтому перечисленные значения приоритетов могут не совпадать со значениями конкретной ОС. Например, в Windows 2000 существует 6 основных приоритетов, а в Windows CE 3.0 – 256 градаций приоритета, обозначаемых числами от 0 до 255.

Если в нижеследующей программе установить потокам одинаковый приоритет, то результат работы программы будет таким, как это показано на рис. 2.34 а). Если этого не делать, а оставить программу такой, как она приведена ниже, то поток более высокого приоритета завершается раньше (см. рис. 2.34 б)).

```
// Пример потоков с приоритетами
using System;
using System.Threading;
namespace ConsoleApplication2
{
class Prior_Thread
{
        // Метод Coundown считает от 1000 до 1
        public static void Coundown()
        {
                for (int counter = 1000; counter > 0; counter--)
                        Console.Write(counter.ToString() + " ");
        }
        static void Main()
        {
                // Создание второго потока
                Thread t2 = new Thread(new ThreadStart(Coundown));
                // Установка второму потоку высшего приоритета
                t2.Priority = ThreadPriority.Highest;
                // Установка текущему потоку низшего приоритета
                Thread.CurrentThread.Priority = ThreadPriority.Lowest;
                // Запуск второго потока
                t2.Start();
                // В то же время несколько измененный
                   // метод         Coundown
                // выполняется в первом потоке
                for (int counter = 1000; counter > 0; counter--)
                        Console.Write(counter.ToString() + " a ");
                Console.ReadLine();
        }
}
}
```

а)

б)

3 Вопрос

Состояния потоков

Объекты класса Thread за время своего существования могут пребывать в разных состояниях. Для получения текущего состояния потока используется свойство ThreadState. Оно возвращает одно из 10 возможных значений:

Unstarted – не запущен;

Running – выполняется;

Background – фоновый;

WaitSleepJoin – заблокирован в результате вызова методов Wait, Sleep или Join;

SuspendRequested – запрос на приостановку;

Suspended – приостановлен;

StopRequested – запрос на остановку;

Stopped – остановлен;

AbortRequested – запрос на отмену (был вызван метод Abort, но поток еще не получил исключение ThreadAbortException;

Aborted – отменен.

Обычно сразу после создания объект класса Thread находится в состоянии Unstarted. После вызова метода Thread.Start, он переходит в состояние Running. Если после этого свойству Is Background присвоить значение true, то поток перейдет в состояние Background и т д.

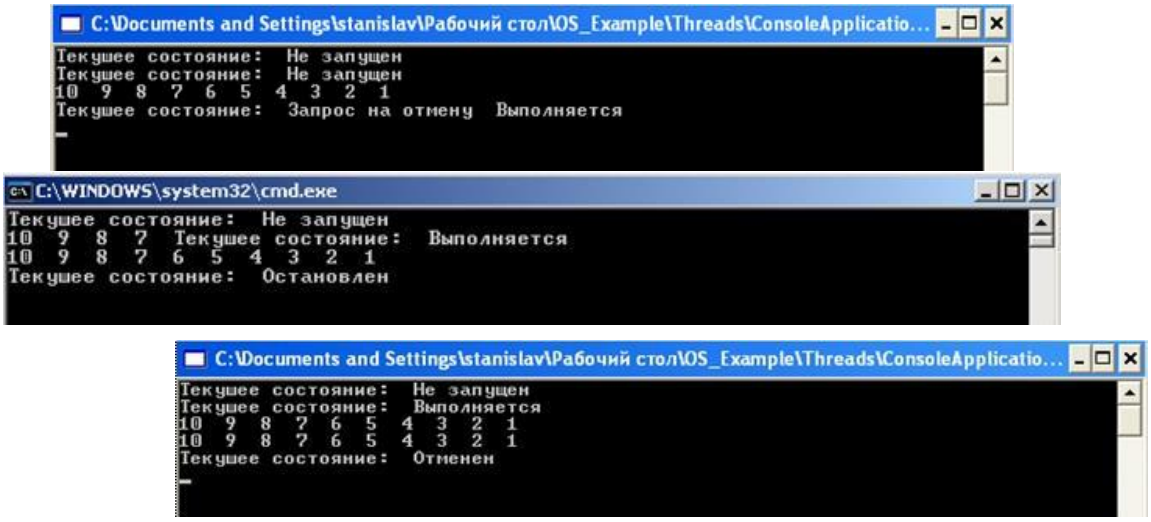
Поток может находиться в нескольких состояниях в один и тот же момент времени. Например, поток, ждущий освобождения ресурса, может быть одновременно в состоянии WaitSleepJoin, и, если в ходе ожидания был вызван метод Abort, в состоянии AbortRequested. Поэтому для проверки состояния потока часто приходится использовать логические конструкции. Такой подход показан в нижеследующей программе.

```
// Анализ состояния потока
using System;
using System.Threading;
namespace ConsoleApplication3
{
class StateThreads
{
        // Метод Countdown считает от 10 до 1
        public static void Countdown()
        {
                for (int counter = 10; counter > 0; counter--)
                        Console.Write(counter.ToString() + " ");
```

```
                Console.WriteLine();
        }
        // Метод DumpThreadState выводит текущее состояние потока
        // ThreadState – битовая маска состояния потока
public static void DumpThreadState (Thread t)
{
        Console.Write("Текушее состояние: ");
        if ((t.ThreadState & ThreadState.Aborted) == ThreadState.Aborted)
                Console.Write ("Отменен ");
        if ((t.ThreadState & ThreadState.AbortRequested) ==
                ThreadState.AbortRequested)
                Console.Write ("Запрос на отмену ");
        if ((t.ThreadState & ThreadState.Background) ==
                ThreadState.Background)
                Console.Write ("Выполняется в фоновом режиме ");
        if ((t.ThreadState & (ThreadState.Stopped | ThreadState.Unstarted |
                ThreadState.Aborted)) == 0)
                Console.Write ("Выполняется ");
        if ((t.ThreadState & ThreadState.Stopped) == ThreadState.Stopped)
                Console.Write ("Остановлен ");
        if ((t.ThreadState & ThreadState.StopRequested) ==
                ThreadState.StopRequested)
                Console.Write ("Запрос на остановку ");
        if ((t.ThreadState & ThreadState.Suspended) ==
                ThreadState.Suspended)
                Console.Write ("Приостановлен ");
        if ((t.ThreadState & ThreadState.SuspendRequested) ==
                ThreadState.SuspendRequested)
                Console.Write ("Запрос на приостановку ");
        if ((t.ThreadState & ThreadState.Unstarted) ==
                ThreadState.Unstarted)
                Console.Write ("Не запущен ");
        if ((t.ThreadState & ThreadState.WaitSleepJoin) ==
                 ThreadState.WaitSleepJoin)
                Console.Write ("Ожидает освобождения ресурсов ");
                Console.WriteLine();
}
        static void Main()
        {
                // Создание еще одного потока
                Thread t2 = new Thread(new ThreadStart(Countdown));
                DumpThreadState(t2);
                // Запуск нового потока
                t2.Start();
                DumpThreadState(t2);
                // В это же время метод Countdown вызывается в
                // основном     потоке
                Countdown();
                // Отмена второго потока
                t2.Abort();
                DumpThreadState(t2);
```

```
            Console.ReadLine();
        }
}
}
```

Эта программа может вывести следующие строки (рис. 2.35).



4 вопрос

Как было показано в предыдущих примерах, обычно порядок выполнения потоков не определен. Однако существуют способы управления потоками, позволяющие сделать их поведение более предсказуемым.

Класс Timer позволяет выполнять потоки через повторяющиеся промежутки времени. *Класс Join* позволяет одному потоку ждать завершения другого. *Классы Lock, Interlocked, Monitor и Mutex*, позволяют координировать действия нескольких потоков и использование ресурсов несколькими потоками.

При помощи класса Timer можно добавить в программу поток, работающий по принципу “вызвали и забыли”. При создании экземпляра класса Timer указывается четыре параметра:

Callback. Делегат типа TimerCallback, представляющий метод, который будет вызываться таймером.

State. Объект, который передается методу TimerCallback. Его можно использовать для того, чтобы таймеру было доступно некоторое постоянное состояние. Если такой необходимости нет, то этот параметр может быть равным null.

DueTime. Количество миллисекунд перед первым срабатыванием таймера.

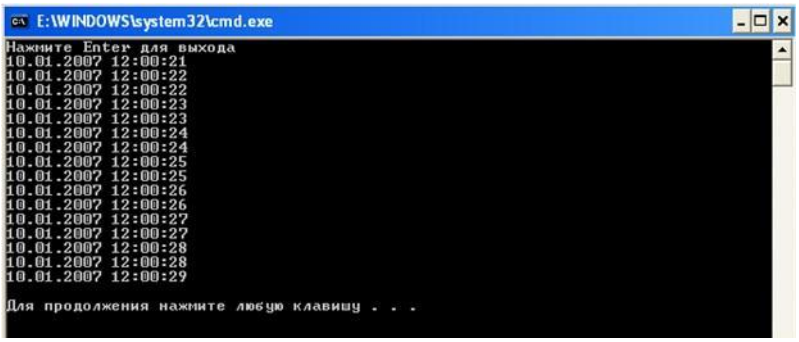
Period. Количество миллисекунд между срабатываниями таймера.

В следующем листинге показан пример использования класса Timer.

```
// Использование класса Timer
using System;
using System.Threading;

namespace ConsoleApplication4
{
class ManagementThreads
{
        // Метод CheckTime вызывается по таймеру
        public static void CheckTime(Object state)
```

```
        {
            Console.WriteLine(DateTime.Now);
        }
        static void Main()
        {
            // Создание делегата, который будет вызываться объектом Timer
            TimerCallback tc = new TimerCallback(CheckTime);
            // Создание таймера, срабатывающего дважды в секунду
            // Первый запуск произойдет через одну секунду
            Timer t = new Timer(tc, null, 1000, 500);
            // Ожидание ввода пользователя
            Console.WriteLine("Нажмите Enter для выхода");
            Console.Read();
            // Освобождение ресурсов
            t.Dispose();
            t = null;

        }
}
}
```

Эта программа может вывести строки



5 Вопрос

Метод, указанный в конструкторе объекта Timer, выполняется в отдельном потоке, созданном системой, а не в потоке, создавшем объект Timer. Обратите внимание на вызов метода Dispose объекта Timer. Это делается для того, чтобы корректно освободить уже ненужные ресурсы, выделенные отдельному потоку.

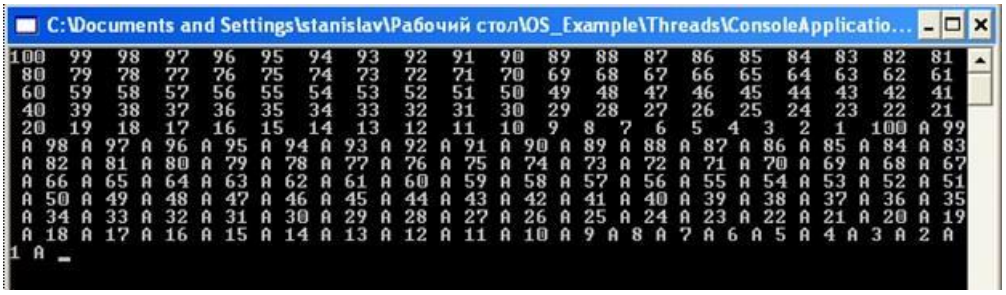
Метод Thread.Join позволяет “прикрепить” один поток к другому. Это означает, что первый, например поток, будет ждать завершения второго, после чего будет запущен. В следующем листинге показано использование этого метода.

```
// Использование метода Join
using System;
using System.Threading;

namespace ConsoleApplication5
{

class Waite_End_Thread
{
        // Метод Countdown считает от 100 до 1
```

```
        public static void Countdown()
        {
            for (int counter = 100; counter > 0; counter--)
                Console.Write(counter.ToString() + " ");
        }
        // Метод Countdown1 считает от 100 до 1 с буквой A
        public static void Countdown1()
        {
            for (int counter = 100; counter > 0; counter--)
                Console.Write(counter.ToString() + " A ");
        }

        static void Main()
        {
            // Создание второго потока
            Thread t2 = new Thread(new ThreadStart(Countdown));
            // Запуск второго потока
            t2.Start();
            // Блокировка первого потока до завершения второго
            t2.Join();
            // Вызов метода Countdown1 из первого потока
            Countdown1();
            Console.Read();
        }
}
}
```

Если выполнить этот пример (см. рис. 2.39), то можно заметить, что вывод двух методов Countdown не пересекается, несмотря на то, что оба потока выполняются с нормальным приоритетом. Это произошло потому, что вызов t2.Join приостанавливает основной поток до тех пор, пока не завершится выполнение метода Countdown во втором потоке.

Метод Join имеет перегруженную версию, которая позволяет указать максимальное время ожидания. Например, для того чтобы ждать завершения второго потока не дольше пяти секунд, можно использовать оператор t2.Join (5000).



Задание для самостоятельной работы

Базовый уровень

1. Модифицируйте рассмотренные в этом разделе программы, организовав в них три и более потоков.

2. Подумайте, как изменится время выполнения многопоточной программы, увеличивающей значение глобальной переменной, при увеличении количества потоков. Не

забывайте, что каждое переключение контекста требует определенных расходов системных ресурсов.

Повышенный уровень

3. Продумайте параллельный алгоритм решения какой-либо задачи и составьте по нему многопоточную программу.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации самостоятельной работы по дисциплине

«Операционные системы»

для студентов специальности 10.05.01 «Компьютерная безопасность» (специализация «Информационно-аналитическая и техническая экспертиза компьютерных систем»)

Ставрополь

2025

СОДЕРЖАНИЕ

Введение
1. Общая характеристика самостоятельной работы студента
2. Технологическая карта самостоятельной работы студента
3. Самостоятельное изучение тем лекций
4. Паспорт фонда оценочных средств для проверки самостоятельной работы
5. Вопросы для собеседования по самостоятельному изучению тем дисциплины
6. Список рекомендуемой литературы

Введение

Целью изучения дисциплины «Операционные системы» является получение теоретических знаний о принципах построения и архитектуре современных операционных систем, обеспечивающих организацию вычислительных процессов в корпоративных информационных системах экономического, управленческого, производственного, научного и другого назначения, а также практических навыков по созданию (настройке) вычислительной среды в корпоративных сетях (интрасетях) предприятий.

Задачи курса:

- формирование фундаментальных знаний обеспечивающих разработку и эксплуатацию программного обеспечения компьютерных сетей, автоматизированных систем, вычислительных комплексов;
- формирование практических навыков разработки и эксплуатации программного обеспечения компьютерных сетей.

1. Общая характеристика самостоятельной работы студента

Основными видами учебной работы по достижению результатов освоения дисциплины являются лекции, лабораторные работы, практические занятия и самостоятельная работа студентов (СРС).

На лекциях раскрываются основные положения и понятия курса, формируются знания в области теории сложности комбинаторных алгоритмов. На лабораторных работах и практических занятиях формируются умения и навыки, необходимые для решения задач профессиональной деятельности.

Приступая к изучению учебной дисциплины, необходимо ознакомиться с учебной программой, получить в библиотеке рекомендованные учебные пособия, а также получить у ведущего преподавателя в электронном виде конспекты лекций, методические рекомендации к лабораторным занятиям и тестовые задания, завести новую тетрадь для конспектирования лекций и выполнения практических заданий.

Для изучения дисциплины предлагается список основной и дополнительной литературы. Основная литература предназначена для обязательного изучения, дополнительная – поможет более глубоко освоить отдельные вопросы, подготовить исследовательские задания и выполнить задания для самостоятельной работы.

В ходе лекционных занятий студент обязан осуществлять конспектирование учебного материала, особое внимание, обращая на категории, формулировки, раскрывающие содержание тех или иных понятий, физических явлений, процессов, эффектов. В рабочих конспектах желательно оставлять поля, на которых следует делать пометки из рекомендованной литературы, дополнять материал прослушанной лекции, формулировать научные выводы и практические рекомендации. Студент имеет право задавать преподавателю уточняющие вопросы с целью уяснения теоретических положений, разрешения спорных ситуаций.

Самостоятельная работа студентов над материалом учебной дисциплины является неотъемлемой частью учебного процесса и должна предполагать углубление знания учебного материала, излагаемого на аудиторных занятиях, и приобретение дополнительных

знаний по отдельным вопросам самостоятельно.

Основными видами самостоятельной работы студентов по учебной дисциплине являются:

- самостоятельное изучение учебного материала по заданным темам;
- подготовка к лабораторным занятиям, оформление отчета по выполненному лабораторной работе и подготовка к собеседованию по результатам работы.

Основными методами самостоятельной работы студентов являются:

1) для овладения знаниями:

- чтение текста (конспекта лекций, учебника, дополнительной литературы) и конспектирование текста;
- работа с нормативными документами;
- поиск информации в Интернет;

2) для закрепления и систематизации знаний:

- работа с конспектом лекции (обработка текста);
- повторная работа над учебным материалом (учебника, дополнительной литературы, слайдами);
- составление плана и тезисов ответа или графическое изображение структуры ответа;
- составление таблиц для систематизации учебного материала;
- изучение нормативных документов;
- ответы на контрольные вопросы;
- тестирование в программе компьютерного тестирования «Iren»;

3) для формирования умений:

- подготовка к лабораторным занятиям;
- решение задач и практических заданий;
- подготовка отчетов по лабораторным занятиям;
- рефлексивный анализ профессиональных умений.

В случае пропуска учебного занятия студент может воспользоваться содержанием различных блоков учебно-методического комплекса (лекции, практические занятия, контрольные вопросы и тесты) для самоподготовки и освоения темы. Для самоконтроля необходимо использовать вопросы и задания, предлагаемые к лабораторным занятиям.

2. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
4 семестр					
ИД-1 ОПК-9 ИД-1 ОПК-12	Подготовка к практическому занятию	Собеседование, проверка конспектов	20	28	48
Итого за 4 семестр			20	28	48
Итого			20	28	48

3. Самостоятельное изучение тем лекций

№ п/п	Темы для самостоятельного изучения	Рекомендуемые источники информации (№ источника)			
		Основная	Дополни-тельная	Методи-ческая	Интернет-ресурсы
1	Тема: Мультипрограммные вычислительные процессы. Поиск информации в сети Интернет по вопросам организации вычислительного процесса в кластерных архитектурах.	1-4	1-3	1	
2	Тема: Технологии организации ввода-вывода данных в современных компьютерах. Поиск информации в сети Интернет по контроллерам прямого доступа к памяти и контроллерам прерывания. Поиск и загрузка свободно распространяемых утилит управления устройствами	1-4	1-3	1	
3	Тема: Поиск информации в сети Интернет по контроллерам прямого доступа к памяти и контроллерам прерывания. Поиск и загрузка свободно распространяемых утилит управления устройствами.	1-4	1-4	1	
4	Тема: Проведение экспериментов с файловой системой: просмотр альтернативных потоков, работа с шифрующей файловой системой.	1-4	1-4	1	
5	Тема: Решение задач по определению размера адресной информации в различных файловых системах.	1-4	1-4	1	

4. Паспорт фонда оценочных средств для проверки самостоятельной работы

Этап формирования компетенции (№ темы)	Тип контроля	Вид контроля	Количество элементов, шт.	
			Базовы й	Повыше нный

1-3	текущий	Вопросы для собеседования по самостоятельному изучению тем дисциплины	6	6
2-5	текущий	Вопросы для собеседования по самостоятельному изучению тем дисциплины	5	2

5. Вопросы для собеседования по самостоятельному изучению тем дисциплины

Тема 1. Мультипрограммные вычислительные процессы. Поиск информации в сети Интернет по вопросам организации вычислительного процесса в кластерных архитектурах

Базовый уровень:

1. Что такое мультипрограммные вычислительные процессы?
2. Что такое кластер?
3. Объясните понятие кластерные архитектуры.

Повышенный уровень:

4. Как организовать вычислительный процесс в кластерных архитектурах?

Тема 2. Технологии организации ввода-вывода данных в современных компьютерах. Поиск информации в сети Интернет по контроллерам прямого доступа к памяти и контроллерам прерывания. Поиск и загрузка свободно распространяемых утилит управления устройствами

Базовый уровень:

1. Какие существуют технологии организации ввода-вывода данных в современных компьютерах?
2. Что такое контроллер прямого доступа к памяти?
3. Что такое контроллер прерывания?

Повышенный уровень:

4. Изучите свободно распространяемые утилиты управления устройствами.

Тема 3. Поиск информации в сети Интернет по контроллерам прямого доступа к памяти и контроллерам прерывания. Поиск и загрузка свободно распространяемых утилит управления устройствами

Базовый уровень:

1. Принцип действия контроллера прямого доступа к памяти.
2. Принцип действия контроллера прерывания.

Тема 4. Проведение экспериментов с файловой системой: просмотр альтернативных потоков, работа с шифрующей файловой системой

Базовый уровень:

1. Что такое альтернативные потоки?
2. Как просмотреть альтернативные потоки?

Повышенный уровень:

3. Какие шифрующие файловые системы вы знаете?
4. На чем основаны их принципы действия?

Тема 5. Решение задач по определению размера адресной информации в различных файловых системах

Базовый уровень:

1. Как определяется размер адресной информации в различных файловых системах?

Повышенный уровень:

2. Продемонстрируете определение размера адресной информации.

Критерии оценки для проверки уровня обученности:

Оценка «отлично» выставляется студенту, если студент показал глубокое, прочное и аргументированное освоение программного учебного материала, при этом поставленный вопрос раскрыт последовательно, четко и логически стройно, в полном исчерпывающем объеме, основные категории, понятия и термины учебного курса формулировались правильно, не допущено при ответе ошибок.

Оценка «хорошо» выставляется студенту, если студент показал твердое знание программного учебного материала, при этом поставленный вопрос раскрыт грамотно и по существу, в достаточно полном объеме, основные категории, понятия и термины учебного курса формулировались правильно, допущены при ответе отдельные неточности или одна ошибка.

Оценка «удовлетворительно» выставляется студенту, если студент показал знание только основной части учебного материала без его частных деталей, при этом поставленный вопрос раскрыт с нарушением логической последовательности, не в полном объеме; были допущены неточные формулировки основных категорий, понятий и терминов учебного курса, а также ошибки (не более двух) или ряд незначительных неточностей, не исказивших существенно суть ответа.

Оценка «неудовлетворительно» выставляется студенту, который не знает значительной части программного материала, допускает существенные ошибки (более двух), существенно исказившие его суть. Оценка неудовлетворительно выставляется также, если отсутствует ответ на вопрос, либо студент отказался его сдавать.

6. Список рекомендуемой литературы Основная литература:

1. Дейтел П.Дж. Операционные системы. Основы и принципы. Книга 1. – Бином, 2013. – 1024 с.

2. Таненбаум Э. Современные операционные системы. – СПб.: Питер, 2013. – 1120 с.

3. Назаров С.В., Широков А.И. Современные операционные системы. Учеб. пособие. – М.: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2010.

4. Назаров С.В. Операционные среды, системы и оболочки. Учеб. пособие. – М.: КУДИЦ-ПРЕСС, 2007.

5. Назаров С.В., Гудыно Л.П., Кириченко А.А. Операционные системы. Практикум. Учеб. пособие. – М.: КНОРУС, 2011

Дополнительная литература:

1. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. СПб.: Питер, 2005.

2. Столингс В. Операционные системы. М.: Вильямс, 2004.

3. Мюллер Дж., Чоудри П. Microsoft Windows 2000. Настройка и оптимизация производительности. М.: ЭКОМ, 2001.

4. Руссинович М., Соломон Д. Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP и Windows 2000. Мастер-класс. / Пер. с англ. – 4-е изд. – М.: Издательство «Русская редакция»; Спб.: Питер, 2006.

Методическая литература:

1. Методические пособия по дисциплине

Интернет-ресурсы:

1. http://www.anti-virus.by/

2. http://www.linuxrsp.ru/
3. http://www.interface.ru/
4. http://www.computerinform.ru/
5. http://www.windows.ru/

Программное обеспечение:

1. Операционные системы UNIX, Windows, LINUX, IBM

Материально-техническое обеспечение дисциплины

1. Мультимедийная лекционная аудитория № 431.
2. Аудитории, оснащённые компьютерами с установленным требуемым программным обеспечением, для проведения лабораторных работ № 433.