Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки

Кафедра «Информатика»

Д. А. Литвинов

АРХИТЕКТУРА ОПЕРАЦИОННЫХ СИСТЕМ

ПРАКТИКУМ
по выполнению лабораторных работ
для слушателей специальности переподготовки
1-40 01 73 «Программное обеспечение информационных систем»
заочной формы обучения

Гомель 2023

УДК 004.65(075.8)
ББК 32.973-018.2я73
Л64

Рекомендовано кафедрой «Информатика» ГГТУ им. П. О. Сухого
(*протокол № 2 от 27.09.2023 г.*)

Рецензент: доц. каф. «Промышленная электроника» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *А. В. Ковалев*

Литвинов, Д. А.

Л64 Архитектура операционных систем : практикум по выполнению лаборатор. работ для слушателей специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / Д. А. Литвинов. – Гомель : ГГТУ им. П. О. Сухого, 2023. – 75 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://elib.gstu.by. – Загл. с титул. экрана.

Практикум по выполнению лабораторных работ посвящен изучению программных средств создания сценариев для операционной системе Windows. Предложенные практические задания позволят изучить язык командных файлов, сервер сценариев Windows Script Host, алгоритмы планирования процессов в операционных системах.

Для слушателей специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем» ИПКиП.

УДК 004.65(075.8)
ББК 32.973-018.2я73

Содержание

Лабораторная работа № 1
Командный интерфейс ОС Windows

Цель работы: изучить интерфейс и язык командного интерпретатора ОС Windows.

1.1 Интерфейс командной строки Windows. Интерпретатор Cmd.exe

В операционной системе Windows, как и в других операционных системах, 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). Командный интерпретатор или оболочка командной строки – это программа, которая, находясь в оперативной памяти, считывает набираемые вами команды и обрабатывает их.

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.

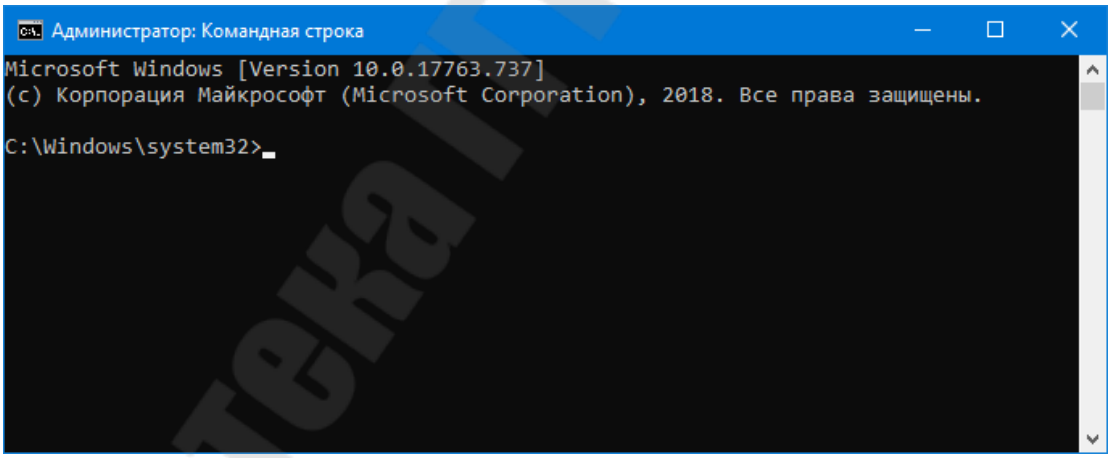


Рисунок 1.1 – Вид окна командного интерпретатора

1.2 Синтаксис командной строки, перенаправление ввода – вывода

Файловая система имеет древовидную структуру и имена файлов задаются в формате *[диск:] [путь\]имя_файла*. Если путь начинается с символа «\», то маршрут вычисляется от корневого каталога – иначе от текущего.

Например, **c:123.txt** задает файл **123.txt** в текущем каталоге, **c:\123.txt** – в корневом, а **DOC\123.txt** – в подкаталоге DOC текущего каталога.

Существуют особые обозначения для текущего каталога (точка «**.**») и трех его верхних уровней (две точки «**..**» - родительский, три «**…**» - второго уровня и, наконец, четыре «**….**» - третьего уровня).

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.

Например, **DIR /? > helpdir.txt** выведет справку по команде DIR в файл. Символ «>>» позволяет не создавать файл заново, а дописать в него. По аналогии символ «<» позволяет читать данные не с клавиатуры, а с файла. Например**, DATE < date.txt** ввод новой даты из файла.

1.3 Переменные окружения

При загрузке ОС Windows в оперативной памяти постоянно хранится набор т.н. *переменных окружения* (environment variables). Хотя в Windows есть более совершенный способ для хранения системных значений – реестр, многие программы по-прежнему используют переменные окружения. Наиболее важные переменные хранят системный путь для поиска (PATH), каталог запуска Windows (WINDIR), место хранения временных файлов (TEMP) и многое другое.

Переменные устанавливаются с помощью команды:

SET [переменная=[строка]]

Запуск SET без параметров приводит к выводу списка переменных среды. Для получения их значений (всегда строки) нужно имя соответствующей переменной заключить в символы «%», **например**: **%TEMP%.**

1.4 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 операционной системы 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, вы после приглашения командной строки (например, C:\) вводите имя этой команды (регистр не важен), ее параметры и ключи (если они необходимы) и нажимаете 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: /?.

1.5 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 амперсанта &, 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.

Символ ^ позволяет использовать командные символы как текст, то есть при этом происходит игнорирование значения специальных символов. Например, если ввести в командной строке – **ECHO Абв & COPY /?** и нажать клавишу <Enter>, то произойдет выполнение подряд двух команд: **ECHO Абв и COPY /?** (команда **ECHO** выводит на экран символы, указанные в командной строке после нее). Если же выполнить команду **ECHO Абв ^& COPY /?** то на экран будет выведено **– Абв & COPY /?.** В этом случае просто выполняется одна команда **ECHO** с соответствующими параметрами.

Условная обработка команд в Windows осуществляется с помощью символов && и || следующим образом. Двойной амперсант && запускает команду, стоящую за ним в командной строке, только в том случае, если команда, стоящая перед амперсантами 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 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**.

1.6. Основные команды

Рассмотрим некоторые наиболее часто используемые команды для работы с файловой системой. Полный список команд можно вывести, набрав **HELP** в командной строке. Для вызова справки по командам можно использовать запись: **HELP имя команды**

1.6.1 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.

1.6.2 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 таблице 1.1.

Таблица 1.1 – 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

Приведем примеры использования команды **COPY**.

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

COPY abc.txt D:\PROGRAM

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

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

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 D: в текущий каталог, достаточно выполнить такую краткую команду:

COPY D:*.*

В качестве источника или результата при копировании можно указывать имена не только файлов, но и устройств компьютера. Например, для того, чтобы распечатать файл abc.txt на принтере, можно воспользоваться командой копирования этого файла на устройство PRN:

COPY abc.txt PRN

Другой интересный пример: создадим новый текстовый файл и запишем в него информацию, без использования текстового редактора. Для этого достаточно ввести команду **COPY CON my.txt**, которая будет копировать то, что вы набираете 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) нескольких файлов в один. Для этого необходимо указать единственный результирующий файл и несколько исходных. Это достигается путем ис-

пользования 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.

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.

1.6.3 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 этой команды:

XCOPY источник [результат] [ключи]

Команда **XCOPY** имеет множество ключей, мы коснемся 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 нахо-

дились 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.

1.6.4 Команда DIR

Еще одной очень полезной командой является **DIR [диск:][путь][имя_файла] [ключи]**, которая 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 ..
TEMPLT02 DAT         227  07.08.98   1:00 templt02.dat
UNINST1  000       1 093  02.03.99   8:36 UNINST1.000
HILITE   DAT       1 082  18.09.98  18:55 hilite.dat
TEMPLT01 DAT          48  07.08.98   1:00 templt01.dat
```

```
UNINST0  000       40 960  15.04.98   2:08 UNINST0.000
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, то есть 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

1.6.5 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.

1.6.6 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**).

1.6.7 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.

1.6.8 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.

1.7 Контрольное задание 1

За выполнения задания 1 балл.

1. Запустить командный процессор: **cmd.exe**
2. Создать на диске E: папку: **mkdir test или md test**
3. Войти в созданную папку: **cd test**
4. В паке создать текстовый файл с информацией о студенте *(ФИО, дата рождения, группа):* **Copy con my.txt .Примечание**: Для окончания ввода нажать CTRL+Z и затем ENTER.

Вывести содержимое паки на экран: **dir**

5. Вывести содержимое созданного файла на экран: **type my.txt**
6. В созданной папке создать две папки с именами A1 и A2. **Примечание**: Для повтора предыдущей команды удобно использовать кнопки – «вверх», «вниз», «вправо».
7. Скопировать *my.txt* в папку A1: **copy my.txt A1**.
8. В папке A2 создать файл с деревом каталога *test*: **tree . >A2\tree.txt**
9. Переименовать файл в папке A1 в *old.txt*: **ren a1\ my.txt old.txt**
10. С помощью команды *move* поменять содержимое папок A1 и A2.

Удалить все папки и файлы в исходной папке *test*: **rd /s**

1.8 Контрольное задание 2

Задание выполняется в командном процессоре. Все команды привести в отчете по пунктам. За выполнения задания 2 балла.

1. Создать папку **Zadanie1**, а в ней три папки A1, A2, A3.
2. В папку A1 скопировать все файлы с расширением **ini** из каталога **c:\windows**.
3. В папке A2 создать файл с именами файлов в паке **c:\windows**.
4. скопировать содержимое папок A1 и A2 в A3 и вывести ее содержимое на экран. При выполнении задания использовать группировку команд (**&**).
5. Заменить расширение у всех файлов из папки A1 с **ini** на **bak** и перед именем каждого файла поставить символ **~.**
6. Сохранить значения системных переменных окружения в файле с именем **set**.**txt** в папке A3.

11. Создать свою системную переменную **myname** с фамилией студента. Сохранить значения системных переменных окружения в файле с именем **new_set.txt** в папке A3. Файл привести в отчете.
7. Показать созданную структуру преподавателю. Удалить папки A1, A2, A3.

1.9 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 операционной системы (как внешние, так и внутренние), а также некоторые дополнительные инструкции и ключевые слова, придающие командным файлам некоторое сходство с алгоритмическими языками программирования.

1.9.1 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 …
DIR C:\
ECHO OFF
REM 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
REM Команда DIR C:\ дублируется на экране
DIR C:\
REM А команда DIR D:\ — нет
@DIR D:\
```

Таким образом, если поставить в самое начало файла команду – **@ECHO OFF** , 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!
```

Для просмотра сообщений, выводимых из пакетного файла, удобно 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
REM Добавление сообщения в файл report.txt в случае
REM удачного завершения копирования
IF NOT ERRORLEVEL 1 ECHO Успешное копирование >> report.txt
```

1.9.2 Использование параметров командной строки

При запуске пакетных файлов в командной строке можно указывать произвольное число параметров, значения которых можно ис-

пользовать 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
```

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.

Рассмотрим пример. 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 таблице 2.1.

Таблица 1.2 – Синтаксические операторы

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
%~Sn	Значение операторов N и X для переменной **%n** изменяется так, что они работают с кратким именем файла
%~$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.

Рассмотрим следующий пример. Пусть мы находимся в каталоге **C:\TEXT** и запускаем пакетный файл с параметром **Рассказ.doc (%1=Рассказ.doc**). Тогда применение операторов, описанных в таблице 1.2, 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
```

1.9.3 Работа с переменными среды

Внутри командных файлов можно работать с так называемыми 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, например, WINDIR, которая определяет расположение каталога Windows, TEMP, которая определяет путь к каталогу для хранения временных файлов Windows или **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.

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

```
@ECHO OFF
CLS
REM Создание переменной MyVar
SET MyVar=Привет
REM Изменение переменной + !!!!!!
SET MyVar=%MyVar%!!!!!!
ECHO Значение переменной MyVar: %MyVar%
```

```
REM 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
```

Значение некоторых переменных по команде SET не выдаются. В основном, это переменные, принимаемые значения которых динамически изменяются:

%CD% - Принимает значение строки текущего каталога.
%DATE% - Принимает значение текущей даты.
%TIME% - Принимает значение текущего времени.
%RANDOM% - Принимает значение случайного десятичного числа в диапазоне 1 -32767.

Для просмотра действующего значения какой-либо переменной обычно используется команда:

```
ECHO %переменная%
ECHO %CD% - отобразить имя текущего каталога
ECHO %TIME% - отобразить текущее время.
```

1.9.4 Преобразования переменных как строк

С переменными среды в командных файлах можно производить некоторые манипуляции. Во-первых, над ними можно производить операцию конкатенации (склеивания). 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
```

Для диалога с пользователем можно использовать команду:

```
SET /P имя переменной = текст
```

При выполнении которой, на экран выдается текстовое сообщение «**текст**» и ожидается ввод ответа. Пример - выполним запрос пароля и присвоим его значение переменной "**pset**":

```
set /p pset="Enter password - "
echo Password is - %pset%
```

Недостатком данного способа является невозможность продолжения выполнения командного файла при отсутствии ответа пользователя.

1.9.5 Операции с переменными как с числами

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

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

1.9.6 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.

1.9.7 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 флеш в USB пор: и нажмите любую клавишу
PAUSE
```

1.9.8 Операторы перехода

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. Приведем пример.

Пусть имеется командный файл следующего содержания:

```
@ECHO OFF
COPY %1 %2
GOTO Label1
ECHO Эта строка никогда не выполнится
:Label1
```

REM Продолжение выполнения
DIR %2
После того, как в этом файле мы доходим до команды
GOTO Label1
его выполнение продолжается со строки
REM 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).
Также 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
GOTO :EOF REM для предотвращения повторного выполнения блока :2
:2
ECHO %1

На экран выведутся три строки:

Копия-1
Копия-2

Использование команды **CALL** очень похоже на обычный вызов подпрограмм (процедур) в алгоритмических языках программирования. Если требуется разместить в файле несколько **CALL** вызовов, то необходимо в конце каждого блока соответствующего своей метке добавить команду перехода на конец файла **GOTO :EOF.**

@ECHO OFF

```
ECHO %1
CALL :1 Func1
CALL :2 Func2
GOTO :EOF REM для предотвращения выполнения блока меток
:1
ECHO %1
GOTO :EOF REM для предотвращения выполнения блока меток :2
:2
ECHO %1
```

Если запустить такой командный файл с параметром **"Main",** получим:

```
Main
Func1
Func2
```

В качестве параметра команды **CALL** можно использовать имя другого командного файла. Пример вызова командного файла 1.bat.

```
@ECHO OFF
ECHO Вызов 1.bat
CALL 1.bat
ECHO Возврат.
```

1.9.9 Операторы условия

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**.

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) 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.

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==Петя ECHO Привет, Петя!
```

При сравнении строк, заданных переменными, следует проявлять определенную осторожность. 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...
REM Начало цикла
:BegLoop
IF -%1==- GOTO ExitLoop
ECHO %1
REM Сдвиг параметров
```

```
SHIFT
REM Переход на начало цикла
GOTO BegLoop
:ExitLoop

REM Выход из цикла
ECHO Все.
```

Если запустить **primer.bat** с четырьмя параметрами:

```
primer.bat А Б В Г
```

В результате выполнения на экран выведется следующая информация:

```
Выполняется файл: primer.bat
Файл запущен со следующими параметрами:
А
Б
В
Г
Все.
```

Рассмотрим теперь оператор **IF** в следующем виде:

```
IF [/I] строка1 оператор_сравнения строка2 команда
```

Синтаксис и значение **операторов_сравнения** представлены в таблице 1.3.

Таблица 1.3 – Операторы сравнения в IF

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

Приведем пример 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.

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

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

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

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%
```

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.

Составим, например, командный файл, который бы копировал файл my.txt на диск C: без вывода на экран сообщений о копировании, а в случае возникновения какой-либо ошибки выдавал предупреждение:

```
@ECHO OFF
XCOPY my.txt C:\ > NUL
REM 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 таблице 1.3. Например:

```
IF ERRORLEVEL LEQ 1 GOTO ErrOccurred
```

1.9.10 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.

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

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

Пример.

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

В результате его выполнения на экране будет напечатано следующее:

```
Раз
Два
Три
```

Параметр **множество** в команде **FOR** задает одну или более текстовых строк, разделенных запятыми, которые вы хотите 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. 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, для вывода 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.

Следующий вариант команды **FOR** реализуется с помощью ключа **/D**:

```
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. Выполнив командный файл:

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

Получим список всех каталогов на диске C:, например:

```
C:\Program Files
C:\Program Files (x86)
C:\Temp
C:\Users
C:\Windows
```

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**. Например, для распечатки всех файлов с рас-

ширением 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\.
```

Ключ **/L** позволяет реализовать с помощью команды **FOR** арифметический цикл, 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) раскрывается в последовательность чисел с заданными началом, концом и шагом приращения. Так, набор (1,1,5) раскрывается в (1 2 3 4 5), а набор (5,-1,1) заменяется на (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.

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
```

Самые мощные возможности (и одновременно самый сложный синтаксис) 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 таблице 1.4 ключевые слова:

Таблица 1.4 – 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-N	Определение номеров подстрок, выделяемых из каждой строки 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=1,3* 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=;**);
- первая 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 первой подстроки, **%%j** – для третьей, а **%%k** получает все оставшиеся подстроки после третьей.

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 bbb vvv,fffff tttt
```

wwwww,yyyy uuuu
;kkk llll mmmmm
eee ttt jjj qqqq

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

aaa vvv fffff tttt
wwwww uuuu
eee jjj qqqq

Для такого варианта цикла, выбираются слова 1,2 и 3, которые присваиваются параметрам цикла **%%i %%j %%k,** соответственно:

FOR /F "EOL=; TOKENS=1-3* DELIMS=, " %%i IN (myfile.txt) DO @ECHO %%i %%j %%k

Результат:

aaa bbb vvv
wwwww yyyy uuuu
eee ttt jjj

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
 ("AAA ББББ ВВВВ,ГГГГГ ДДДД") DO @ECHO %%i %%j %%k
```

при своем выполнении напечатает

ББББ ВВВВ ГГГГГ ДДДД

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

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

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 1.5).

Таблица 1.5 – 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

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
```

На экран выведутся полные имена всех файлов с расширением txt:

```
C:\Program Files\Far\Contacts.txt
C:\Program Files\Far\FarFAQ.txt
C:\Program Files\Far\Far_Site.txt
C:\Program Files\Far\License.txt
C:\Program Files\Far\License.xUSSR.txt
C:\Program Files\Far\ReadMe.txt
C:\Program Files\Far\register.txt
C:\Program Files\Far\WhatsNew.txt
```

Если была бы использована простая конструкция **%%i**, были бы выведены только имена фалов.

1.9.11 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" (**ожидалось–**"a=РазДваТри")**, 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 (Раз,Два,Три) DO SET a=!a!%%i
ECHO a=%a%
```

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

1.10 Контрольные задания

В каждом задании разработать пакетный файл. **Все параметры передаются через командную строку или вводятся через диалог с пользователем. Выполнить обязательную проверку на наличие всех требуемых параметров.** Командный файл и результаты его работы привести в отчете. *Обязательна демонстрация работы пакетного файла преподавателю.*

Количество и сложность заданий (таблица 1.6) студент определяет самостоятельно. Максимальный возможный бал по работе – 10.

Таблица 1.6 – Варианты заданий

Вариант	Задание	Сложность, балл
1	В заданном месте создать, папку с именем зарегистрированного пользователя. В файл с именем «Log In. log», сохранить текущее время и дату	2
2	Выполнить резервное копирование файлов заданного расширения из указанной папки (с вложенными) в указанное место. Папки и расширение задать как параметры. Выполнить проверку на наличие заданных параметров, отключить вывод на экран списка копируемых файлов	3
3	Удалить на заданном диске (включая вложенные папки) все файлы с заданным расширением (например bak, tmp) вне зависимости от его атрибутов. Диск и расширение задать как параметры. Выполнить проверку на наличие заданных параметров	3
4	Скопировать все файлы с заданным расширением в заданную папку без сохранения структуры каталогов (четные номера компьютеров), с сохранением структуры (нечет-	2

	ные номера компьютеров). Папки и расширение задать как параметры	
5	Сохранить местоположение всех файлов с заданным расширением в файле с указанным именем. Расширение и имя файла задать как параметры. Заархивировать полученный список файлов любым архиватором	3
6	Составить пакетный файл для копирования заданного файла в соответствии со списком папок хранящихся в другом файле. Имена файлов задать как параметры	3
7	Создать на заданном диске папку с именем пользователя вошедшего в систему и сделать ее активной. Диск и имя файла задать как параметры	2
8	Создать в указанном месте структуру папок, хранящуюся в заданном файле	3
9	Выполнить резервное копирование с последующей архивацией файлов заданного расширения, находящихся в заданной и вложенных папках	3
10	Разработать пакетный файл для построения системы студенческих каталогов. Исходными данными являются имя группы и число пользователей в группе	3
11	Разработать пакетный файл для построения системы студенческих каталогов. Исходными данными являются имя группы и файл с именами пользовательских папок	4
12	Разработать пакетный файл для копирования только новых и обновленных файлов заданного расширения, из одного каталога в другой (включая подкаталоги)	2
13	Разработать пакетный файл для копирования файлов заданного расширения, изменённых в указанный день или после, из одного каталога в другой (включая подкаталоги)	2
14	Разработать командный файл для добавления/удаления пользователя в систему. Ис-	2

	ходными данными являются: имя пользователя, пароль добавить/удалить. Дополнительно можно задавать группу для пользователя (+1 балл)	
15	Разработать командный файл для добавления/удаления пользователей в систему согласно списку. Исходными данными является файл, в котором хранится имя пользователя, пароль, группа	4

Пример выполнения задания

Выполнить резервное копирование содержимого заданной папки (с вложенными) в указанное место. Папки задать как параметры. Выполнить проверку на наличие заданных параметров.

```
@ECHO OFF
CLS
ECHO Файл %0 копирует каталог %1 в %2
IF -%1==- GOTO NoParam
IF -%2==- GOTO NoParam
XCOPY %1\ %2 /S
GOTO :eof
:NoParam
ECHO Не заданы необходимые параметры командной строки!
PAUSE
```

Лабораторная работа № 2
Сервер сценариев Windows Script Host

Цель работы: изучить объектную модель Windows Script Host. Ознакомиться с технологией создания WSH-сценариев.

2.1 Сценарии Windows Script Host

Windows Script Host (WSH) – это инструмент, предназначенный для создания сценариев работающих в ОС Windows и поддерживающих технологию **ActiveX Scripting**. В качестве стандартных языков поддерживаются **Visual Basic Script Edition** (**VBScript**) и **JScript**. WSH является удобным инструментом для автоматизации повседневных задач пользователей и администраторов операционной системы Windows. Используя WSH, можно работать с файловой системой компьютера, а также управлять работой других приложений. WSH - сценарий, представляют собой текстовые файлы с расширением **js** или **vbs**. Сценарий можно выполнить в двух режимах консольном и графическом. Например:

cscript C:\MyScript.vbs – консольный режим
wscript C:\MyScript.vbs – графический режим

Если сценарий запускается в графическом режиме, то его свойства можно устанавливать с помощью вкладки **Сценарий** (**Script**) диалогового окна, задающего свойства файла в Windows.

2.2 Собственная объектная модель WSH

Объектная модель WSH версии 5.6 состоит из следующих объектов:

1. **WScript** - главный объект, который служит для создания других объектов или связи с ними, содержит сведения о сервере сценариев, а также позволяет вводить данные с клавиатуры и выводить информацию на экран или в окно Windows.
2. **WshArguments** - обеспечивает доступ ко всем параметрам командной строки запущенного сценария или ярлыка Windows.
3. **WshNamed** - обеспечивает доступ к именным параметрам командной строки запущенного сценария.
4. **WshUnnamed** - Обеспечивает доступ к безымянным параметрам командной строки запущенного сценария.

5. **WshShell** - Позволяет запускать независимые процессы, создавать ярлыки, работать с переменными среды, системным реестром и специальными папками Windows.
6. **WshSpecialFolders** - Обеспечивает доступ к специальным папкам Windows.
7. **WshShortcut** - Позволяет работать с ярлыками Windows.
8. **WshUrlShortcut** - Предназначен для работы с ярлыками сетевых ресурсов.
9. **WshEnvironment** - Предназначен для просмотра, изменения и удаления переменных среды.
10. **WshNetwork** - Используется при работе с локальной сетью: содержит сетевую информацию для локального компьютера, позволяет подключать сетевые диски и принтеры.
11. **WshScriptExec** - Позволяет запускать консольные приложения в качестве дочерних процессов, обеспечивает контроль состояния этих приложений и доступ к их стандартным входным и выходным потокам.
12. **WshController** - Позволяет запускать сценарии на удаленных машинах.
13. **WshRemote** - Позволяет управлять сценарием, запущенным на удаленной машине.
14. **WshRemoteError** - Используется для получения информации об ошибке, возникшей в результате выполнения сценария, запущенного на удаленной машине.
15. **FileSystemObject** - обеспечивающий доступ к файловой системе компьютера.

2.3 Объект WScript

В сценарии WSH объект **WScript** можно использовать, без предварительного описания или создания, так как его экземпляр создается автоматически.

Таблица 2.1 – Основные свойства объекта **WScript**

Свойство	Описание
Arguments	Содержит указатель на коллекцию WshArguments, содержащую параметры командной строки для исполняемого сценария
ScriptFullName	Содержит полный путь к запущенному сценарию

ScriptName	Содержит имя запущенного сценария
StdErr	Позволяет записывать сообщения в стандартный поток для ошибок
StdIn	Позволяет читать информацию из стандартного входного потока
StdOut	Позволяет записывать информацию в стандартный выходной поток

Таблица 2.2 – Основные методы объекта **WScript**

Метод	Описание
CreateObject (strProgID [, strPrefix])	Создает объект, заданный параметром strProgID
Echo([Arg1] [, Arg2] [,…])	Выводит текстовую информацию на консоль или в диалоговое окно
Quit([intErrorCode])	Прерывает выполнение сценария с заданным параметром intErrorCode кодом выхода. Если параметр intErrorCode не задан, то объект WScript установит код выхода равным нулю
Sleep(intTime)	Приостанавливает выполнения сценария на заданное число миллисекунд

Для использования же всех остальных объектов нужно использовать метод **CreateObject**. Основные свойства и методы объекта **WScript** представлены в таблицах 2.1 и 2.2 соответственно.

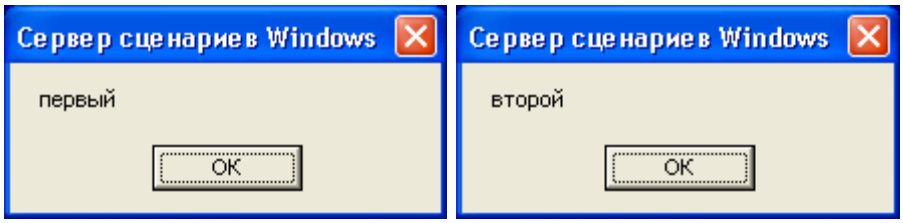
2.3.1 Свойство Arguments

Для перебора коллекции параметров командной строки, можно воспользоваться циклом For Each … Next.

```
Dim i,objArgs,Arg
Set objArgs = WScript.Arguments ' Создаем объект WshArguments
For Each Arg In objArgs
  WScript.Echo Arg ' Вывод значения аргумента
Next
```

Пример использования:

```
wscript 01.vbs первый второй
```



2.3.2 Свойства StdErr, StdIn, StdOut

Доступ к стандартным входным и выходным потокам можно получить только в случаи, если сценарий запущен в консольном режиме.

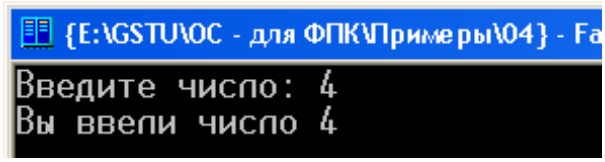
Таблица 2.3 – Основные методы для работы с потоками

Метод	Описание
Read(n)	Считывает из потока StdIn заданное параметром n число символов и возвращает полученную строку
ReadAll()	Читает символы из потока StdIn до тех пор, пока не встретится символ конца файла (<Ctrl>+<Z>)
ReadLine()	Возвращает строку, считанную из потока StdIn
Skip(n)	Пропускает при чтении из потока StdIn заданное параметром n число символов
SkipLine()	Пропускает целую строку при чтении из потока StdIn
Write(string)	Записывает в поток StdOut или StdErr строку string (без символа конца строки)
WriteBlankLines(n)	Записывает в поток StdOut или StdErr заданное параметром n число пустых строк
WriteLine(string)	Записывает в поток StdOut или StdErr строку string (вместе с символом конца строки)

Пример

```
Dim  s
' Выводим строку на экран
WScript.StdOut.Write "Введите число: "
' Считываем строку
s = WScript.StdIn.ReadLine
' Выводим строку на экран
WScript.StdOut.WriteLine "Вы ввели число " & s
```

Запуск : **cscript 2.vbs**



2.3.3 Метод CreateObject

Строковый параметр **strProgID**, указываемый в методе **CreateObject**, называется программным идентификатором объекта. Пример создания объекта **WshNetwork:**

```
set WshNetwork = WScript.CreateObject("WScript.Network")
```

2.3.4 Метод Echo

Параметры **Arg1**,**Arg2**,… метода **Echo** задают аргументы для вывода. Если сценарий был запущен с помощью **wscript**, то метод **Echo** направляет вывод в диалоговое окно, если же для выполнения сценария применяется cscript.exe, то вывод будет направлен на консоль.

```
WScript.Echo 1,2,3     'Выводим числа
WScript.Echo "Привет!"  'Выводим строку
```

2.4 Объект WshShell

С помощью объекта **WshShell** можно запускать новый процесс, создавать ярлыки, работать с системным реестром, получать доступ к переменным среды и специальным папкам Windows. Создается этот объект следующим образом:

```
var WshShell=WScript.CreateObject("WScript.Shell")
```

Таблица 2.4 – Основные свойства объекта WshShell

Свойство	Описание
CurrentDirectory	Здесь хранится полный путь к текущему каталогу (к каталогу, из которого был запущен сценарий)
Environment	Содержит объект WshEnvironment, который обеспечивает доступ к переменным среды операционной системы
SpecialFolders	Содержит объект WshSpecialFolders для доступа к специальным папкам Windows (рабочий стол, меню Пуск (Start) и т. д.)

Таблица 2.5 – Основные методы объекта WshShell

Метод	Описание
AppActivate(title)	Активизирует заданное параметром title окно приложения. Строка title задает название окна.
CreateShortcut (strPathname)	Создает объект WshShortcut для связи с ярлыком Windows или объект WshUrlShortcut для связи с сетевым ярлыком. Параметр strPathname задает полный путь к создаваемому или изменяемому ярлыку
Environment(strType)	Возвращает объект WshEnvironment, содержащий переменные среды заданного вида ("System" - переменные среды операционной системы), "User" - переменные среды пользователя, "Volatile" - временные переменные, "Process" - переменные среды текущего командного окна)
Exec(strCommand)	Создает новый дочерний процесс, который запускает консольное приложение, заданное параметром strCommand.
ExpandEnvironment-Strings (strString)	Возвращает значение переменной среды текущего командного окна, заданной строкой strString (имя переменной должно быть окружено знаками "%")
Popup(strText,[nSecToWait], [strTitle], [nType])	Выводит на экран информационное окно с сообщением, заданным параметром strText. Параметр nSecToWait задает количество секунд, по истечении которых окно будет автоматически закрыто, параметр strTitle определяет заголовок окна, параметр nType указывает тип кнопок и значка для окна
RegDelete(strName)	Удаляет из системного реестра заданный параметр или раздел целиком
RegRead(strName)	Возвращает значение параметра реестра или значение по умолчанию для раздела реестра
RegWrite(strName, anyValue [,strType])	Записывает в реестр значение заданного параметра или значение по умолчанию для раздела
Run(strCommand, [intWindowStyle], [bWaitOnReturn])	Создает новый независимый процесс, который запускает приложение, заданное параметром strCommand

SendKeys(string)	Посылает одно или несколько нажатий клавиш в активное окно (эффект тот же, как если бы вы нажимали эти клавиши на клавиатуре)
SpecialFolders (strSpecFolder)	Возвращает строку, содержащую путь к специальной папке Windows, заданной параметром strSpecFolder

Пример сценария, для создания двух ярлыков - на сам выполняемый сценарий (объект **oShellLink**) и на сетевой ресурс (**oUrlLink**).

```
Dim WshShell,oShellLink,oUrlLink
' Создаем объект WshShell
Set WshShell=WScript.CreateObject("WScript.Shell")
' Создаем ярлык на файл
Set oShellLink=WshShell.CreateShortcut("Current Script.lnk")
' Устанавливаем путь к файлу
oShellLink.TargetPath=WScript.ScriptFullName
' Сохраняем ярлык
oShellLink.Save

' Создаем ярлык на сетевой ресурс
Set oUrlLink = WshShell.CreateShortcut("GSTU.URL")
' Устанавливаем URL
oUrlLink.TargetPath = "http://gstu.local"
' Сохраняем ярлык
oUrlLink.Save
```

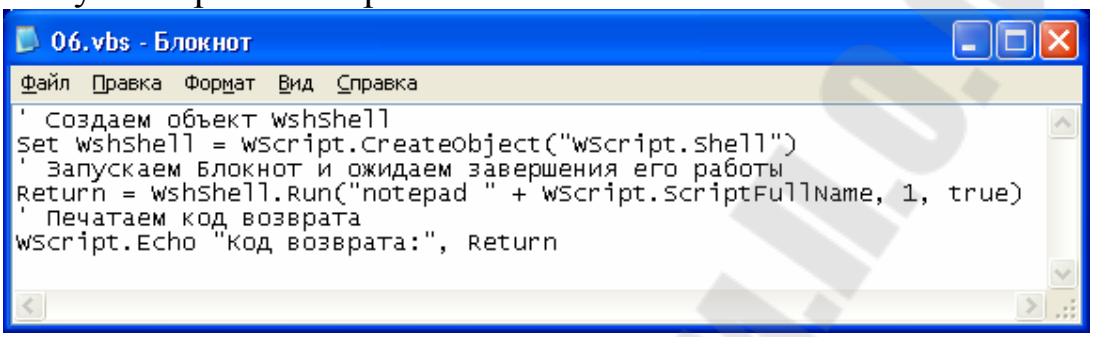
Пример сценария, для вывода некоторых переменных среды.

```
Dim WshShell,WshSysEnv
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Создание коллекции WshEnvironment
Set WshSysEnv = WshShell.Environment("SYSTEM")
WScript.Echo WshSysEnv("OS")
WScript.Echo
WshShell.Environment.Item("NUMBER_OF_PROCESSORS")
```

Пример сценария, для вывода кода выхода запущенного приложения

```
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Запускаем Блокнот и ожидаем завершения его работы
Return = WshShell.Run("notepad " + WScript.ScriptFullName, 1, true)
' Печатаем код возврата
WScript.Echo "Код возврата:", Return
```

Результат работы скрипта:



2.5 Объект WshEnvironment

Объект **WshEnvironment** позволяет получить доступ к коллекции, содержащей переменные среды заданного типа. Объект имеет свойство **Length**, в котором хранится число элементов в коллекции, и методы **Count** и **Item**. Для того чтобы получить значение определенной переменной среды, в качестве аргумента метода **Item** указывается имя этой переменной в двойных кавычках.

Пример сценария, вывода значения с переменной среды **PATH.**

```
Dim WshShell, WshSysEnv
Set WshShell=WScript.CreateObject("WScript.Shell")
Set WshSysEnv=WshShell.Environment
WScript.Echo "Системный путь:",WshSysEnv.Item("PATH")
```

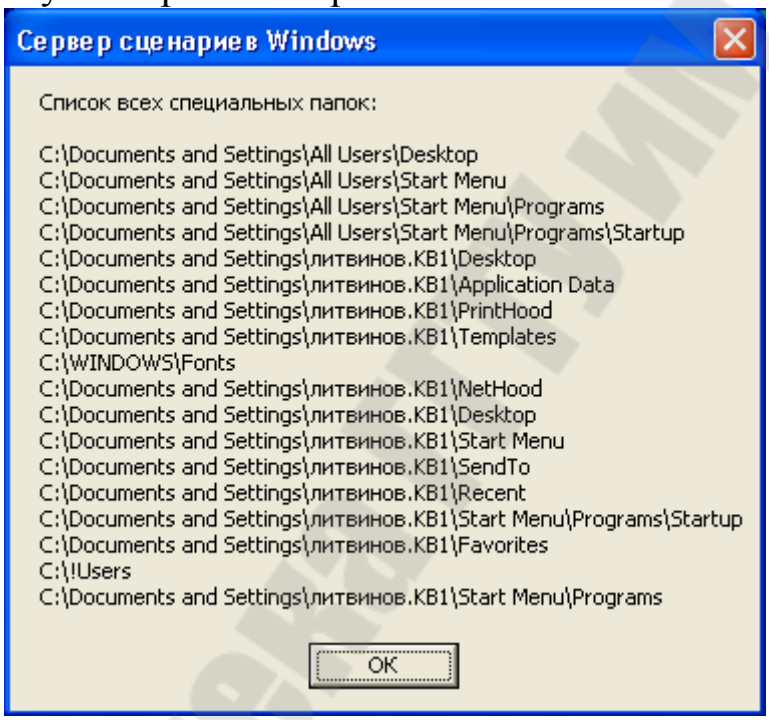
2.6 Объект WshSpecialFolders

Объект **WshSpecialFolders** обеспечивает доступ к коллекции, содержащей пути к специальным папкам Windows. В Windows поддерживаются следующие имена специальных папок: Desktop; Favorites; Fonts; MyDocuments; NetHood; PrintHood; Programs; Recent; SendTo; StartMenu; Startup; Templates; AllUsersDesktop; AllUsersStartMenu; AllUsersPrograms; AllUsersStartup.

Пример сценария, формирующего список всех имеющихся в системе специальных папок.

```
Dim WshShell, WshFldrs, SpecFldr, s  ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
' Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
s="Список всех специальных папок:" & vbCrLf & vbCrLf
' Перебираем все элементы коллекции WshFldrs
For Each SpecFldr In WshFldrs
  ' Формируем строки с путями к специальным папкам
  s=s & SpecFldr & vbCrLf
Next
WScript.Echo s
```

Результат работы скрипта:

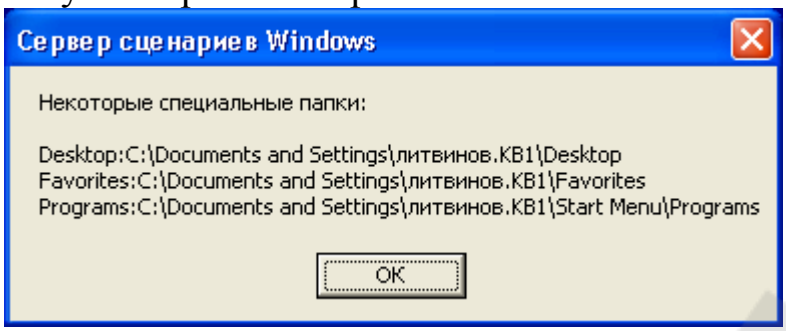


Пример сценария, для доступа к заданной специальной папке.

```
Dim WshShell, WshFldrs, s   ' Объявляем переменные
' Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
' Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
' Формируем строки с путями к конкретным специальным папкам
s="Некоторые специальные папки:" & vbCrLf & vbCrLf
s=s+"Desktop:"+WshFldrs("Desktop") & vbCrLf
```

```
s=s+"Favorites:"+WshFldrs("Favorites") & vbCrLf
s=s+"Programs:"+WshFldrs("Programs")
WScript.Echo s   ' Выводим сформированные строки на экран
```

Результат работы скрипта:



2.7 Объектная модель FileSystemObject

С помощью методов объекта **FileSystemObject** можно выполнять все основные действия с папками и файлами, а читать информацию из текстовых файлов и записывать в них данные. В таблице 4.6 приведены основные объекты для работы с файловой системой.

Таблица 2.6 – Основные объекты для работы с файловой системой

Операция	**Используемые объекты, свойства и методы**
Получение сведений об определенном диске (файловая система, метка тома, объем и т.д.)	Свойства объекта **Drive**. Объект создается с помощью метода **GetDrive** объекта **FileSystemObject**
Получение сведений о заданном каталоге или файле (дата создания, размер, атрибуты и т.д.)	Свойства объектов **Folder** и **File**. Объекты создаются с помощью методов **GetFolder** и **GetFile** объекта **FileSystemObject**
Проверка существования определенного диска, каталога или файла	Методы **DriveExists**, **FolderExists** и **FileExists** объекта **FileSystemObject**
Копирование файлов и каталогов	Методы **CopyFile** и **CopyFolder** объекта **FileSystemObject**, а также методы **File.Copy** и **Folder.Copy**
Перемещение файлов и каталогов	Методы **MoveFile** и **MoveFolder** объекта **FileSystemObject**, или методы **File.Move**

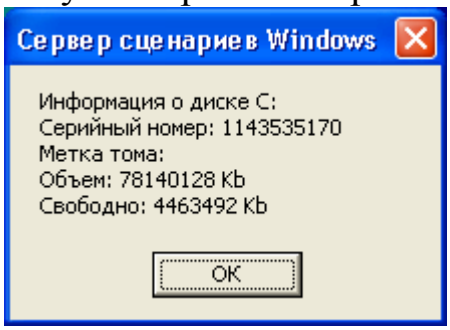
	и **Folder**.**Move**
Удаление файлов и каталогов	Методы **DeleteFile** и **DeleteFolder** объекта **FileSystemObject**, или методы **File**.**Delete** и **Folder**.**Delete**
Создание каталога	Методы **FileSystemObject**.**CreateFolder** или **Folders**.Add
Создание текстового файла	Методы **FileSystemObject**.**CreateTextFile** или **Folder**.**CreateTextFile**
Получение списка всех доступных дисков	Коллекция **Drives**, содержащаяся в свойстве **FileSystemObject**.**Drives**
Получение списка всех подкаталогов заданного каталога	Коллекция **Folders**, содержащаяся в свойстве **Folder**.**SubFolders**
Получение списка всех файлов заданного каталога	Коллекция **Files**, содержащаяся в свойстве **Folder**.**Files**
Открытие текстового файла для чтения, записи или добавления	Методы **FileSystemObject**.**CreateTextFile** или **File**.**OpenAsTextStream**
Чтение информации из заданного текстового файла или запись ее в него	Методы объекта **TextStream**

Пример сценария, для вывода на экран свойств диска C.

```
'Объявляем переменные
Dim FSO,D,TotalSize,FreeSpace,s
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект Drive для диска C
Set D = FSO.GetDrive("C:")
s = "Информация о диске C:" & VbCrLf
'Получаем серийный номер диска
s = s &"Серийный номер: " & D.SerialNumber & VbCrLf
'Получаем метку тома диска
s = s & "Метка тома: " & D.VolumeName & VbCrLf
'Вычисляем общий объем диска в килобайтах
TotalSize = D.TotalSize/1024
s = s & "Объем: " & TotalSize & " Kb" & VbCrLf
```

```
'Вычисляем объем свободного пространства диска в килобайтах
FreeSpace = D.FreeSpace/1024
s = s & "Свободно: " & FreeSpace & " Kb" & VbCrLf
'Выводим свойства диска на экран
WScript.Echo s
```

Результат работы скрипта:



Пример сценария, для вывода свойств текущего каталога.

```
Dim FSO,WshShell,FoldSize,s  'Объявляем переменные
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
'Определяем каталог, из которого был запущен сценарий
'(текущий каталог)
Set Folder = FSO.GetFolder(WshShell.CurrentDirectory)
'Получаем имя текущего каталога
s = "Текущий каталог: " & Folder.Name & VbCrLf
'Получаем дату создания текущего каталога
s = s & "Дата создания: " & Folder.DateCreated & VbCrLf
'Вычисляем размер текущего каталога в килобайтах
FoldSize=Folder.Size/1024
s = s & "Объем: " & FoldSize & " Kb" & VbCrLf
'Выводим информацию на экран
WScript.Echo s
```

Пример сценария, для проверки существования заданного файла.

```
Dim FSO,FileName 'Объявляем переменные
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
```

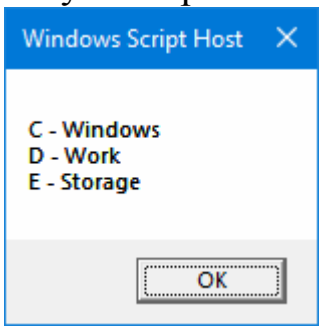
```
FileName = "C:\boot.ini"
if  FSO.FileExists(FileName)  Then
 'Выводим информацию на экран
 WScript.Echo "Файл " & FileName & " существует"
else
 'Выводим информацию на экран
 WScript.Echo "Файл " & FileName & " не существует"
end if
```

Пример сценария, для вывода на экран сведения о всех доступных дисках.

```
'Объявляем переменные
Dim FSO,s,ss,Drives,D
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем коллекцию дисков, имеющихся в системе
Set Drives = FSO.Drives
s = ""
'Перебираем все диски в коллекции
For Each D In Drives
 'Получаем букву диска
 s = s & D.DriveLetter
 s = s & " - "
 if (D.DriveType = 3) then  'Проверяем, не является ли диск сетевым
  'Получаем имя сетевого ресурса
  ss = D.ShareName
 else
  'Диск является локальным
  if (D.IsReady) then  'Проверяем готовность диска
   'Если диск готов, то получаем метку тома для диска
   ss = D.VolumeName
  else
   ss = "Устройство не готово"
  end if
 s = s & ss & VbCrLf
 end if
Next
```

```
'Выводим полученные строки на экран
WScript.Echo s
```

Результат работы скрипта:



Пример сценария, для вывода на экран названия всех файлов, которые содержатся в специальной папке «Мои документы».

```
'Объявляем переменные
Dim FSO,F,File,Files,WshShell,PathList,s
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
'Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
'Определяем путь к папке Мои документы
PathList = WshFldrs.item("MyDocuments") & "\"
'Создаем объект Folder для папки Мои документы
Set F = FSO.GetFolder(PathList)
'Создаем коллекцию файлов каталога Мои документы
Set Files = F.Files
s = "Файлы из каталога " & PathList & VbCrLf
'Цикл по всем файлам
For Each File In Files
 'Добавляем строку с именем файла
 s = s & File.Name & VbCrLf
Next
'Выводим полученные строки на экран
WScript.Echo s
```

Пример сценария, для создания в каталоге «E:\Users» подкаталогов «My Folder»

```
'Объявляем переменные
Dim FSO, F, SubFolders
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'1. способ
FSO.CreateFolder("E:\Users\My Folder")
'2. способ
'Создаем объект Folder для каталога E:\Users
Set F = FSO.GetFolder("E:\Users ")
'Создаем коллекцию подкаталогов каталога E:\Users
Set SubFolders = F.SubFolders
'Создаем каталог E:\Users\My Folder
SubFolders.Add "My Folder"
```

Пример сценария, реализующего операции записи и чтения строк из текстового файла.

```
Dim FSO,F,TextStream,s  'Объявляем переменные
' Инициализируем константы
Const ForReading = 1, ForWriting = 2, TristateUseDefault = -2

' Создаем объект FileSystemObject
Set FSO=WScript.CreateObject("Scripting.FileSystemObject")
' Создаем в текущем каталоге файл test1.txt
FSO.CreateTextFile "test1.txt"
' Создаем объект File для файла test1.txt
set F=FSO.GetFile("test1.txt")
' Создаем объект TextStream (файл открывается для записи)
Set TextStream=F.OpenAsTextStream(ForWriting, TristateUseDefault)
' Записываем в файл строку
TextStream.WriteLine "Это первая строка"
' Закрываем файл
TextStream.Close
' Открываем файл для чтения
Set TextStream=F.OpenAsTextStream(ForReading, TristateUseDefault)
' Считываем строку из файла
s=TextStream.ReadLine
' Закрываем файл
```

```
TextStream.Close
' Отображаем строку на экране
WScript.Echo "Первая строка из файла test1.txt:" & vbCrLf & vbCrLf & s
```

Пример сценария, в котором на диске E создается файл TestFile.txt, который затем копируется на рабочий стол.

```
'Объявляем переменные
Dim FSO,F,WshShell,WshFldrs,PathCopy
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Создаем файл
Set F = FSO.CreateTextFile("E:\TestFile.txt", true)
'Записываем в файл строку
F.WriteLine "Тестовый файл"
'Закрываем файл
F.Close
'Создаем объект WshShell
Set WshShell = WScript.CreateObject("Wscript.Shell")
'Создаем объект WshSpecialFolders
Set WshFldrs = WshShell.SpecialFolders
'Определяем путь к рабочему столу
PathCopy = WshFldrs.item("Desktop")+"\"
'Создаем объект File для файла C:\TestFile.txt
Set F = FSO.GetFile("E:\TestFile.txt")
'Копируем файл на рабочий стол
F.Copy PathCopy
```

Пример сценария, реализующего удаления файла E:\Users\TestFile.txt.

```
'Объявляем переменные
Dim FSO,F,FileName
'Создаем объект FileSystemObject
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")
'Задаем имя файла
FileName="E:\Users\TestFile.txt"
FSO.DeleteFile FileName
WScript.Echo "Файл удален"
```

2.8 Контрольное задание

Создать сценарий, реализующий в консольном или оконном, режиме диалог с пользователем в виде меню. Выполнение сценария прекращается, выбором пункта меню – «Выход». Первый пункт меню должен выводить информацию о создателе (ФИО, группа) и краткое описание выполняемых скриптом действий, остальные пункты реализуют действия, указанные в таблице в соответствии с выбранным вариантом. Все дополнительные параметры сценариев, задаются в результате диалога с пользователем. Количество и сложность заданий (таблица 2.7) студент определяет самостоятельно. Максимальный возможный бал по работе – 10. **Однотипные задания не выбирать**.

Таблица 2.7 – Варианты заданий

Вариант	**Задание**	**Сложность, балл**
1	Создать папку в указанном месте с заданным именем	3
2	Скопировать заданный файл с указанного места в заданное.	3
3	Скопировать все файлы с указанного места в заданное.	4
4	Удалить файлы заданного расширения в заданной папке.	4
5	Удалить все содержимое заданной папки.	3
6	Вывести на экран и сохранить в текстовом файле список папок в указанном месте с датой создания совпадающей с заданной.	5
7	Скопировать файлы заданного расширения с указанного места в папку «*BackUp*», на указанном диске.	4
8	Перенести файлы заданного расширения с указанного места в папку «*BackUp*», на заданном диске.	4
9	Вывести на экран и сохранить в текстовом файле полную структуру папок начиная с заданного места.	5
10	Перенести заданный файл с указанного места в заданное.	3

11	Переименовать заданную папку.	3
12	Архивирование заданной папки.	4
13	Архивирование файлов заданного расширения в заданной папке.	4
14	Вывести на экран список всех подключенных сетевых ресурсов	4
15	Создать ярлык для вызова заданной программы и разместить его на рабочем столе.	4
16	Создать ярлык, ссылающийся на заданную папку и разместить его на рабочем столе	4
17	Создать ярлык ссылку на заданный сетевой ресурс и разместить его на рабочем столе	4
18	Вывести на экран путь к заданной специальной папке.	4
19	Вывести на экран и сохранить в текстовом файле параметра реестра (выбрать самостоятельно)	5
20	Отредактировать заданный параметра реестра (выбрать самостоятельно). Вывести на экран прошлое значение и новое.	5
21	Вывести на экран и сохранить в текстовом файле список дисков с их размером.	5
22	Вывести на экран и сохранить в текстовом файле список дисков с значением свободного места	5
23	Вывести на экран и сохранить в текстовом файле дату создания и размер заданной папки	4
24	Вывести на экран и сохранить в текстовом файле дату создания и размер заданного файла	4
25	Вывести на экран и сохранить в текстовом файле список папок в заданном каталоге.	4
26	Вывести на экран и сохранить в текстовом файле список фалов с заданным расширением в указанном каталоге и подкаталогах.	5
27	Вывести на экран и сохранить в текстовом файле список специальных папок	4
28	Вывести на экран и сохранить в текстовом файле список файлов с заданным расширени-	5

	ем, в папке «*мои документы*» с датой их создания.	
29	Создать в заданном месте, резервную копию ярлыков рабочего стола	5
30	Определить общий размер файлов заданного расширения на указанном диске	5

Пример выполнения задания

Создать сценарий, реализующий в консольном режиме диалог с пользователем в виде меню. Сценарий создает ярлык на заданный сетевой ресурс.

```
'* Имя: Laba2.vbs
'* Язык: VBScript
'* Описание: пример лабораторной работы
'*************************************************************
Dim  s
' Выводим строку на экран
do
 WScript.StdOut.WriteLine "МЕНЮ:"
 WScript.StdOut.WriteLine "-------------------------"
 WScript.StdOut.WriteLine "1. Информация о авторе"
 WScript.StdOut.WriteLine "2. Создание ярлыка на заданный сетевой ресурс"
 WScript.StdOut.WriteLine "3. Выход"
 WScript.StdOut.Write "Выберите пункт меню:"
 ' Считываем строку
 s = WScript.StdIn.ReadLine
 ' Создаем объект WshShell
 Set WshShell = WScript.CreateObject("WScript.Shell")

 if (s="1") Then
   WScript.StdOut.WriteLine "ФИО, группа"

  elseif(s="2") Then
   WScript.StdOut.Write "Введите имя ярлыка:"
   f = WScript.StdIn.ReadLine
   ' Создаем ярлык на сетевой ресурс
```

```
   Set oUrlLink = WshShell.CreateShortcut(f+".URL")
   ' Устанавливаем URL
   WScript.StdOut.Write "Введите имя ресурса:"
   f = WScript.StdIn.ReadLine
   oUrlLink.TargetPath = f
   ' Сохраняем ярлык
   oUrlLink.Save
 End if
loop until (s="3")
```

Пример работы с элементами оконного диалога.

```
Dim WshShell, Res, Text, Title, vbOkCancel , vbOk, s
' Инициализация констант для диалоговых окон
vbOkCancel = 1
vbOk = 1

WScript.Echo "Привет!"
s = InputBox("Hello World!", 65, "MsgBox Example")
MsgBox ("You entered: " & s)

' Создание объекта WshShell
set WshShell = WScript.CreateObject("WScript.Shell")
Text = "Вы действительно хотите удалить файлы ?"
Title = "Сообщение"

' Вывод диалогового окна на экран
Res = WshShell.Popup(Text,0, Title, vbOkCancel)

'Определение, какая из кнопок была нажата в диалоговом окне
if (Res = vbOk) then
   WshShell.Popup ("Нажата кнопка ОК")
else
   WshShell.Popup ("Нажата кнопка Отмена")
end if
```

Лабораторная работа № 3
Алгоритмы планирования процессов

Цель работы: изучить алгоритмы планирования процессов.

3.1 Алгоритмы кратковременного планирования процессов

Для любого процесса, находящегося в вычислительной системе, вся информация, необходимая для совершения операций над ним, доступна операционной системе. Обычно она хранится в одной или нескольких структурах (таблицах) данных, которые принято называть - называть PCB (Process Control Block) или блоком управления процессом. Блок управления процессом является моделью процесса для операционной системы. Любая операция, производимая операционной системой над процессом, вызывает определенные изменения в PCB. В рамках принятой модели состояний процессов содержимое PCB между операциями остается постоянным.

Существует большой набор разнообразных алгоритмов планирования, которые предназначены для достижения различных целей и эффективны для разных классов задач. Многие из них могут использоваться на нескольких уровнях планирования. Рассмотрим наиболее распространённые алгоритмы.

3.2 First-Come, First-Served (FCFS)

Простейшим алгоритмом планирования является алгоритм, который принято обозначать аббревиатурой **FCFS** по первым буквам его английского названия – **First-Come, First-Served** (первым пришел, первым обслужен). Пусть процессы, находящиеся в состоянии готовность, выстроены в очередь. Когда процесс переходит в состояние готовность, он (ссылка на его PCB) помещается в конец этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на его PCB. Очередь подобного типа имеет в программировании специальное наименование – FIFO), сокращение от First In, First Out (первым вошел, первым вышел).

Такой алгоритм выбора процесса осуществляет не вытесняющее планирование. Процесс, получивший в свое распоряжение процессор, занимает его до истечения текущего CPU burst. После этого для выполнения выбирается новый процесс из начала очереди. Преимущест-

вом алгоритма FCFS является легкость его реализации, но в то же время он имеет и недостатки.

Таблица 3.1 – Данные о планируемых процессах

Процесс	p0	p1	p2
Продолжительность очередного CPU burst	13	4	1

Рассмотрим пример. Пусть в состоянии готовность находятся три процесса p0, p1 и p2, для которых известны времена их очередных CPU burst. Эти времена приведены в таблице 3.1. в некоторых условных единицах. Для определенности будем полагать, что деятельность процессов ограничивается использованием только одного промежутка CPU burst, что процессы не совершают операций ввода-вывода и что время переключения контекста так мало, что им можно пренебречь.

Если процессы расположены в очереди процессов, готовых к исполнению, в порядке p0, p1, p2, то последовательность их выполнения представлена на рисунке 3.1. Первым для выполнения выбирается процесс p0, который получает процессор на все время своего CPU burst, т. е. на 13 единиц времени. После его окончания в состояние исполнение переводится процесс p1, он занимает процессор на 4 единицы времени. И, наконец, возможность работать получает процесс p2. Время ожидания для процесса p0 составляет 0 единиц времени, для процесса p1 – 13 единиц, для процесса p2 – 13 + 4 = 17 единиц. Таким образом, среднее время ожидания в этом случае – (0 + 13 + 17)/3 = 10 единиц времени. Полное время выполнения для процесса p0 составляет 13 единиц времени, для процесса p1 – 13 + 4 = 17 единиц, для процесса p2 – 13 + 4 + 1 = 18 единиц. Среднее полное время выполнения оказывается равным (13 + 17 + 18)/3 = 16 единицам времени.

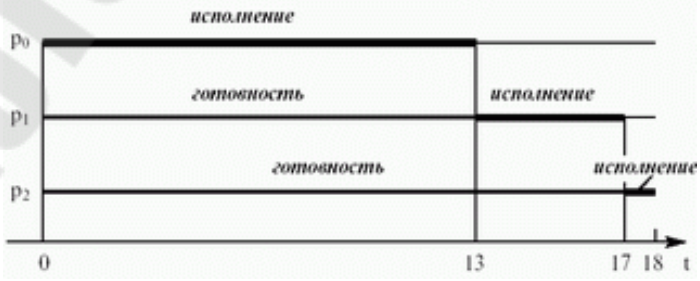


Рисунок 3.1 – Алгоритм FCFS. Выполнение процессов в порядке p0, p1, p2

Если те же самые процессы расположены в порядке p2, p1, p0, то последовательность их выполнения будет соответствовать рисунку 3.2. Время ожидания для процесса p0 равняется 5 единицам времени, для процесса p1 – 1 единице, для процесса p2 – 0 единиц. Среднее время ожидания составит (5 + 1 + 0)/3 = 2 единицы времени. Это в 5 (!) раз меньше, чем в предыдущем случае. Полное время выполнения для процесса p0 получается равным 18 единицам времени, для процесса p1 – 5 единицам, для процесса p2 – 1 единице. Среднее полное время выполнения составляет (18 + 5 + 1)/3 = 8 единиц времени, что почти в 2 раза меньше, чем при первой расстановке процессов.

Рисунок 3.2 – Алгоритм FCFS. Выполнение процессов в порядке p2, p1, p0

Как видно, среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно зависят от порядка расположения процессов в очереди. Если есть процесс с длительным CPU burst, то короткие процессы, перешедшие в состояние готовность после длительного процесса, будут долго ждать начала выполнения. Поэтому алгоритм FCFS практически неприменим для систем разделения времени – слишком большим получается среднее время отклика в интерактивных процессах.

3.3 Round Robin (RR)

Модификацией алгоритма FCFS является алгоритм, получивший название **Round Robin** (RR). По сути, это тот же алгоритм, только реализованный в режиме вытесняющего планирования. Пусть все готовые к выполнению процессы, организуют циклическую очередь, в которой каждый процесс получает в свое распоряжение процессор на строго фиксированный небольшой квант времени, обычно 10 – 100 миллисекунд (рисунок 3.3). В это время процесс находится в состоянии исполнение.

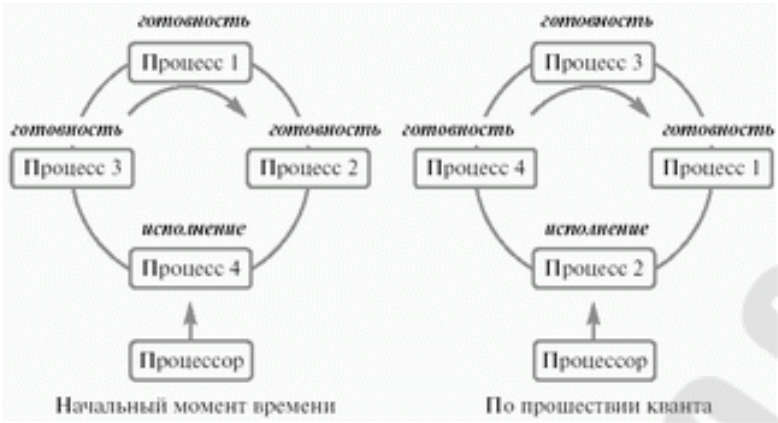


Рисунок 3.3 – Алгоритм Round Robin

Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся в состоянии готовность, в очередь FIFO. Планировщик выбирает для очередного исполнения процесс, расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении определенного кванта времени. При выполнении процесса возможны два варианта.

1. Время непрерывного использования процессора, необходимое процессу (остаток текущего CPU burst), меньше или равно продолжительности кванта времени. Тогда процесс **сам освобождает процессор** до истечения кванта времени, на исполнение поступает новый процесс из начала очереди, и таймер начинает отсчет кванта заново.
2. Продолжительность остатка текущего CPU burst процесса больше, чем квант времени. Тогда по истечении этого кванта **процесс прерывается** таймером и помещается в конец очереди процессов, готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале.

Рассмотрим предыдущий пример с порядком процессов p0, p1, p2 и величиной кванта времени равной 4. Выполнение этих процессов иллюстрируется таблицей 3.2. Обозначение "И" – для процесса, находящегося в состоянии исполнение, обозначение "Г" – для процессов в состоянии готовность, пустые ячейки соответствуют завершившимся процессам.

Таблица 3.2 – Последовательность выполнения процессов

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p0	И	И	И	И	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И
p1	Г	Г	Г	Г	И	И	И	И										
p2	Г	Г	Г	Г	Г	Г	Г	Г	И									

Первым для исполнения выбирается процесс p0. Продолжительность его CPU burst больше, чем величина кванта времени, и поэтому процесс исполняется до истечения кванта, т. е. в течение 4 единиц времени. После этого он помещается в конец очереди готовых к исполнению процессов, которая принимает вид p1, p2, p0. Следующим начинает выполняться процесс p1. Время его исполнения совпадает с величиной выделенного кванта, поэтому процесс работает до своего завершения. Теперь очередь процессов в состоянии готовность состоит из двух процессов, p2 и p0. Процессор выделяется процессу p2. Он завершается до истечения отпущенного ему процессорного времени, и очередные кванты отдаются процессу p0 как единственному не закончившему к этому моменту свою работу. Время ожидания для процесса p0 (количество символов "Г" в соответствующей строке) составляет 5 единиц времени, для процесса p1 – 4 единицы времени, для процесса p2 – 8 единиц времени. Таким образом, среднее время ожидания для этого алгоритма получается равным (5 + 4 + 8)/3 = 5,6(6) единицы времени. Полное время выполнения для процесса p0 (количество непустых столбцов в соответствующей строке) составляет 18 единиц времени, для процесса p1 – 8 единиц, для процесса p2 – 9 единиц. Среднее полное время выполнения оказывается равным (18 + 8 + 9)/3 = 11,6 единицы времени.

На производительность алгоритма RR сильно влияет величина кванта времени. Рассмотрим тот же самый пример для величины кванта времени, равной 1 (таблица 5.3). Время ожидания для процесса p0 составит 5 единиц времени, для процесса p1 – тоже 5 единиц, для процесса p2 – 2 единицы. В этом случае среднее время ожидания получается равным (5 + 5 + 2)/3 = 4 единицам времени. Среднее полное время исполнения составит (18 + 9 + 3)/3 = 10 единиц времени.

Таблица 3.3 – Последовательность выполнения процессов

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p0	И	Г	Г	И	Г	И	Г	И	Г	И	И	И	И	И	И	И	И	И
p1	Г	И	Г	Г	И	Г	И	Г	И									
p2	Г	Г	И															

При очень больших величинах кванта времени, когда каждый процесс успевает завершить свой CPU burst до возникновения прерывания по времени, алгоритм RR вырождается в алгоритм FCFS. При очень малых величинах создается иллюзия того, что каждый из n процессов работает на собственном виртуальном процессоре с произ-

водительностью ~ 1/n от производительности реального процессора. В реальных условиях при слишком малой величине кванта времени и, соответственно, слишком частом переключении контекста накладные расходы на переключение резко снижают производительность системы.

3.4 Shortest-Job-First (SJF)

При рассмотрении алгоритмов FCFS и RR мы выяснили, что существенным является порядок расположения процессов в очереди. Если короткие задачи расположены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрастает. Если известно время следующих CPU burst для процессов, находящихся в состоянии готовность, то можно выбрать для исполнения процесс не из начала очереди, а процесс с минимальной длительностью CPU burst. Если же таких процессов два или больше, то для выбора одного из них можно использовать уже известный нам алгоритм FCFS. Квантование времени при этом не применяется. Описанный алгоритм получил название "кратчайшая работа первой" или **Shortest Job First** (SJF).

SJF – алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим. При невытесняющем SJF-планировании процессор предоставляется избранному процессу на все необходимое ему время, независимо от событий, происходящих в вычислительной системе. При вытесняющем SJF-планировании учитывается появление новых процессов в очереди готовых к исполнению во время работы выбранного процесса. Если CPU burst нового процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

Рассмотрим пример работы невытесняющего алгоритма SJF. Пусть в состоянии готовность находятся четыре процесса, p0, p1, p2 и p3, для которых известны времена их очередных CPU burst. Эти времена приведены в таблице 3.4.

Таблица 3.4 – Данные о планируемых процессах

Процесс	p0	p1	p2	p3
Продолжительность очередного CPU burst	5	3	7	1

При использовании невытесняющего алгоритма SJF первым для исполнения будет выбран процесс p3, имеющий наименьшее значение продолжительности очередного CPU burst. После его завершения для исполнения выбирается процесс p1, затем p0 и, наконец, p2. Эта картина отражена в таблице 3.5.

Таблица 3.5 – Последовательность выполнения процессов

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p0	Г	Г	Г	Г	И	И	И	И	И							
p1	Г	И	И	И												
p2	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
p3	И															

Среднее время ожидания для алгоритма SJF составляет (4 + 1 + 9 + 0)/4 = 3,5 единицы времени. Для алгоритма FCFS при порядке процессов p0, p1, p2, p3 эта величина будет равняться (0 + 5 + 8 + 15)/4 = 7 единицам времени, т. е. в два раза больше, чем для алгоритма SJF.

Для рассмотрения примера вытесняющего SJF планирования мы возьмем ряд процессов p0, p1, p2 и p3 с различными временами CPU burst и различными моментами их появления в очереди процессов, готовых к исполнению (таблица 3.6).

Таблица 3.6 – Данные о планируемых процессах

Процесс	Время появления в очереди очередного CPU burst	Продолжительность
p0	0	6
p1	2	2
p2	6	7
p3	0	5

В начальный момент времени в состоянии готовность находятся только два процесса, p0 и p3. Меньшее время очередного CPU burst оказывается у процесса p3, поэтому он и выбирается для исполнения (таблица 3.7). По прошествии 2 единиц времени в систему поступает процесс p1. Время его CPU burst меньше, чем остаток CPU burst у процесса p3, который вытесняется из состояния исполнение и переводится в состояние готовность. По прошествии еще 2 единиц времени процесс p1 завершается, и для исполнения вновь выбирается процесс p3. В момент времени t = 6 в очереди процессов, готовых к исполнению, появляется процесс p2, но поскольку ему для работы нужно 7 единиц времени, а процессу p3 осталось трудиться всего 1 единицу

времени, то процесс p3 остается в состоянии исполнение. После его завершения в момент времени t = 7 в очереди находятся процессы p0 и p2, из которых выбирается процесс p0. Последним получит возможность выполняться процесс p2.

Таблица 3.7 – Последовательность выполнения процессов

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
p0	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И							
p1			И	И																
p2							Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
p3	И	И	Г	Г	И	И	И													

Основную сложность при реализации алгоритма SJF представляет невозможность точного знания продолжительности очередного CPU burst для исполняющихся процессов. При краткосрочном планировании мы можем делать только прогноз длительности следующего CPU burst, исходя из предыстории работы процесса.

3.5 Приоритетное планирование

При приоритетном планировании каждому процессу присваивается определенное числовое значение – приоритет, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего CPU burst. Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс.

Алгоритмы назначения приоритетов процессов могут опираться как на внутренние параметры, связанные с происходящим внутри вычислительной системы, так и на внешние по отношению к ней. К внутренним параметрам относятся различные количественные и качественные характеристики процесса такие как: ограничения по времени использования процессора, требования к размеру памяти, число открытых файлов и используемых устройств ввода-вывода, отношение средних продолжительностей I/O burst к CPU burst и т. д. Алгоритмы SJF и гарантированного планирования используют внутренние параметры. В качестве внешних параметров могут выступать важность процесса. Планирование с использованием приоритетов может быть как вытесняющим, так и невытесняющим. При вытесняющем планировании процесс с более высоким приоритетом, вытесняет ис-

полняющийся процесс с более низким приоритетом. В случае невытесняющего планирования он просто становится в начало очереди готовых процессов.

Пусть в очередь процессов, находящихся в состоянии готовность, поступают те же процессы, что и в примере для вытесняющего алгоритма SJF, только им дополнительно им присвоены приоритеты (таблица 3.8). Будем предполагать, что большее значение соответствует меньшему приоритету, т. е. наиболее приоритетным в нашем примере является процесс p3, а наименее приоритетным – процесс p0.

Таблица 3.8 – Данные о планируемых процессах

Процесс	Время появления в очереди	Продолжительность очередного CPU burst	Приоритет
p0	0	6	4
p1	2	2	3
p2	6	7	2
p3	0	5	1

При невытесняющем приоритетном планировании, первым для выполнения в момент времени t = 0 выбирается процесс p3, как обладающий наивысшим приоритетом. После его завершения в момент времени t = 5 в очереди процессов, готовых к исполнению, окажутся два процесса p0 и p1. Большим приоритетом обладает p1, он и начнет выполняться (таблица 3.9). Затем в момент времени t = 8 для исполнения будет избран процесс p2, и лишь потом – процесс p0.

Таблица 3.9 – Последовательность выполнения процессов

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
p0	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И
p1			Г	Г	Г	И	И													
p2							Г	И	И	И	И	И	И	И						
p3	И	И	И	И	И															

В случаи вытесняющего приоритетного планирования (таблица 3.10), первым, как и в предыдущем случае, начнет исполняться процесс p3, а по его окончании – процесс p1. Однако в момент времени t = 6 он будет вытеснен процессом p2 и продолжит свое выполнение только в момент времени t = 13. Последним, как и раньше, будет исполняться процесс p0.

В рассмотренном выше примере приоритеты процессов с течением времени не изменялись. Такие приоритеты принято называть ста-

тическими. Более гибкими являются динамические приоритеты процессов, изменяющие свои значения по ходу исполнения процессов. Схемы с динамической приоритетностью гораздо сложнее в реализации и связаны с большими издержками по сравнению со статическими схемами. Однако их использование предполагает, что эти издержки оправдываются улучшением работы системы.

Таблица 3.10 – Последовательность выполнения процессов

Время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
p0	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И
p1			Г	Г	Г	И	Г	Г	Г	Г	Г	Г	Г	И						
p2							И	И	И	И	И	И	И							
p3	И	И	И	И	И															

3.6 Контрольное задание 1. Не вытесняющие алгоритмы планирования процессов

В соответствии с вариантом (**номер по журналу**), для данных приведенных в таблице 3.11, выполнить следующие алгоритмы планирований – **First-Come, First-Served** прямой и обратный (1 балл), **Round Robin** (1 балл), **Shortest-Job-First** не вытесняющий (2 балла), **Shortest-Job-First** не вытесняющий приоритетный (2 балла).

Вычислить полное время выполнения всех процессов и каждого в отдельности, время ожидания для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц иллюстрирующих работу процессов.

3.7 Контрольное задание 2. Вытесняющие алгоритмы планирования процессов

В соответствии с вариантом (**номер по журналу**), для данных приведенных в таблице 5.11, выполнить вытесняющие алгоритмы планирования – ***Shortest-Job-First*** (2 балла) и ***Shortest-Job-First*** приоритетный (2 балла).

Вычислить полное время выполнения все процессов и каждого в отдельности, время ожидание для каждого процесса. Рассчитать среднее время выполнения процесса и среднее время ожидания. Результаты оформить в виде таблиц иллюстрирующих работу процессов.

Сравнить полученные результаты и сделать выводы о эффективности рассмотренных алгоритмов.

Таблица 3.11 – Варианты заданий

Вариант	Продолжительности процессов	Время появления в очереди	Приоритеты процессов
1	P0 – 5; P1 – 8; P2 – 2; P3 – 4;	P0 – 0; P1 – 3; P2 – 1; P3 – 4;	P0 – 2; P1 – 1; P2 – 3; P3 – 3;
2	P0 – 10; P1 – 3; P2 – 3; P3 – 7;	P0 – 2; P1 –0 ; P2 – 4; P3 –5;	P0 – 3; P1 – 2; P2 – 3; P3 – 1;
3	P0 – 5; P1 – 8; P2 – 2; P3 – 4;	P0 – 3; P1 – 1; P2 – 3; P3 –0;	P0 – 3; P1 – 2; P2 – 3; P3 – 1;
4	P0 – 2; P1 – 5; P2 – 9; P3 – 3;	P0 – 0; P1 – 4; P2 – 6; P3 – 8;	P0 – 1; P1 – 3; P2 – 3; P3 – 2;
5	P0 – 7; P1 – 8; P2 – 3; P3 – 4;	P0 – 4; P1 – 0; P2 – 3; P3 – 5;	P0 – 1; P1 – 2; P2 – 3; P3 – 4;
6	P0 – 9; P1 – 2; P2 – 2; P3 – 3;	P0 – 3; P1 – 0; P2 – 0; P3 – 4;	P0 – 3; P1 – 1; P2 – 3; P3 – 3;
7	P0 – 7; P1 – 2; P2 – 3; P3 – 1;	P0 – 0; P1 – 4; P2 – 3; P3 – 0;	P0 – 1; P1 – 3; P2 – 2; P3 – 3;
8	P0 – 2; P1 – 3; P2 – 1; P3 – 8;	P0 – 4; P1 – 4; P2 – 0; P3 – 0;	P0 – 3; P1 – 1; P2 – 3; P3 – 4;
9	P0 – 3; P1 – 5; P2 – 2; P3 – 7;	P0 – 3; P1 – 0; P2 – 1; P3 – 7;	P0 – 1; P1 – 3; P2 – 4; P3 – 1;
10	P0 – 7; P1 – 3; P2 – 4; P3 – 6;	P0 – 1; P1 – 2; P2 – 4; P3 – 0;	P0 – 2; P1 – 1; P2 – 3; P3 – 4;
11	P0 – 3; P1 – 2; P2 – 5; P3 – 6;	P0 – 3; P1 – 4; P2 – 4; P3 – 0;	P0 – 1; P1 – 1; P2 – 3; P3 – 3;
12	P0 – 8; P1 – 3; P2 – 10; P3 – 2;	P0 – 8; P1 – 4; P2 – 0; P3 – 3;	P0 – 2; P1 – 2; P2 – 1; P3 – 4;
13	P0 – 3; P1 – 7; P2 – 1; P3 – 3;	P0 – 4; P1 – 0; P2 – 0; P3 – 0;	P0 – 4; P1 – 3; P2 – 3; P3 – 2;
14	P0 – 4; P1 – 7; P2 – 6; P3 – 3;	P0 – 4; P1 – 4; P2 – 0; P3 – 7;	P0 – 1; P1 – 4; P2 – 3; P3 – 3;
15	P0 – 1; P1 – 3; P2 – 2; P3 – 7;	P0 – 0; P1 – 4; P2 – 6; P3 – 4;	P0 – 2; P1 – 1; P2 – 4; P3 – 1;
16	P0 – 5; P1 – 3; P2 – 4; P3 – 2;	P0 – 0; P1 – 4; P2 – 8; P3 – 4;	P0 – 3; P1 – 1; P2 – 3; P3 – 1;
17	P0 – 2; P1 – 4; P2 – 1; P3 – 6;	P0 – 3; P1 – 0; P2 – 6; P3 – 7;	P0 – 3; P1 – 4; P2 – 1; P3 – 3;
18	P0 – 3; P1 – 6; P2 – 3; P3 – 1;	P0 – 4; P1 – 1; P2 – 1; P3 – 0;	P0 – 1; P1 – 3; P2 – 3; P3 – 1;
19	P0 – 10; P1 – 2; P2 – 5; P3 – 3;	P0 – 11; P1 – 7; P2 – 3; P3 – 0;	P0 – 3; P1 – 3; P2 – 1; P3 – 2;
20	P0 – 8; P1 – 3; P2 – 2; P3 –3 ;	P0 – 0; P1 – 4; P2 – 7; P3 – 3;	P0 – 2; P1 – 2; P2 – 3; P3 – 1;
21	P0 – 3; P1 – 6; P2 – 4; P3 – 2;	P0 – 7; P1 – 0; P2 – 5; P3 – 3;	P0 – 4; P1 – 2; P2 – 4; P3 – 1;
22	P0 – 7; P1 – 6; P2 – 5; P3 – 3;	P0 – 0; P1 – 4; P2 – 0; P3 – 4;	P0 – 1; P1 – 3; P2 – 4; P3 – 2;
23	P0 – 3; P1 – 7; P2 – 4; P3 – 2;	P0 – 6; P1 – 0; P2 – 0; P3 – 7;	P0 – 3; P1 – 1; P2 – 2; P3 – 1;
24	P0 – 2; P1 – 5; P2 – 1; P3 – 7;	P0 – 1; P1 – 8; P2 – 0; P3 – ;	P0 – 2; P1 – 1; P2 – 2; P3 – 3;
25	P0 – 3; P1 – 4; P2 – 7; P3 – 1;	P0 – 3; P1 – 4; P2 – 4; P3 – 0;	P0 – 1; P1 – 1; P2 – 2; P3 – 2;

Для Round Robin (RR) величина кванта времени 3 для всех вариантов. Для приоритетных алгоритмов меньшее значение соответствует более высокому приоритету.

Лабораторная работа № 4
Программирование планировщиков процессов

Цель работы: изучение алгоритмов программирования планировщиков процессов.

4.1 Контрольное задание

Разработать программу (любой язык программирования), осуществляющую моделирование работы заданного алгоритма планирования. В процессе работы программы, на экран должна выводиться следующая информация:

1. Номер текущего кванта времени процессора.
2. Таблица процессов с указанием имени процессов, продолжительности и времени появления процесса, приоритета (в зависимости от задания), оставшегося времени выполнения.
3. Таблица планирования процессов с отображением текущего состояния процессов.

После запуска, программа должна в диалоговом режиме, запросить следующую информацию процессах – имя, длительность, приоритет, время появления. Для алгоритмов RR длительность кванта времени. Выполнение должно производиться в **пошаговом режиме** (по нажатию на кнопку). По окончанию работы процесса на экране должно выводиться сообщение о его завершении «*Процесс 1 завершил работы*».

Примерный вариант предоставления информации о работе планировщика процессов, представлен на рисунке 4.1. Некоторые колонки в зависимости от заданного алгоритма могут отсутствовать.

№	Имя	Появление	Длительность	Приоритет	Осталось
1	P6	0	3	3	1
2	P2	2	4	2	3
3	P3	3	5	1	4

Текущий квант времени процесса: 6

№	Имя	0	1	2	3	4	5	6	…	…	…	…	…
1	P6	И	И	И									
2	P2	Г	Г	Г	И	Г	Г	И	И	И	И	И	И
3	P3	Г	Г	Г	Г	И	И						

Рисунок 4.1 – Пример предоставления информации о работе планировщика

Вариант задания (таблица 4.1) выбирается студентом самостоятельно, оценивая его сложность. Максимальный возможный бал по работе – 10. Дополнительные 2 балла могут быть добавлены преподавателем за:

1. Удобный интерфейс с инструментами автоматизации ввода;
2. Алгоритмические решения и др.

Таблица 4.1 – Варианты заданий

Вариант	**Алгоритм планировщика**	**Сложность, балл**
1	Алгоритм FCFS. Не вытесняющий	4
2	Алгоритм FCFS. Не вытесняющий, приоритетный	4
3	Алгоритм FCFS. Вытесняющий.	5
4	Алгоритм FCFS. Вытесняющий, приоритетный	5
5	Алгоритм SJF. Не вытесняющий	5
6	Алгоритм SJF. Не вытесняющий, приоритетный	6
7	Алгоритм SJF. Вытесняющий.	7
8	Алгоритм SJF. Вытесняющий, приоритетный	7
9	Алгоритм RR.	5
10	Алгоритм RR. Не вытесняющий, приоритетный	5
11	Алгоритм RR. Не вытесняющий, приоритетный (задается длительностью процесса)	6
12	Алгоритм RR. Вытесняющий, приоритетный	7
13	Алгоритм RR. Вытесняющий, приоритетный (задается длительностью процесса)	7
14	Алгоритм RR. Не вытесняющий, приоритетный (приоритет увеличивается на единицу при каждом 10 (можно задавать при вводе) кванте состояния – «ГОТОВНОСТЬ»)	8
15	Алгоритм RR. Вытесняющий, приоритетный (приоритет увеличивается на единицу при каждом 10 (можно задавать при вводе) кванте состояния – «ГОТОВНОСТЬ»)	8

Список использованных источников

1. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум, Т. Остин ; [перевел с англ. Е. Матвеев]. - 6-е изд.. - Санкт-Петербург [и др.] : Питер, 2014. - 811 с.
2. Таненбаум, Э. Современные операционные системы : [перевод с английского] / Э. Таненбаум. - 3-е изд.. - Санкт-Петербург [и др.] : Питер, 2015.
3. Степанов А. Н. Архитектура вычислительных систем и компьютерных сетей : учеб. пособие для вузов. - Санкт-Петербург : Питер, 2007. - 508 с..
4. Сущенко С. П. Архитектура вычислительных систем. Томск: «СКК-Пресс», 2006. - 198 с.
5. В. Ф. Мелехин, Е. Г. Павловский. Вычислительные машины, системы и сети. М:Академия, 2010 г., - 560 с.
6. Таненбаум, Э. Архитектура компьютера / Э. Таненбаум. - 5-е изд.. - Санкт-Петербург [и др.] : Питер, 2010.
7. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. - 3-е изд.. - Санкт-Петербург [и др.] : Питер, 2010. - 1115 с.
8. Цилькер Б. Я., Орлов С. А. Организация ЭВМ и систем. Учебник для вузов. – СПб.: Питер, 2004. - 668 с.

Литвинов Дмитрий Александрович

АРХИТЕКТУРА ОПЕРАЦИОННЫХ СИСТЕМ

Практикум
по выполнению лабораторных работ
для слушателей специальности переподготовки
1-40 01 73 «Программное обеспечение информационных систем»
заочной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 26.10.23.
Рег. № 153Е.
http://www.gstu.by