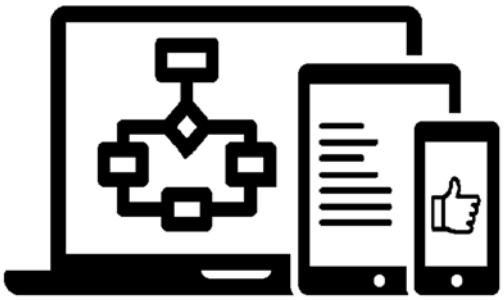
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Методические рекомендации к самостоятельной работе для студентов специальности 1-53 01 02 «Автоматизированные системы обработки информации» заочной формы обучения



Могилев 2022

УДК 004.7
ББК 32.973.26
С40

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «31» января 2022 г., протокол № 7

Составитель ст. преподаватель Е. А. Зайченко

Рецензент канд. техн. наук, доц. В. М. Ковальчук

В методических рекомендациях представлены описание аудиторной контрольной работы, критерии её оценки, изложена последовательность изучения теоретических вопросов, приведены основные термины и понятия, а также перечень заданий для написания аудиторной контрольной работы.

Учебно-методическое издание

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Ответственный за выпуск	В. В. Кутузов
Корректор	Т. А. Рыжикова
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Целью дисциплины является получение студентами знаний о множестве задач, которые решает операционная система, об особенностях разработки системного программного обеспечения, а также о перспективных направлениях в развитии современных операционных систем.

В результате изучения дисциплины студент будет знать:

– способы использования функций операционной системы и администрирования;

– назначение и возможности операционной системы;

– командные средства системного программирования.

Методические рекомендации включают в себя краткие теоретические сведения, задания и примеры выполнения заданий, на основании которых выполняются аудиторные контрольные работы (АКР) по дисциплине «Системное программное обеспечение».

На выполнение каждой аудиторной контрольной работы согласно учебной программе студентам отводится 2 ч.

Для получения зачета по АКР необходимо написать решение задачи.

АКР оценивается исходя из 10 (десяти) баллов. АКР считается зачтенной, если сумма полученных баллов составляет не менее 5 (пяти) баллов.

1 Самостоятельная работа № 1

1.1 Создание командных файлов ОС Windows

1.1.1 Краткие теоретические сведения.

Пакетные (командные) файлы, также называемые программами пакетной обработки или сценариями, позволяют упростить выполнение утомительных и часто используемых операций. Для ранних версий операционных систем семейства Windows в качестве интерпретатора команд использовался командный процессор command.com, для всех остальных ОС семейства Windows (NT//XP/Vista/7/8 и старше) используется командный процессор cmd.exe. Несмотря на непрерывное совершенствование средств создания и выполнения сценариев с использованием объектно-ориентированных языков, обычная командная строка и простые командные файлы по-прежнему остаются основным инструментом для выполнения рутинных действий, диагностики сетевых проблем, автоматизации процессов резервного копирования и т. п. Пакетный файл в ОС Windows представляет собой неформатированный текстовый файл, содержащий одну или несколько команд и имеющий расширение имени **.bat** или **.cmd.** Когда имя такого файла вводится в командной строке (расширение можно не указывать), программа Cmd.exe выполняет по порядку команды, записанные в файле. В пакетный файл можно включить любую команду. Некоторые команды, такие как **for**, **goto** и **if**, позволяют выполнять обработку условий в пакетных файлах. Например, **if** позволяет запускать команды в зависимости от выполнения заданного условия. Другие команды позволяют управлять вводом и выводом, а также запускать другие пакетные файлы.

Прервать выполнение командного файла можно с помощью комбинации ***Ctrl + C*** или ***Ctrl + Break.*** В командном файле можно вызывать другой командный файл, указав его имя и, если надо, параметры. Но после окончания работы вызванного файла выполнение исходного командного файла продолжено не будет. Если нужно продолжить выполнение после выполнения вложенного файла, можно воспользоваться командой **CALL**.

Для просмотра перечня основных команд командного процессора **CMD** служит команда HELP. Для получения сведений об определенной команде наберите HELP имя команды.

1.1.2 Команды, используемые в командных файлах ОС Windows.

Команда **echo**.

Вывод на экран сообщения или задание режима вывода на экран сообщений команд. Вызванная без параметров команда **echo** выводит текущий режим.

Синтаксис

echo [{**on**|**off**}] [*сообщение*]

Параметры:

{**on**|**off**} – включение или отключение режима отображения на экране информации о работе команд;

сообщение – текст для вывода на экран.

Команда **echo** *сообщение* может оказаться полезной, если отключен режим отображения работы команд. Для вывода сообщений из нескольких строк без вывода дополнительных команд между ними следует использовать несколько последовательных команд **echo** *сообщение* после команды **echo off** в пакетной программе.

Если используется команда **echo off**, приглашение командной строки не отображается на экране. Чтобы отобразить приглашение, введите команду **echo on**.

Чтобы отключить вывод строк, введите символ «коммерческого эт» (@) перед командой в пакетном файле.

Чтобы вывести на экране пустую строку, введите следующую команду:

echo.

Если требуется отключить режим отображения команд и при этом не выводить на экран строку самой команды **echo**, введите символ @ перед командой:

@echo off

Команда **goto**.

В пакетной программе передает управление Windows в строку, определенную меткой. Когда метка найдена, выполнение продолжается со следующей за ней строки.

Синтаксис

goto *метка*

Параметры:

метка – строка в пакетной программе, к которой выполняется переход.

Работа с расширениями команды **goto**.

Если расширения команды включены (по умолчанию они включены) и в команде **goto** используется метка **:EOF**, управление будет передано в конец файла текущего пакетного сценария для выхода из него без назначения метки. Когда используется команда **goto** с меткой **:EOF**, перед меткой должно быть вставлено двоеточие. Например: **goto :EOF**

Использование допустимых значений *метки*.

Метка может включать пробелы, но не может включать другие разделители, такие как точка с запятой или знак равенства. В команде **goto** используются только первые восемь знаков метки.

Совпадение *меток* в пакетных программах.

Метка, заданная в команде, должна соответствовать метке в тексте пакетной программы. В Windows строка пакетной программы, начинающаяся с двоеточия (:), опознается как метка и не обрабатывается как команда. Если в пакетном файле не содержится заданной метки, программа будет остановлена, а на экран будет выведено следующее сообщение:

Метка не найдена.

Команда **Pause**.

Приостанавливает выполнение пакетной программы с выводом сообщения об ожидании нажатия пользователем любой клавиши.

Синтаксис

pause

Примечания.

При запуске команды **prompt** выводится следующее сообщение:

Для продолжения нажмите любую клавишу . . .

Если при работе пакетной программы была нажата комбинация CTRL + C для ее остановки, то на экран будет выведено следующее сообщение:

Завершить выполнение пакетного файла [Y(да)/N(нет)]?

Если была нажата клавиша Y (подтверждение), пакетная программа будет закончена и управление вернется операционной системе. По этой причине команда **pause** может быть вставлена перед разделом пакетного файла, который потребуется пропустить. Команда **pause** приостанавливает выполнение пакетной программы, аналогичное действие можно сделать, нажав комбинацию клавиш CTRL+C и затем Y.

Команда **rem**.

Добавляет комментарии в пакетные файлы или файлы настройки.

Синтаксис

rem [*текст*]

Параметры:

текст – задает строку символов, используемую в качестве комментария.

1.1.3 Параметры командных файлов.

Командным файлам из командной строки могут быть переданы аргументы. Аргументы задаются параметрами командной строки после имени файла. Файл Cmd.exe использует переменные с %0 по %9. При использовании пакетных параметров переменная %0 заменяется на имя пакетного файла, а переменные с %1 по %9 — на соответствующие аргументы, напечатанные в командной строке. Для доступа к переменным больше %9 используется команда **shift**. Параметр %* ссылается на все аргументы, которые передаются пакетному файлу, за исключением параметра %0.

Команда **shift**.

Изменяет положения пакетных параметров в пакетных файлах.

Синтаксис

shift

Параметры отсутствуют.

Использование параметра командной строки **shift** с расширениями командного процессора.

Если расширения командного процессора разрешены (используются по умолчанию), команда **shift** поддерживает ключ **/n**, который указывает команде начинать сдвиг с *n*-го аргумента, где *n* – число от нуля до восьми. Например:

SHIFT /2

сдвинет %3 на %2, %4 на %3 и т. д. %0 и %1 останутся неизменными.

Работа команды **shift**.

Эта команда изменяет значения замещаемых параметров %0 – %9 путем копирования каждого параметра в предыдущий. Другими словами, значение %1 копируется в %0, значение %2 – в %1 и т. д. Такой прием оказывается полезным при написании пакетных файлов, выполняющих одну и ту же операцию над любым числом параметров.

Работа более чем с 10 пакетными параметрами.

Команда **shift** также может быть использована для создания пакетных программ, воспринимающих более 10 параметров. В командной строке такой программы можно задать более 10 параметров, при этом все параметры, следующие за десятым (то есть за переменной %9) будут последовательно помещены в эту переменную %9.

Использование %* с командой **shift**.

Команда **shift** не влияет на пакетный параметр %*.

Смещение параметров в обратном направлении.

Команды, выполняющей обратный сдвиг, не существует. После сдвига параметров командой **shift** начальное значение первого параметра (%0) не может быть восстановлено.

Команда **call**.

Вызов одного пакетного файла из другого без завершения выполнения первого файла. Команда **call** принимает метки в качестве объекта вызова. Используемая в командной строке, а не в сценарии или пакетном файле команда **call** игнорируется.

Синтаксис

call [[*диск***:**][*путь*] *имя_файла* [*пакетные_параметры*]] [**:***метка* [*аргументы*]]

Параметры:

[*диск***:**][*путь*] *имя_файла* - задает имя и местоположение пакетного файла для запуска. Параметр *имя_файла* должен иметь расширение .bat или .cmd;

пакетные_параметры - задает данные командной строки, используемые программой пакетной обработки, включая параметры командной строки, имена файлов, пакетные параметры (в диапазоне от %0 до %9) или переменные (например, *%baud%*);

:*метка* – указывает метку, на которую должно быть передано управление программы пакетной обработки. При использовании команды **call** с этим параметром создается новый контекст пакетного файла, а управление передается инструкции, следующей за указанной меткой. Когда первый раз встречается конец пакетного файла (после перехода на метку), управление возвращается на инструкцию, следующую за инструкцией **call**. При втором достижении конца файла выполнение пакетной программы прекращается;

аргументы – задает данные командной строки, которые передаются в новый экземпляр программы пакетной обработки, начинающейся с **:***метки*, включая параметры командной строки, имена файлов, пакетные параметры (в диапазоне от %1 до %9) или переменные (например, *%baud%*).

Пример

Для вызова программы Checknew.bat из другого пакетного файла в тексте родительского пакетного файла введите следующую строку:

call checknew

Если родительская пакетная программа принимает два пакетных параметра и требуется передать их в файл Checknew.bat, включите следующую команду в родительскую пакетную программу:

call checknew %1 %2

1.1.4 Организация в пакетных файлах проверки условий и циклов.

Команда **if**.

Обработка условий в пакетных программах.

Синтаксис

if [**not**] **errorlevel** *число команда* [**else** *выражение*]

if [**not**] *строка1==строка2 команда* [**else** *выражение*]

if [**not**] **exist** *имя_файла команда* [**else** *выражение*]

Параметры:

not – задает выполнение команды только в случае невыполнения условия;

errorlevel *число* – условие выполняется, если предыдущая команда, обработанная интерпретатором команд Cmd.exe, завершилась с кодом, равным или большим *числа*;

команда – команда, которая должна быть обработана в случае выполнения условия;

строка1==строка2 – условие выполняется, если строки *строка1* и *строка2* совпадают. Строки могут быть заданы явно или могут быть пакетными переменными (например, *%1*). Явно заданные строки нет необходимости заключать в кавычки;

exist *имя_файла* – условие выполняется, если существует файл с именем *имя_файла*;

оп_сравнения – трехзначный оператор сравнения. В таблице 1 перечислены допустимые значения *оп_сравнения*.

Таблица 1 – Операторы сравнения

Оператор	Описание
EQU	Равно
NEQ	Не равно
LSS	Меньше
LEQ	Меньше или равно
GTR	Больше
GEQ	Больше или равно

Если условие, заданное в команде **if**, выполняется, будет выполнена команда, следующая за условием. Если условие не выполняется, команда, заданная

в операторе **if**, пропускается, а управление переходит к команде оператора **else**, если она задана.

Когда программа завершается, она возвращает код завершения. С помощью параметра **errorlevel** коды завершения можно использовать в качестве условий.

Команда **for**.

Запуск некоторой команды для каждого файла из заданного множества.

Синтаксис

for {**%***переменная*|**%%***переменная*} **in (***множество***) do** *команда* [*ПараметрыКоманднойСтроки*]

Параметры:

{**%***переменная*|**%%***переменная*} – обязательный параметр;

(*множество***)** – обязательный параметр. Задает один или несколько файлов, каталогов, диапазон значений или текстовых строк, подлежащих обработке заданной командой. Скобки являются обязательными;

команда – обязательный параметр. Задает команду, которая будет выполнена для каждого файла, каталога диапазона значений или текстовой строки, включенной в указанный параметр **(***множество***)**;

ПараметрыКоманднойСтроки – задает параметры командной строки, которые используются с указанной командой.

Использование программы **for**.

Команда **for** может быть использована в пакетном файле или непосредственно из командной строки.

Использование параметров командной строки пакетного файла.

Перечисленные ниже атрибуты применяются к команде **for**.

В команде **for** переменная **%%***переменная* (или **%%***переменная*) будет заменяться текстовой строкой из заданного параметра *множество*, пока параметр *команда* не обработает все файлы этого множества.

Имена параметров *переменная* команды **for** учитывают регистр буквы, они являются глобальными, и одновременно может быть активно не больше 52 переменных.

Для обозначения параметра *переменная* можно использовать любые символы, кроме цифр 0–9, чтобы не было конфликта с параметрами пакетных файлов *%0–%9*. Для простых пакетных файлов вполне достаточно обозначений с одним символом, например **%%f**.

Задание множества файлов.

Параметр *множество* может представлять группу файлов или несколько групп файлов. Для задания групп файлов можно использовать подстановочные знаки (* и ?). Следующие множества файлов являются допустимыми:

(*.doc *.txt *.me)

(jan*.doc jan*.rpt feb*.doc feb*.rpt)

(ar??1991.* ap??1991.*)

1.2 Пример выполнения задания

Задача. Написать пакетный файл, который будет выполнять копирование в каталог, заданный первым параметром, множества файлов, имена которых заданы в качестве остальных параметров командной строки. Имена копируемых файлов записывать в файл протокола. Выполнять проверку наличия параметров, в случае их отсутствия выводить сообщение об ошибке.

Пакетный файл может иметь вид:

```
@echo on
If "%1" == "" echo Error & goto :EOF
If "%2" == "" echo Error & goto :EOF
:getfile
if "%2"=="" goto end
copy %2 %1
echo %2 >> proto.txt
shift /2
goto getfile
echo EXIT!
:end
```

При запуске файла после его имени необходимо записать имя каталога и имена файлов, которые надо скопировать, например:

Mycopy.bat каталог файл1 файл2 файл3

1.3 Задания для самостоятельного выполнения

В таблице 2 представлены варианты заданий.

Таблица 2 – Варианты заданий

Номер варианта	Задание
1	Написать пакетный файл, который будет проверять в каталогах, указанных в качестве параметров, наличие файлов с расширением .TMP и удалять их. Если пути поиска не заданы, вывести сообщение. Создать протокол удаляемых файлов
2	Написать пакетный файл, который будет копировать из текущего каталога все файлы с расширением .DRV, кроме одного файла, указанного в качестве второго параметра командной строки, в каталог, указанный первым параметром. Если имя каталога, в который должно производиться копирование, не задано, то вывести сообщение об этом и прервать выполнение файла
3	Написать пакетный файл, который будет проверять наличие в указанном в качестве первого параметра каталоге файлов. Если файлы там есть, копировать их в указанные в качестве остальных параметров каталоги. Если копирование прошло успешно, очистить исходный каталог. Проверять наличие параметров в командной строке, выводить сообщения об ошибках

Окончание таблицы 2

Номер варианта	Задание
4	Написать пакетный файл, который будет копировать из текущего каталога все файлы с расширением .txt, кроме одного файла, указанного в качестве первого параметра командной строки, в каталог, указанный вторым параметром. Переписывать только те файлы, которые новее одноименных в каталоге-приемнике. Если произошла ошибка копирования, выдать сообщение и прервать выполнение файла
5	Создать пакетный файл, который выводил бы содержимое каталогов, указанных в качестве параметров командной строки в файл протокола, находящийся в каталоге, заданном качестве первого параметра. Проверять наличие параметров в командной строке, выводить сообщения об ошибках
6	Написать пакетный файл, который будет копировать из корневого каталога съемного диска все файлы с расширением .cpp в заданные в качестве параметров каталоги. Записывать в файл logcopy.txt дату выполнения операции и имя каждого копируемого файла
7	Написать пакетный файл, который будет осуществлять поиск файла, заданного в качестве первого параметра в каталогах, заданных в качестве остальных параметров. Вести протокол поиска, записывая, в каких каталогах обнаружен файл. Файл протокола сделать скрытым
8	Создать пакетный файл, который выводил бы содержимое каталогов, указанных в качестве параметров командной строки, в файл протокола, находящийся в каталоге, заданном качестве первого параметра. Проверять наличие параметров в командной строке, выводить сообщения об ошибках
9	Написать пакетный файл, который будет копировать из корневого каталога съемного диска все файлы с расширением .txt в заданные в качестве параметров каталоги. Записывать в файл logcopy.txt имя каждого копируемого файла, дату и время копирования
10	Написать пакетный файл, который будет проверять в каталогах, указанных в качестве параметров, наличие файлов с расширением .TMP и удалять их. Если пути поиска не заданы, вывести сообщение. Создать протокол, содержащий дату и имена удаляемых файлов

1.4 Создание командных файлов ОС UNIX

1.4.1 Краткие теоретические сведения.

Shell (Sh) – командный интерпретатор, используемый в операционных системах семейства UNIX, в котором пользователь может либо давать команды операционной системе по отдельности, либо запускать скрипты, состоящие из списка команд. **Shell** является также и языком программирования, который применяется для написания командных файлов (shell-файлов). Shell-файл содержит одну или несколько выполняемых команд (процедур), а имя файла в этом случае используется как имя команды.

Для создания сценариев можно использовать любой текстовый редактор.

1.4.2 Переменные командного интерпретатора.

Как любой язык программирования, командный язык **Shell** поддерживает

переменные. Тип их – строковый. Для обозначения переменных **Shell** используется последовательность букв, цифр и символов подчеркивания; переменные не могут начинаться с цифры.

Оператор присваивания выглядит так:

$имя_переменной=значение

Имя должно начинаться с буквы и может состоять из латинских букв, цифр, знака подчеркивания. Если значение переменной содержит специальные символы в имени файла, то при указании его имени в команде этот символ нужно экранировать знаком «\» (обратный слеш) или заключать все имя в двойные кавычки.

Ряд символов, которые имеют специальное значение для командного интерпретатора (их использование не рекомендуется):

~ ! @ # $ & % * ( ) [ ] { } ' " \ : ; > < пробел

Переменные **Shell** чувствительны к регистру. Например, **Myvar** и **myvar** – это имена разных переменных.

1.4.3 Позиционные переменные.

Переменные вида $n, где n - целое число, используются для идентификации позиций элементов в командной строке с помощью номеров, начиная с нуля. Например, в командной строке

cat text_1 text_2...text_9

аргументы идентифицируются параметрами **$1...$9**. Для имени команды всегда используется **$0**.

1.4.4 Вывод переменных.

Операция подстановки значения переменной обозначается символом **$**. Вывести значение переменной можно командой **echo**:

$ var=”Это моя переменная!”

$ echo var # выводит имя переменной var

$ echo $var # выводит значение переменной “Это моя переменная!”

Ставя перед именем переменной знак **$**, мы сообщаем интерпретатору, что нужно заменить ее значением. Это называется подстановкой переменной.

Для получения информации обо всех аргументах (включая последний) используют метасимвол *. Например:

echo $*

1.4.5 Арифметические операции.

Команда ***expr*** (express – выражать) вычисляет выражение expression и записывает результат в стандартный вывод. Элементы выражения разделяются пробелами; символы, имеющие специальный смысл в командном языке, нужно экранировать. Строки, содержащие специальные символы, заключают в апострофы. Используя команду ***expr***, можно выполнять сложение, вычитание, умножение, деление, взятие остатка, сопоставление символов и т. д.

Пример

Сложение, вычитание:

b=190

a=` expr 200 - $b`

где ` – обратная кавычка (левая верхняя клавиша). Умножение *, деление /, взятие остатка %:

d=` expr $a + 125 "*" 10`

c=` expr $d % 13`

Здесь знак умножения заключается в двойные кавычки, чтобы интерпретатор не воспринимал его как метасимвол. Во второй строке переменной ***c*** присваивается значение остатка от деления переменной ***d*** на 13.

1.4.6 Встроенные функции.

Встроенные команды являются частью интерпретатора и не требуют для своего выполнения проведения последовательного поиска файла команды и создания новых процессов. Перечень основных встроенных функций приведен в таблице А.1.

1.4.7 Команды для работы с данными.

echo [ключи] параметры

Копирует свои параметры на стандартный вывод (но с учетом специальных символов, если они имеются). Если не задан ключ –n, то в конце выдачи добавляется перевод строки.

more [файл]

Простая, но полезная команда, которая выводит файл-параметр (или, в его отсутствие, стандартный ввод) порциями, умещающимися на экране. Для вывода следующей порции нужно нажать клавишу «пробел».

less [файл]

Более современная команда просмотра файла (имя команды явно пародирует more). Позволяет, в частности, перемещаться по файлу вперед и назад. Для выдачи сводки по командам перемещения следует ввести h, для выхода из просмотра нужно ввести q.

wc [ключи] [файлы]

Для каждого параметра-файла (или для стандартного ввода) выдается строка, содержащая, в зависимости от ключа, число строк в файле (ключ -l), число слов (ключ -w) или число символов (ключ -c). По умолчанию (без ключей) выдаются все три числа.

head [ключи] [файл]

Выдает указанное число первых строк файла-параметра или стандартного ввода. По умолчанию выдаются 10 строк. Ключ –n число указывает иное число строк. Ключ –c размер указывает, что вместо определенного числа строк следует выдать указанное число начальных байтов.

tail [ключи] [файл]

Выдает на стандартный вывод несколько последних строк файла-параметра или стандартного ввода. По умолчанию выдаются 10 строк. Ключи **–n** число

и **–c** размер действуют так же, как для команды **head**. Если перед числом строк или байтов записан знак +, то соответствующее число указывает, сколько надо пропустить от начала файла, в противном случае – сколько нужно оставить в конце файла.

grep [ключи] образец [список_файлов]

Выполняет поиск заданной строки-образца в указанных файлах или в стандартном вводе. Если образец содержит пробелы, его следует заключить в кавычки. По умолчанию на стандартный вывод выдаются все строки, содержащие образец. Если проверяется несколько файлов, то перед выводимыми строками выдается имя файла.

При задании образца можно использовать регулярные выражения, задающие шаблон поиска строки. Синтаксис и семантика регулярных выражений напоминают использование подстановочных знаков при поиске в Microsoft Word. Основные символы, используемые при записи регулярных выражений, приведены в таблице 3.

Таблица 3 – Логические связки

Ключ	Значение
.	Любой символ
\w	Любая буква или цифра
\W	Любой символ, кроме букв и цифр
[символы]	Любой из перечисленных символов. Можно задавать диапазоны через знак - (например, [0-9] соответствует любой цифре)
[^символы]	Любой символ, кроме перечисленных. Например, [^A-Za-z] означает любой символ, кроме латинских букв
^	Начало строки
$	Конец строки
выраж*	Выражение присутствует ноль или более раз
выраж?	Выражение присутствует ноль или один раз
выраж+	Выражение присутствует один или более раз
выраж1выраж2	Последовательное соединение строк, соответствующих выражениям выраж1 и выраж2
выраж1|выраж2	Строка, соответствующая либо выраж1, либо выраж2
(…)	Используются для группировки выражений
\символ	Экранирует специальный символ, т. е. делает его обычным

Например, регулярное выражение '^A([0-9]+|[^0-9])B?' означает: «В начале строки должна стоять буква А, за которой может следовать либо одна или несколько цифр, либо ровно один символ, отличный от цифры. После этого должна следовать буква В». То же самое условие можно записать и проще: '^A.[0-9]*B?'.

Команда grep может иметь ряд ключей, некоторые из них приведены в таблице 4.

Таблица 4 – Ключи команды grep

Ключ	Значение
-c	Выдается только число подходящих строк, а не сами строки
-n	Перед каждой строкой выводится ее номер в файле
-i	Игнорируется различие строчных и прописных букв
-h	Отменяется выдача имен файлов
-v	Выдаются только строки, которые НЕ содержат образца
-r	Ищет во всех файлах указанного каталога и его подкаталогов

Команда возвращает код завершения 0, если удалось хотя бы раз найти искомый образец (если задан ключ -v, то наоборот, если не удалось).

sort [ключи] [-k от_поля [, до_поля]] [список_файлов]

Команда сортирует строки указанных файлов или стандартного ввода и выводит результат (сортированный файл) на стандартный вывод. Если указано несколько файлов, они объединяются перед сортировкой.

Ключи команды определяют способ сортировки. По умолчанию строки сортируются по возрастанию, как в словаре. Некоторые ключи приведены в таблице 5.

Таблица 5 – Ключи команды **sort**

Ключ	Значение
-b	Игнорируются пробелы и табуляции в начале строки
-f	Игнорируется различие между прописными и строчными буквами
-n	Поля рассматриваются как числа (возможно, со знаком и десятичной точкой) и сравниваются по числовому значению
-r	Сортировка ведется по убыванию
-t символ	Указанный символ рассматривается как разделитель полей (вместо пробела и табуляции)
-o файл	Результат записывается в указанный файл (вместо стандартного вывода)
-u	Из нескольких строк с одинаковыми значениями сравниваемых полей сохраняется только одна

Каждая строка рассматривается как набор полей, разделенных пробелами или символами табуляции. Параметр от_поля указывает, какое первое поле, считая от начала строки, следует учитывать при сравнении строк. Если параметр не задан, учитываются поля, начиная с первого. Параметр до_поля указывает, какое последнее поле должно участвовать в сравнении. Если параметр не задан, учитываются поля до конца строки.

Допускается указывать параметры от_поля и до_поля не в виде одного числа, а в виде пары чисел «m.n», где m – номер поля, а n – номер символа в поле. Как поля, так и символы в поле нумеруются, начиная с 1.

1.5 Пример выполнения задания

Разработать скрипт, выполняющий следующее: по заданному имени входа пользователя выдать, в зависимости от указанного ключа, либо его полное имя, либо идентификатор пользователя, либо идентификатор группы пользователя. Допускается задание сразу нескольких ключей.

Пусть скрипт носит имя userinfo, а допустимые ключи – n (полное имя), u (идентификатор пользователя) и g (идентификатор группы). Синтаксис вызова

userinfo [-nug] имя_входа

Пример

```
#!/bin/bash
info=
while getopts nug option ; do
 case $option in
 n) info=$info" `grep '^'$2 /etc/passwd | cut –f 5 –d :`"
 u) info=$info" `grep '^'$2 /etc/passwd | cut –f 3 –d :`"
 g) info=$info" `grep '^'$2 /etc/passwd | cut –f 4 –d :`"
 *) echo Bad option: $option
 esac
done
echo $info
```

Требуемая информация выбирается из системного файла /etc/passwd с помощью команды grep, при этом искомой строкой является имя входа ($2). Символ ^ перед именем заставляет искать имя только в начале строки. Этим гарантируется невозможность ложного сравнения в случае, когда имя пользователя случайно совпадает с частью какой-либо совсем другой строки файла passwd.

Используется вызов конвейера в обратных апострофах, чтобы собрать стандартный вывод в переменной info. При вырезании нужного поля из строки файла passwd учитывается, что разделителем полей в этом файле является двоеточие.

1.6 Задания для самостоятельного выполнения

По номеру варианта определить номера задач из таблицы 6, для решения которых разработать **shell**-программы.

Таблица 6 – Варианты заданий

Номер варианта	Задание
1	Вывод на экран списка параметров командной строки с указанием номера каждого параметра
2	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в алфавитном порядке и общего количества файлов

Продолжение таблицы 6

Номер варианта	Задание
3	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в порядке возрастания времени создания файла и общего количества файлов
4	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в порядке возрастания времени создания файла и общего количества файлов
5	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в порядке убывания времени обращения к файлу и общего количества файлов
6	Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в порядке возрастания времени изменения файла и общего количества файлов
7	Запрос и ввод имени каталога, переход в этот каталог, формирование файла с листингом каталога и возвращение в исходный каталог
8	Запрос и ввод имени пользователя, сравнение с текущим логическим именем пользователя и вывод сообщения: верно/неверно
9	Запрос и ввод числового идентификатора пользователя, сравнение с текущим идентификатором пользователя и вывод сообщения: верно/неверно
10	Запрос и ввод имени файла в текущем каталоге и количества дней. Вывод сообщения о файле: обновлялся/не обновлялся
11	Запрос и ввод имени файла (задается полный путь) и его типа, сравнение с действительным типом файла и вывод сообщения: совпадает/не совпадает
12	Циклическое чтение системного времени и выполнение очистки экрана в заданный момент
13	Циклическое чтение системного времени и переход в корневой каталог в заданный момент
14	Циклический просмотр списка файлов и выдача сообщения при появлении заданного имени в списке
15	Циклический просмотр списка каталогов и выдача сообщения при появлении заданного имени в списке
16	Циклическое чтение системного времени и в заданный момент создание каталога
17	Для каждого из файлов, перечисленных в списке параметров, создать отдельный подкаталог своего домашнего каталога и переместить туда файл. В случае, если нельзя выполнить перемещение (нельзя удалить файл), запрашивать пользователя, выполнять ли копирование или пропустить файл. Имена подкаталогов строить путем добавления к имени домашнего каталога чисел 1, 2, 3 и т. д.
18	Создать вручную «телефонный справочник», состоящий из нескольких записей, содержащих три поля: фамилия, адрес, номер телефона. Поля записи разделять знаком табуляции. Составить скрипт, который по заданной фамилии или адресу, или номеру телефона (в зависимости от указанного ключа) выдает значения двух других полей соответствующей записи
19	Провести копирование из одного каталога (источника) в другой каталог (приемник) всех файлов, имена которых удовлетворяют заданному шаблону. В зависимости от заданного ключа запрашивать подтверждение копирования либо для каждого файла, либо только в случае замены существующего файла, либо никогда

Окончание таблицы 6

Номер варианта	Задание
20	Выполнить в диалоге настройку поиска файла: запросить и ввести шаблон имени, начальный каталог поиска, тип файла, число дней после изменения файла или после обращения к нему. Выполнить поиск и вывести имена найденных файлов
21	Найти в указанном каталоге все файлы, содержащие заданную строку. Для каждого найденного файла запросить действие, которое необходимо выполнить: удалить файл, запретить доступ к нему прочих пользователей или оставить, как есть

2 Самостоятельная работа № 2

2.1 Синхронизация процессов и потоков при помощи событий и семафоров в ОС Windows

2.1.1 Краткие теоретические сведения.

В операционных системах Windows объектами синхронизации называются объекты ядра, которые могут находиться в одном из двух состояний: сигнальном (signaled) и несигнальном (nonsignaled). Объекты синхронизации могут быть разбиты на три класса. К первому классу относятся объекты синхронизации, которые служат только для решения проблемы синхронизации параллельных потоков. К таким объектам синхронизации в Windows относятся:

– мьютекс (mutex);

– событие (event);

– семафор (semaphore).

Ко второму классу объектов синхронизации относится ожидающий таймер (waitable timer). К третьему классу объектов синхронизации относятся объекты, которые переходят в сигнальное состояние по завершении своей работы или при получении некоторого сообщения. Примерами таких объектов синхронизации являются потоки и процессы. Пока эти объекты выполняются, они находятся в несигнальном состоянии. Если выполнение этих объектов заканчивается, то они переходят в сигнальное состояние.

2.1.2 Объекты-события в Windows.

Использование блокировки или монитора полезно для предотвращения одновременного выполнения блоков кода, но эти структуры не позволяют одному потоку передавать события в другой. Для этого требуются события синхронизации – объекты, применяющиеся для активации и приостановки потоков, которые могут находиться в одном из двух состояний: сигнальном (signaled) и несигнальном (nonsignaled). Потоки можно приостанавливать, заставляя их ожидать, если события находятся в несигнальном состоянии, и активировать, меняя состояние события на сигнальное. Если поток попытается ожидать события, которое уже находится в сигнальном состоянии, то выполнение потока продолжится

без задержки.

Существует два типа событий синхронизации: **AutoResetEvent** и **ManualResetEvent**. Отличие только одно: **AutoResetEvent** автоматически изменяется с сигнального состояния на несигнальное состояние всегда при активации потока. В отличие от него, **ManualResetEvent** вернется в несигнальное состояние только при вызове своего метода **Reset.**

Событие обоих типов устанавливается в сигнальное состояние при вызове метода **Set** этого события.

Потоки можно заставить дожидаться определенных событий, вызвав один из методов ожидания, например, **WaitOne**, **WaitAny** или **WaitAll**.

WaitHandle.WaitOne() приводит к ожиданию потока до тех пор, пока единственное событие не становится сигнализирующим,

WaitHandle.WaitAny() блокирует поток до тех пор, пока одно или несколько указанных событий не становятся сигнализирующими.

WaitHandle.WaitAll() блокирует поток до тех пор, пока все указанные события не становятся сигнализирующими.

2.1.3 Объекты-семафоры в Windows.

Семафор подобен мьютексу, за исключением того, что он предоставляет одновременный доступ к общему ресурсу не одному, а нескольким потокам. Поэтому семафор пригоден для синхронизации целого ряда ресурсов. Семафор управляет доступом к общему ресурсу, используя для этой цели счетчик. Если значение счетчика больше нуля, то доступ к ресурсу разрешен. А если это значение равно нулю, то доступ к ресурсу запрещен. С помощью счетчика ведется подсчет количества разрешений. Следовательно, для доступа к ресурсу поток должен получить разрешение от семафора.

Обычно поток, которому требуется доступ к общему ресурсу, пытается получить разрешение от семафора. Если значение счетчика семафора больше нуля, то поток получает разрешение, а счетчик семафора декрементируется. В противном случае поток блокируется до тех пор, пока не получит разрешение. Когда же потоку больше не требуется доступ к общему ресурсу, он высвобождает разрешение, а счетчик семафора инкрементируется. Если разрешения ожидает другой поток, то он получает его в этот момент. Количество одновременно разрешаемых доступов указывается при создании семафора. Так, если создать семафор, одновременно разрешающий только один доступ, то такой семафор будет действовать как мьютекс.

Семафоры особенно полезны в тех случаях, когда общий ресурс состоит из группы или пула ресурсов. Например, пул ресурсов может состоять из целого ряда сетевых соединений, каждое из которых служит для передачи данных. Поэтому потоку, которому требуется сетевое соединение, все равно, какое именно соединение он получит. В данном случае семафор обеспечивает удобный механизм управления доступом к сетевым соединениям.

Семафор реализуется в классе **System.Threading.Semaphore**, у которого имеется несколько конструкторов. Далее приведена простейшая форма конструктора данного класса:

public Semaphore(int initialCount, int maximumCount)

где **initialCount** – это первоначальное значение для счетчика разрешений семафора, т. е. количество первоначально доступных разрешений;

maximumCount – максимальное значение данного счетчика, т. е. максимальное количество разрешений, которые может дать семафор.

Семафор применяется таким же образом, как и описанный ранее мьютекс. В целях получения доступа к ресурсу в коде программы вызывается метод **WaitOne()** для семафора. Этот метод наследуется классом **Semaphore** от класса **WaitHandle**. Метод **WaitOne()** ожидает до тех пор, пока не будет получен семафор, для которого он вызывается. Таким образом, он блокирует выполнение вызывающего потока до тех пор, пока указанный семафор не предоставит разрешение на доступ к ресурсу.

Если коду больше не требуется владеть семафором, он освобождает его, вызывая метод **Release()**. Далее приведены две формы этого метода.

public int Release()

public int Release(int releaseCount)

В первой форме метод **Release()** высвобождает только одно разрешение, а во второй – количество разрешений, определяемых параметром **releaseCount**. В обеих формах данный метод возвращает подсчитанное количество разрешений, существовавших до высвобождения.

Метод **WaitOne()** допускается вызывать в потоке несколько раз перед вызовом метода **Release()**. Но количество вызовов метода **WaitOne()** должно быть равно количеству вызовов метода **Release()** перед высвобождением разрешения. С другой стороны, можно воспользоваться формой вызова метода **Release(int)**, чтобы передать количество высвобождаемых разрешений, равное количеству вызовов метода **WaitOne()**.

Далее приведен пример программы, в которой демонстрируется применение семафора. В этой программе семафор используется в классе **MyThread** для одновременного выполнения только двух потоков типа **MyThread**. Следовательно, разделяемым ресурсом в данном случае является процессор.

Пример

```
// Использовать семафор.
using System;
using System.Threading;
// Этот поток разрешает одновременное выполнение
// только двух своих экземпляров,
class MyThread
{
   public Thread Thrd;
   // Здесь создается семафор, дающий только два
   // разрешения из двух первоначально имеющихся,
   static Semaphore sem = new Semaphore(2, 2);
   public MyThread(string name)
   {
      Thrd = new Thread(this.Run);
```

```
        Thrd.Name = name;
        Thrd.Start();
    }
    // Точка входа в поток,
    void Run() {
Console .WriteLine (Thrd.Name + " ожидает разрешения.");
        sem.WaitOne ();
Console.WriteLine(Thrd.Name + " получает разрешение.");
for(char ch='A'; ch < 'D'; ch++)
{
Console.WriteLine (Thrd.Name + " : " + ch + " ");
    Thread.Sleep(500);
}
Console .WriteLine (Thrd. Name + " высвобождает разрешение.");
// Освободить семафор,
sem.Release () ;
    }
}
class SemaphoreDemo
{
    static void Main()
    {
        // Сконструировать три потока.
        MyThread mtl = new MyThread("Поток   #1");
        MyThread mt2 = new MyThread("Поток   #2");
        MyThread mt3 = new MyThread("Поток   #3");
        mtl.Thrd.Join(); mt2.Thrd.Join(); mt3.Thrd.Join();
    }
}
```

На рисунке 1 приведен пример выполнения программы.

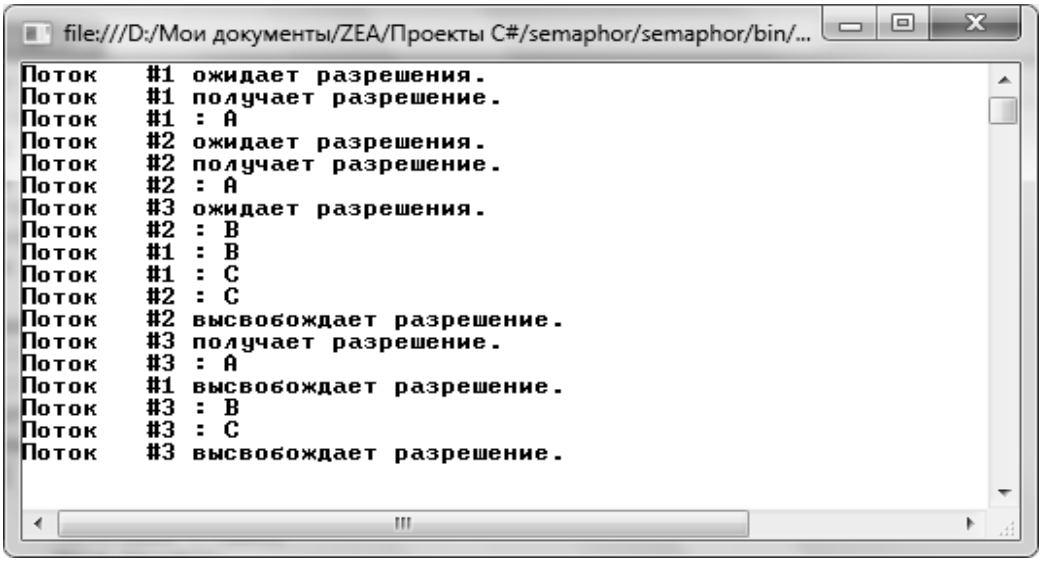


Рисунок 1 – Пример выполнения программы

В классе **MyThread** объявляется семафор **sem**. При этом создается семафор, способный дать не более двух разрешений на доступ к ресурсу из двух первоначально имеющихся разрешений.

Выполнение метода **MyThread.Run ()** не может быть продолжено до тех пор, пока семафор **sem** не даст соответствующее разрешение. Если разрешение отсутствует, то выполнение потока приостанавливается. Когда же разрешение появляется, выполнение потока возобновляется. В методе **Main()** создаются три потока. Но выполняться могут только два первых потока, а третий должен ожидать окончания одного из этих двух потоков. Далее приведен результат выполнения рассматриваемой здесь программы, хотя у Вас он может оказаться несколько иным.

Семафор, созданный в предыдущем примере, известен только тому процессу, который его породил. Но семафор можно создать и таким образом, чтобы он был известен где-нибудь еще. Для этого он должен быть именованным. Ниже приведены формы конструктора класса **Semaphore**, предназначенные для создания такого семафора.

public Semaphore(int initialCount, int maximumCount, string имя)

public Semaphore(int initialCount, int maximumCount, string имя, out bool createdNew)

В обеих формах конструктора имя обозначает конкретное имя, передаваемое конструктору. Если в первой форме семафор, на который указывает имя, еще не существует, то он создается с помощью значений, определяемых параметрами **initialCount** и **maximumCount**. А если он уже существует, то значения параметров **initialCount** и **maximumCount** игнорируются. После возврата из второй формы конструктора параметр **createdNew** будет иметь логическое значение **true**, если семафор был создан. В этом случае значения параметров **initialCount** и maximumCount используются для создания семафора. Если же параметр **createdNew** будет иметь логическое значение **false**, значит, семафор уже существует и значения параметров **initialCount** и **maximumCount** игнорируются. Существует и третья форма конструктора класса **Semaphore**, в которой допускается указывать управляющий доступом объект типа **SemaphoreSecurity**. С помощью именованных семафоров можно синхронизировать взаимодействие процессов.

2.1.4 Создание межпроцессных EventWaitHandle.

Конструктор **EventWaitHandle** позволяет создавать именованные **EventWaitHandle**, способные действовать через границы процессов. Имя задается обыкновенной строкой и может быть любым. Если задаваемое имя уже используется на компьютере, возвращается ссылка на существующий **EventWaitHandle**, в противном случае операционная система создает новый.

Пример

EventWaitHandle wh = new EventWaitHandle(false, EventResetMode.Auto, "MyCompany.MyApp.SomeName");

Исполнив этот код, два или более приложения получили бы возможность сигнализировать друг другу из любого потока всех процессов, которым известно имя события.

2.2 Пример выполнения задания

Рассмотрим пример взаимодействия процессов с использованием именованных событий, созданных с помощью конструктора **EventWaitHandle**. Одно приложение, названное СЕРВЕР, запускается первым, и его задача заключается в том, чтобы следить за работой второго приложения КЛИЕНТ. Приложение КЛИЕНТ позволяет вводить с помощью клавиатуры и отображать в своем окне произвольные символы. Каждый раз, когда пользователь вводит в окне приложения КЛИЕНТ символ "*" или "-", в окне контролирующего приложения СЕРВЕР отображается этот символ, ввод других символов игнорируется.

Пример приложения-сервера

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace Server
{
    class Program
    {
            static void Main(string[] args)
            {
//Создаем два события с ручным сбросом
EventWaitHandle starEvent = new EventWaitHandle(false, EventResetMode.ManualReset, "StarEvent");
EventWaitHandle hypEvent = new EventWaitHandle(false, EventResetMode.ManualReset, "HypEvent");

                    Console.WriteLine("Ожидание клиентов...");
                    while (true)
                    {
//Устанавливаем режим ожидания одного из двух событий
                        int eventIndex = WaitHandle.WaitAny(new WaitHandle[] { starEvent, hypEvent });
                        if (eventIndex == 0)
                        {
//При установке ожидаемого события в сигнальное состояние выводим //соответствующий символ и сбрасываем событие
                            Console.Write("*");
                            starEvent.Reset();
                        }
                        else if (eventIndex == 1)
                        {
                            Console.Write("-");
                            hypEvent.Reset();
                        }
```

```
                }
            }
    }
}
```

Пример приложения-клиента

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
namespace Client
{
    class Program
    {
            static void Main(string[] args)
            {
//Открываем созданные Сервером события
EventWaitHandle starEvent = EventWaitHandle.OpenExisting("StarEvent");
EventWaitHandle hypEvent = EventWaitHandle.OpenExisting("HypEvent");
Console.WriteLine("Введите * или - для передачи серверу");
                    while (true)
                    {
                        string message = Console.ReadLine();
                        if (message == "*")
                        {
//При нажатии * устанавливаем событие starEvent в сигнальное состояние
                            starEvent.Set();
                        }
                        else if (message == "-")
//При нажатии - устанавливаем событие hypEvent в сигнальное состояние
                        {
                            hypEvent.Set();
                        }
                    }
            }
    }
}
```

На рисунке 2 приведен пример выполнения программы.

Предположим, что нужно исполнять задачи в фоновом режиме без затрат на создание каждый раз нового потока для новой задачи. Этой цели можно достигнуть, используя единственный рабочий поток с постоянным циклом, ожидающим появление задачи. Получив задачу, он приступает к ее выполнению. После окончания выполнения поток снова переходит в режим ожидания. Это обычный многопоточный сценарий. Наряду с избавлением от накладных расходов на создание потоков, мы получаем последовательно исполняющиеся задачи, устраняя потенциальные проблемы взаимодействия между потоками и чрезмерное потребление ресурсов.

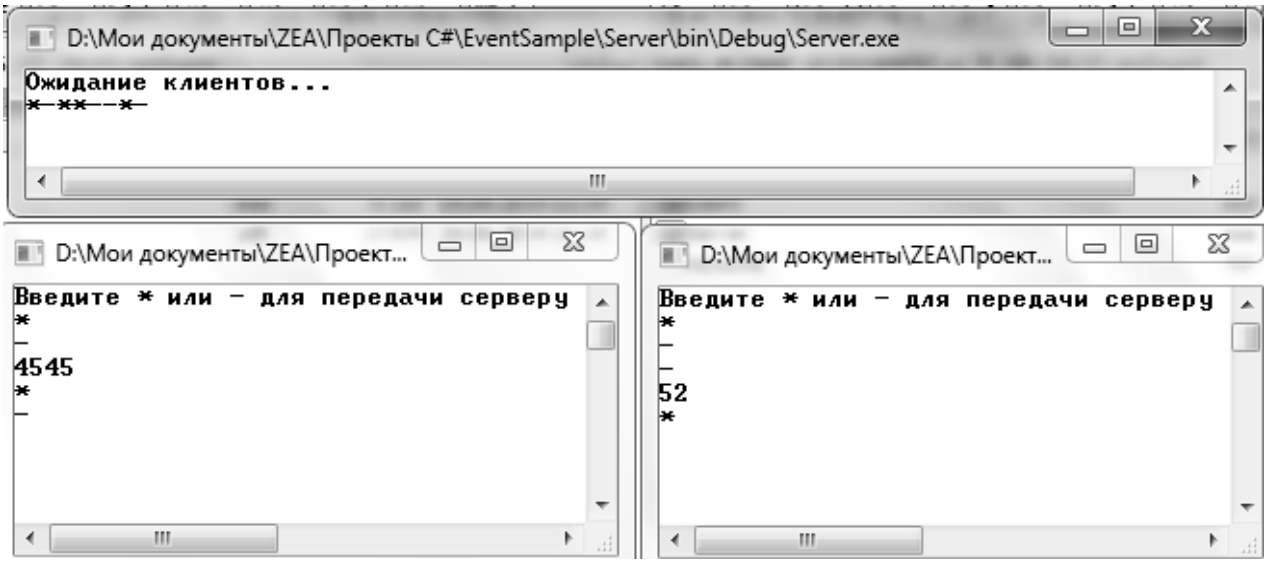


Рисунок 2 – Пример выполнения программы

2.3 Задания для самостоятельного выполнения

Разработайте приложения для решения задач в таблице 7.

Таблица 7 – Варианты заданий

Вариант	Задание
1	Написать программы для консольного процесса **Boss** (Резидент) и консольных процессов **Scout** (Шпион). Для моделирования передачи сообщений ввести специальные события, которые обозначают «точку» и «тире», конец сеанса. Процесс **Boss**: запрашивает у пользователя количество процессов **Scout**, которые он должен запустить; запускает заданное количество процессов **Scout**; принимает от каждого процесса **Scout** сообщение и выводит его на консоль в одной строке. Принимать сообщение может только от одного процесса, передача остальных сообщений от других процессов должна блокироваться с помощью семафора. Процесс **Scout**: запрашивает с консоли символы: «-», «.» (событие «тире», событие «точка») и передает соответствующие события процессу **Boss**; завершает свою работу, когда будет введён символ, обозначающий конец ввода сообщений. Каждый отдельный процесс открывать в **отдельном консольном окне**

Продолжение таблицы 7

Вариант	Задание
2	Написать программы для консольного процесса **Boss** (Резидент) и консольных процессов **Scout** (Шпион). Для моделирования передачи сообщений ввести специальные события, которые обозначают «1», «2» и конец сеанса для процессов **Scout**/ Процесс **Boss**: запрашивает у пользователя количество процессов **Scout**, которые он должен запустить; запускает заданное количество процессов **Scout**; принимает от каждого процесса **Scout** сообщение и выводит его на консоль в одной строке. Принимать сообщение может только от двух процессов, передача остальных сообщений от других процессов должна блокироваться с помощью семафора. Процесс **Scout**: запрашивает с консоли сообщения, состоящее из «1», «2», и передает их (по одному) процессу **Boss**; завершает свою работу. Каждый отдельный процесс открывать в **отдельном консольном окне**
3	Написать программы для консольного процесса **Boss** (Резидент) и консольных процессов **Scout** (Шпион). Для моделирования передачи сообщений ввести специальные события, которые обозначают любые четыре цифры. Процесс **Boss**: запрашивает у пользователя количество процессов **Scout**, которые он должен запустить; запускает заданное количество процессов **Scout**; принимает от каждого процесса **Scout** сообщение и выводит его на консоль в одной строке. Принимать сообщение может только от трёх процессов, передача остальных сообщений от других процессов должна блокироваться с помощью семафора; если приходит сообщение, с цифрой не из пароля, то выводит на консоль текст «ошибка». Процесс **Scout**: запрашивает с консоли сообщение, состоящее из цифр, и передает их (по одному) процессу **Boss**; завершает свою работу. Каждый отдельный процесс открывать в **отдельном консольном окне**
4	Написать программы для консольного процесса **Boss** и консольных процессов **Employee**. Для моделирования передачи сообщений ввести специальные события, которые «0», «1», «2», «3» и конец сеанса для процессов **Employee**. Процесс **Boss**: запрашивает у пользователя количество процессов **Employee**, которые он должен запустить; запускает заданное количество процессов **Employee**; принимает от каждого процесса **Employee** сообщение и выводит его на консоль в одной строке. Принимать сообщение может только от трёх процессов, передача остальных сообщений от других процессов должна блокироваться с помощью семафора. Процесс **Employee**: запрашивает с консоли сообщения, состоящее из «0», «1», «2», «3», конец сеанса работы и передает (по одному) его процессу **Boss**; завершает свою работу. Каждый отдельный процесс открывать в **отдельном консольном окне**

Продолжение таблицы 7

1	2
5	Написать программы для консольного процесса **Administrator** и процессов **Writer**. Для моделирования передачи сообщений ввести специальные события, которые обозначают сообщение «A», сообщение «B» и конец сеанса для процесса **Writer**. Одновременно принимать и отправлять сообщения может **только три процесса Writer**, передача остальных сообщений от других процессов должна блокироваться с помощью семафора; Процесс **Administrator**: запрашивает у пользователя количество процессов **Writer**, которые он должен запустить; запускает заданное количество процессов **Writer**; принимает сообщение от процесса **Writer**; выводит на консоль сообщение; принимает от каждого процесса **Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке; завершает свою работу. Процесс **Writer**: запрашивает с консоли сообщения и передает их (по одному) процессу **Administrator**; передает сообщение о завершении сеанса процессу **Administrator**; завершает свою работу. Каждый отдельный процесс открывать в **отдельном консольном окне**
6	Написать программы для консольного процесса **Boss** и консольных процессов **Employee**. Для моделирования передачи сообщений ввести специальные события, которые обозначают «1», «2», «3» и конец сеанса для процессов **Employee**. Процесс **Boss**: запрашивает у пользователя количество процессов **Employee**, которые он должен запустить; запускает заданное количество процессов **Employee**; принимает от каждого процесса **Employee** сообщение и выводит его на консоль в одной строке. Принимать сообщение может только от трёх процессов, передача остальных сообщений от других процессов должна блокироваться с помощью семафора; завершает свою работу. Процесс **Employee**: запрашивает с консоли сообщения, состоящее из «1», «2», «3», конец сеанса работы и передает (по одному) его процессу **Boss**; завершает свою работу. Каждый отдельный процесс открывать в **отдельном консольном окне**
7	Написать программы для консольного процесса **Administrator** и **процессов Writer**. Для моделирования передачи сообщений ввести специальные события, которые обозначают сообщение «+», сообщение «-», и конец сеанса для процесса **Writer**. Одновременно принимать и отправлять сообщения может **только два процесса Writer**, передача остальных сообщений от других процессов должна блокироваться с помощью семафора

Продолжение таблицы 7

1	2
	Процесс **Administrator**: запрашивает у пользователя количество процессов **Writer**, которые он должен запустить; запускает заданное количество процессов **Writer**; принимает сообщение от процесса **Writer**; выводит на консоль сообщение; принимает от каждого процесса **Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке; завершает свою работу. Процесс **Writer**: запрашивает с консоли сообщения и передает их (по одному) процессу **Administrator**; передает сообщение о завершении сеанса процессу **Administrator**; завершает свою работу. Каждый отдельный процесс открывать в **отдельном консольном окне**
8	Написать программы для консольного процесса **Boss** и консольных процессов **Parent**, **Child**. Для моделирования передачи сообщений ввести специальные события, которые обозначают любые две цифры и конец сеанса для процессов **Parent** и **Child**. Процесс **Boss**: запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить; запускает заданное количество процессов **Parent**, **Child** и передает каждому не более трех сообщений, передача остальных сообщений от других процессов должна блокироваться с помощью семафора; отправляет сообщения для процессов **Parent**, **Child**, завершает свою работу. Процесс **Parent**: получает сообщение от процесса **Boss** и выводит его на консоль; завершает свою работу. Процесс **Child**: получает сообщение, от процесса **Boss** и выводит его на консоль; завершает свою работу Каждый отдельный процесс открывать в **отдельном консольном окне**
9	Написать программы для консольного процесса **Administrator** и консольных процессов **Reader** и **Writer**. Для моделирования передачи сообщений ввести специальные события, которые обозначают сообщение «A», сообщение «B», и конец сеанса для процессов **Reader** и **Writer**. Одновременно принимать и отправлять сообщения могут только два процесса **Writer** и два процесса **Reader**, передача остальных сообщений от других процессов должна блокироваться с помощью семафоров. Процесс **Administrator**: запрашивает у пользователя количество процессов **Writer** (Reader); запускает заданное количество процессов Reader и **Writer**; принимает от каждого процесса **Writer** сообщение и выводит на консоль, затем отправляет его процессу **Reader**; принимает от каждого процесса **Reader** и **Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке; завершает свою работу

Окончание таблицы 7

1	2
	Процесс **Writer**: запрашивает с консоли сообщения, состоящее из «A», «B», и передает их (по одному) процессу **Administrator**; передает сообщение о завершении сеанса процессу **Administrator**; завершает свою работу. Процесс **Reader**: принимает сообщение от процесса **Administrator**; выводит на консоль сообщение; передает сообщение о завершении сеанса процессу **Administrator**; завершает свою работу. Каждый отдельный процесс открывать в **отдельном консольном окне**
10	Написать программы для консольного процесса **Boss** и консольных процессов **Parent**, **Child**. Для моделирования передачи сообщений ввести специальные события, которые обозначают «C», «D» и конец сеанса для процессов **Parent** и **Child**. Процесс **Boss**: запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить; запускает заданное количество процессов **Parent**, **Child**; принимает от каждого процесса **Parent**, **Child** сообщение и выводит на консоль в одной строке; не более четырех сообщений, передача остальных сообщений от других процессов должна блокироваться с помощью семафора; завершает свою работу. Процесс **Parent**: запрашивает с консоли сообщения, состоящее из «C» и передает их (по одному) процессу **Boss**; завершает свою работу. Процесс **Child**: запрашивает с консоли сообщения, состоящее из «D» и передает их (по одному) процессу **Boss**; завершает свою работу. Каждый отдельный процесс открывать в **отдельном консольном окне**

Список литературы

1 Операционные системы. Основы UNIX: учебное пособие / А. Б. Вавренюк [и др.]. – Москва : ИНФРА-М, 2018.

2 **Станек, У. Р.** Командная строка Microsoft Windows: справочник администратора / У. Р. Станек. – Москва: Русская редакция, 2012.

3 **Назаров, С. В.** Операционные системы. Практикум: учебное пособие / С. В. Назаров. – Москва: КНОРУС, 2016.

Приложение А
(обязательное)

Команды ОС LINUX

Таблица А.1 – Команды ОС LINUX

Функция	Описание
cd	Команда изменяет текущий каталог
pwd	Выводит название текущего рабочего каталога
SECONDS	Содержит время работы скрипта в секундах
USER	Содержит имя пользователя
UID	Числовой идентификатор текущего пользователя
set	Команда изменяет значения внутренних переменных сценария
unset	Команда удаляет переменную, фактически устанавливает ее значение в *null*
export	Команда экспортирует переменную, делая ее доступной дочерним процессам
declare, typeset	Команды *declare* и *typeset* задают и/или накладывают ограничения на переменные. declare -r делает переменную доступной только для чтения (аналог const в C)
Exit	Безусловное завершение работы сценария. Команде можно передать целое число, которое будет возвращено вызывающему процессу как код завершения. Вообще, считается хорошей практикой завершать работу сценария, за исключением простейших случаев, командой *exit 0*, чтобы проинформировать родительский процесс об успешном завершении
exec	Это встроенная команда интерпретатора **shell**, заменяет текущий процесс новым процессом, запускаемым командой *exec*. Обычно когда командный интерпретатор встречает эту команду, то он порождает дочерний процесс, чтобы исполнить команду. При использовании встроенной команды *exec* оболочка не порождает еще один процесс, а заменяет текущий процесс другим. Для сценария это означает его завершение сразу после исполнения команды *exec*. По этой причине, если встретится *exec* в сценарии, то, скорее всего, это будет последняя команда в сценарии
wait	Останавливает работу сценария до тех пор, пока не будут завершены все фоновые задания или пока не будет завершено задание/процесс с указанным номером задания. Возвращает код завершения указанного задания/процесса. Можно использовать команду *wait* для предотвращения преждевременного завершения сценария до того, как завершит работу фоновое задание
kill	Принудительное завершение процесса путем передачи ему соответствующего сигнала. **kill $$ # Сценарий завершает себя сам**
sh <filename.sh>	Вызов файла filename.sh

Продолжение таблицы А.1

Функция	Описание
trap [cmd] [cond]	Перехват сигналов прерывания, где cmd – выполняемая команда; cond=0 или EXIT – в этом случае команда cmd выполняется при завершении интерпретатора; cond=ERR – команда cmd выполняется при обнаружении ошибки; cond – символьное или числовое обозначение сигнала, в этом случае команда cmd выполняется при приходе этого сигнала
umask [-o | -s] [nnn]	Устанавливает маску создания файла (маску режимов доступа создаваемого файла, равную восьмеричному числу nnn: три восьмеричных цифры для пользователя, группы и других). Если аргумент nnn отсутствует, то команда сообщает текущее значение маски. При наличии флага -o маска выводится в восьмеричном виде, при наличии флага -s – в символьном представлении
ls[ключи] параметры	Для каждого параметра-каталога выдает информацию о файлах этого каталога. Для параметра-файла выдает информацию о данном файле. Если параметров нет, выдает информацию о текущем каталоге. Ключи: -l – подробный формат, по одному файлу в строке. Указываются: тип файла, права доступа для владельца, группы-владельца и прочих пользователей, количество жестких связей, владелец, группа-владелец, размер в байтах, дата и время последнего изменения, имя; -a – включаются данные об элементах каталога; -i – включается номер индексного дескриптора (inode); -d – для каталога выводятся данные о самом каталоге, а не о его содержимом; -t – сортировка по времени последнего изменения файла; -u – сортировка по времени последнего обращения к файлу; -c – сортировка по времени создания файла; -r – обратный порядок сортировки
who	Выводит список пользователей, работающих в данный момент в системе (в пределах домена локальной сети). Для каждого пользователя выводится одна строка, включающая его имя входа, имя терминала, время входа в систему и имя компьютера
history [-c] [число]	Выдает «список истории», содержащий последние введенные команды (по умолчанию – до 500 строк). С ключом -c очищает список истории. С аргументом – положительным числом N выдает только последние N строк истории
hostname [ключи]	Выдает на стандартный вывод имя данного компьютера. Чтобы получить полное интернетовское имя (Fully Qualified Domain Name), надо указать ключ -fqdn
times	Выдает суммарное процессорное время, затраченное в режиме ядра и в режиме задачи данным шеллом и всеми процессами, запускавшимися из шелла
ps [ключи]	Выдает информацию о процессах, выполняющихся в системе. По умолчанию выдает информацию только о процессах, запущенных с данного терминала, причем для каждого процесса выдаются четыре поля данных: идентификатор процесса PID, имя терминала TTY, затраченное процессорное время TIME, имя выполняемой команды CMD. Если процесс не прикреплен ни к какому терминалу, вместо имени терминала выдается знак ?.

Продолжение таблицы А.1

Функция	Описание
	При вызове с ключом -f дополнительно выдаются идентификатор пользователя UID, идентификатор процесса-родителя PPID, приоритет C, время запуска STIME, а для выполняемой команды указываются полное имя и параметры. При использовании ключа -e перечисляются все процессы, в том числе запущенные системой или другими пользователями. Имеется еще очень много ключей, позволяющих изменить объем и формат выдаваемой информации о процессах
uname [ключи]	Выводит строку информации о системе, согласно следующим ключам: -s – имя ОС; -n – сетевое имя компьютера; -v – версия ОС; -r – номер выпуска ОС; -m – тип компьютера; -a – вся перечисленная выше информация; Отсутствие ключей эквивалентно заданию ключа **–s**
date [ключи] [новая_дата]	Выдает системную дату и время либо устанавливает новые дату и время. Ключи позволяют изменить формат выдачи. Новая дата и время задаются в формате ММДДЧЧмм[[СС]ГГ][.сс] (месяц, день, часы, минуты, столетие, год, секунды)
cal [ключи] [месяц [год]]	Выдает табель-календарь на указанный месяц. С ключом –y выдает календарь на весь год. При заданном ключе –m первым днем недели считается понедельник (по умолчанию неделя начинается с воскресенья)
du [ключи] [список_имен]	Выдает суммарные данные об использовании дискового пространства файлами, содержащимися в указанных каталогах и их подкаталогах. Некоторые ключи приведены в следующей таблице: -a – выдает данные не только о каталогах, но и о каждом файле; -b – выдает объем дисковой памяти в байтах (по умолчанию выдается число блоков); -c – формирует общую сумму; -S – не включает в размер каталога размеры его подкаталогов; -s – для каждого параметра-каталога выдается только общая сумма, без подкаталогов
stat [список_имен]	Выдает информацию о файле, содержащуюся в дескрипторе файла inode. Формат выдачи ориентирован скорее на восприятие человеком, чем на дальнейшую обработку (информация выдается в несколько строк, с названиями полей)
finger [ключи] [список_имен]	Выдает доступную информацию о перечисленных в списке пользователях системы. Если список не задан, выдает информацию обо всех пользователях, работающих в системе в данный момент. Если задан ключ -s, выдает минимальный набор информации в одной строке. Ключ -l задает подробный многострочный формат информации. Если ключи не заданы, то при заданном списке имен используется формат -l, а если список не задан, то формат -s
cd [путь] или chdir [путь]	Устанавливает каталог по указанному пути в качестве текущего. Если путь не указан, использует «домашний» каталог пользователя, полное имя которого хранится в переменной HOME

Продолжение таблицы А.1

Функция	Описание
cp файл1 файл2 или cp файлы каталог	В первой форме копирует параметр файл1 в файл2. Во второй форме копирует один или несколько файлов в указанный каталог
ln [-s] новое_имя файл	Без ключа создает жесткую связь с файлом, т. е. дает существующему файлу дополнительное имя в том же или в другом каталоге. Имя может содержать путь к каталогу. Счетчик связей файла увеличивается на 1. С ключом -s создает символическую связь, т. е. новый файл, содержащий полное имя существующего файла (аналог ярлыка Windows). Счетчик связей файла при этом не увеличивается
mv файл1 файл2 или mv список_файлов каталог	В первой форме переименовывает файл1 в файл2 (или перемещает в другой каталог). Во второй форме перемещает один или несколько файлов в указанный каталог
rename старое_имя новое_имя	Переименовывает файл или каталог
rm [ключи] файлы	Удаляет указанные файлы, точнее – удаляет из каталогов имена и уменьшает на 1 счетчики связей файлов. Действительному удалению подлежат только файлы, для которых число связей стало равным 0 (т. е. удалены все имена файла). Ключ -r позволяет удалять целые каталоги вместе с их содержимым. Ключ -i требует от системы задавать для каждого файла вопрос о необходимости его удаления
mkdir список_каталогов	Создает один или несколько пустых каталогов с заданными именами. Каждый пустой каталог содержит, тем не менее, два имени: имя. описывает сам каталог, а имя .. описывает родительский каталог
find список_каталогов [ключи]	Служит для поиска файлов с известным именем и/или другими атрибутами в дереве файловой системы. Заданный каталог или несколько каталогов определяют части файловой системы, в которых ведется поиск. Выполняется просмотр подкаталогов всех уровней, начиная с заданного каталога. Ключи определяют условия поиска файлов и действия с найденными файлами. Некоторые ключи приведены в таблице. Если задано более одного условия, проверяется истинность всех (конъюнкция). В отличие от большинства других команд, ключи задаются не одной буквой, а целым словом
chmod права список_файлов	Позволяет изменить права доступа к заданным файлам (или каталогам). Изменяемые права могут быть заданы двумя способами: либо в символьном виде, либо с помощью трех восьмеричных цифр. Символьное задание прав состоит из трех элементов: категория пользователей, для которой задаются права (u – владелец файла, g – группа-владелец, o – прочие пользователи, a – все пользователи), выполняемая операция (+ – добавить право, - – отменить право, = – присвоить только это право, отменив остальные права) и конкретное право (r – чтение, w – запись, x – выполнение). Можно указать несколько категорий пользователей и несколько операций с разными правами для одной категории. Можно также в одной команде задать разные права для разных категорий пользователей, разделив их запятыми. Например, запись ug+r-w,o=x означает: «Для владельца и группы разрешить чтение и запретить запись (право на выполнение не менять), для остальных пользователей разрешить выполнение, запретить чтение и запись»

Окончание таблицы А.1

Функция	Описание
	Второй способ задания прав предполагает явное задание всех прав в виде восьмеричного числа из трех цифр. Первая цифра задает три бита прав для владельца, вторая цифра – для группы, третья – для прочих. Например, число 751 означает набор прав, который команда ls -l отобразила бы в виде rwxr-x--x, т. е. все права для владельца, чтение и выполнение для группы, только выполнение для прочих
name имя_файла	Истина, если имя файла (без пути) совпадает с заданным
atime число_дней	Истина, если к файлу были обращения за последние дни
mtime число_дней	Истина, если файл был изменен за последние дни
newer файл	Истина, если файл «новее», чем указанный файл, т. е. был изменен позднее
type символ	Истина для всех файлов указанного типа (f – обычный файл, d – каталог, b – блочное устройство, c – символьное устройство, p – именованный канал, s – символическая связь)
group имя_группы	Истина, если имя группы-владельца совпадает с заданным
print	Не проверяет никаких условий, а лишь указывает, что нужно выдать на стандартный вывод полные имена найденных файлов
mkdir [ключи] каталог	Создает каталог с указанным именем. Если ключи не заданы, то требуется, чтобы уже существовал родительский каталог. При заданном ключе –p команда может создать сразу несколько вложенных каталогов
rmdir [ключи] список_каталогов	Удаляет перечисленные каталоги. Удаляемый каталог должен быть пустым. При заданном ключе –p команда будет также удалять родительские каталоги, если они становятся пустыми. Для удаления непустого каталога вместе с его файлами можно использовать команду rm -r
cat список_файлов	Читает файлы-параметры и копирует их содержимое на стандартный вывод. Если параметры не заданы, просто передает стандартный ввод на стандартный вывод