

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

Ход работы:

1. Запустить интерпретатор командной строки.
2. Увеличить размер окна интерпретатора и задать цвет фона и цвет шрифта (рекомендуется синий фон и белый шрифт).
3. Создать список фамилий студентов группы. Отсортировать список в алфавитном порядке и сохранить его в новом файле.
4. Создать текстовый файл, содержащий справочные сведения по командам DIR, COPY и XCOPY.
5. Вывести содержимое указанного в табл.1.1 каталога по указанному формату на экран и в файл.
6. Скопировать все имеющиеся в каталоге Windows растровые графические файлы в каталог WinGrafika на диске C:. Если диск C: недоступен, использовать любой другой доступный диск.
7. Скопировать все имеющиеся в каталоге Windows исполняемые файлы в каталог WinEx на диске C:. Если диск C: недоступен, использовать любой другой доступный диск.

Таблица 1.1

Имя каталога	Что выводить	Сортировать по	Атрибуты файлов и каталогов
%Windows%	Только подкаталоги	Именам	Только чтение

Описание выполнения работы

1. Запустим интерпретатор командной строки. Вид окна интерпретатора командной строки представлен на рисунке 1.1.



Рисунок 1.1 – Командное окно интерпретатора

2. Откроем свойства командного окна интерпретатора и увеличим размер окна (рисунок 1.2).

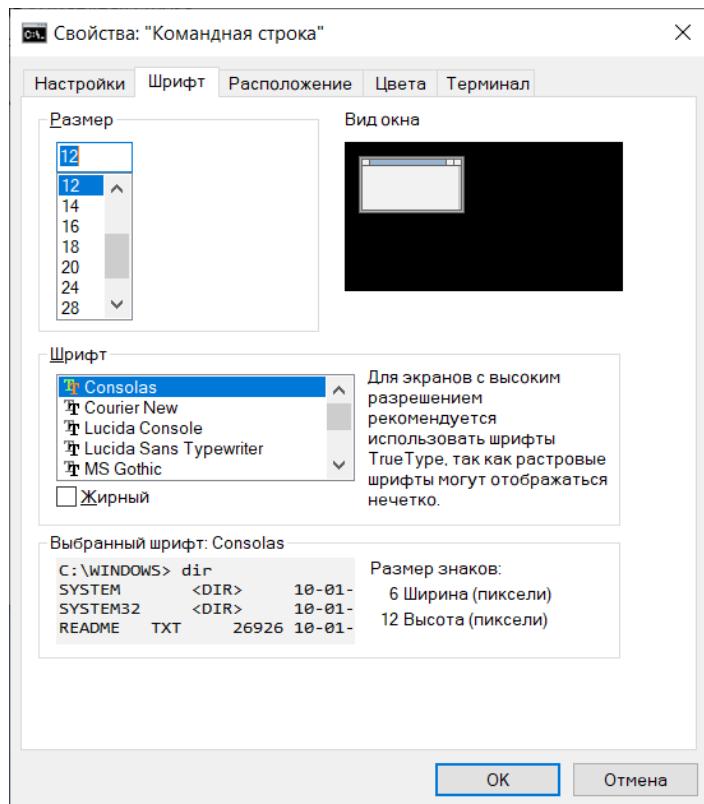


Рисунок 1.2 – Изменение размера окна интерпретатора

Изменим цвет фона окна интерпретатора на бежевый (рисунок 1.3).

Изменим цвет шрифта окна интерпретатора на коричневый (рисунок 1.4).

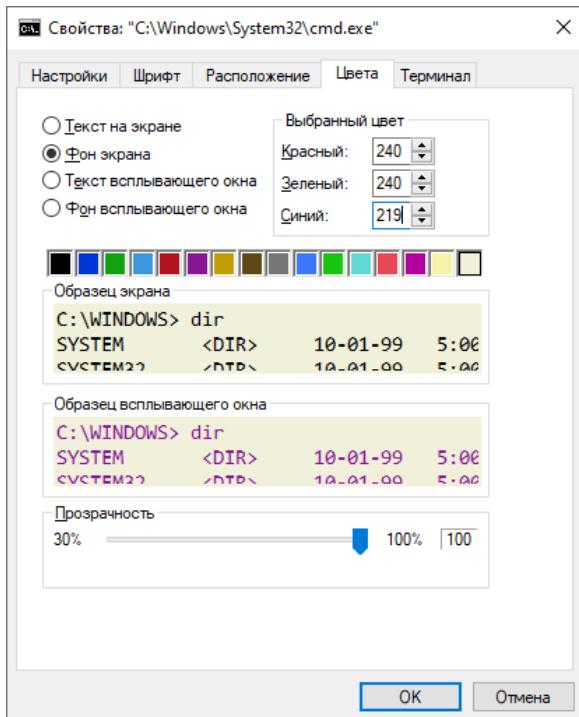


Рисунок 1.3 – Изменение цвета фона окна интерпретатора

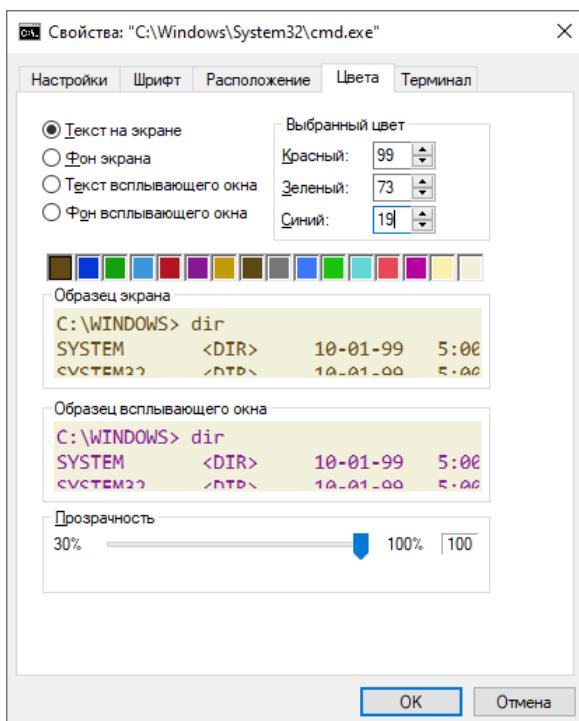


Рисунок 1.4 – Изменение цвета шрифта окна интерпретатора

3. Создадим список студентов группы и запишем его в файл с помощью команды (рисунок 1.5):
copy con group.txt

Содержимое полученного файла представлено на рисунке 1.6.

```
D:\GH\university\term6\OS\lab1>copy con group.txt
тюрин
якунчев
пиняйкина
назаров
терешкина
пименов
морозов
максимова
ошина
кудашкина
кручинкин
минин
жуков
ежиков
косолапов
егорова
конышев
бочков
бурлакова
арискин
аэаркина
^Z
Скопировано файлов: 1.
```

Рисунок 1.5 – Создание списка студентов группы

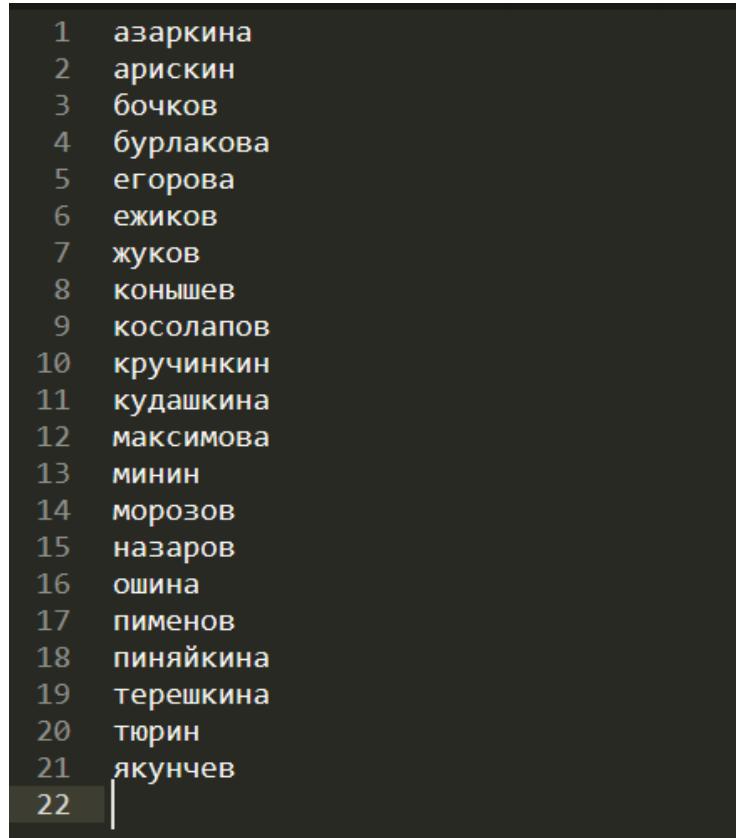
```
1  тюрин
2  якунчев
3  пиняйкина
4  назаров
5  терешкина
6  пименов
7  морозов
8  максимова
9  ошина
10  кудашкина
11  кручинкин
12  минин
13  жуков
14  ежиков
15  косолапов
16  егорова
17  конышев
18  бочков
19  бурлакова
20  арискин
21  аэаркина|
```

Рисунок 1.6 –Содержимое файла group.txt

Отсортируем список студентов по алфавиту с помощью команды sort и направим вывод результата работы данной команды в файл sortgroup.txt:

```
sort < group.txt > sortgroup.txt
```

Содержимое файла sortgroup.txt, в котором хранится отсортированный список студентов, представлено на рисунке 1.7.



```
1 азаркина
2 арискин
3 бочков
4 бурлакова
5 егорова
6 ежиков
7 жуков
8 конышев
9 косолапов
10 кручинкин
11 кудашкина
12 максимова
13 минин
14 морозов
15 назаров
16 ошина
17 пименов
18 пиняйкина
19 терешкина
20 тюрин
21 якунчев
22 |
```

Рисунок 1.7 – Содержимое файла с результатом работы команды SORT

4. Создадим текстовый файл, содержащий справочные сведения по командам DIR, COPY и XCOPY с помощью команды

```
(DIR /? || COPY /? || XCOPY /?) > info.txt
```

Результат работы команды представлен на рисунках 1.8-1.10.

```

1 Вывод списка файлов и подкаталогов в указанном каталоге.
2
3 DIR [drive:] [path] [filename] [/A[[:]attributes]] [/B] [/C] [/D] [/L] [/N]
4   [/O[[:]sortorder]] [/P] [/Q] [/R] [/S] [/T[[:]timefield]] [/W] [/X] [/4]
5
6 [drive:] [path] [filename]
7   | Диск, каталог или имена файлов для включения в список.
8
9 /A Отображение файлов с указанными атрибутами.
10 атрибуты D Каталоги.          R Файлы, доступные только для чтения
11           H Скрытые файлы      A Файлы, готовые для архивирования
12           S Системные файлы    I Файлы с неиндексированным содержимым
13           L Точки повторной обработки O Автономные файлы
14           - Префикс "--" имеет значение НЕ
15 /B Вывод только имен файлов.
16 /C Применение разделителя групп разрядов при выводе размеров файлов.
17   | Используется по умолчанию. Чтобы отключить применение разделителя групп разрядов, задайте ключ /-C.
18 /D Вывод списка в нескольких столбцах с сортировкой по столбцам.
19 /L Использовать нижний регистр.
20 /N Новый формат длинного списка, имена файлов выводятся в крайнем правом столбце.
21 /O Сортировка списка отображаемых файлов.
22 sortorder N По имени (по алфавиту)      S По размеру (начиная с минимального)
23           E По расширению (по алфавиту)  D По дате и времени (начиная с самого старого)
24           G Начать список с каталогов - Префикс "--" обращает порядок
25 /P Пауза после заполнения каждого экрана.
26 /Q Вывод сведений о владельце файла.
27 /R Отображение альтернативных потоков данных этого файла.
28 /S Отображение файлов из указанного каталога и всех его подкаталогов.
29 /T Выбор поля времени для сортировки.
30 timefield C Создание
31           A Последнее использование
32           W Последнее изменение
33 /W Вывод списка в несколько столбцов.
34 /X Отображение коротких имен для файлов, чьи имена не соответствуют стандарту 8.3.
35   | Формат аналогичен выводу с ключом /N, но короткие
36   | имена файлов выводятся слева от длинных. Если короткого имени у
37   | файла нет, вместо него выводятся пробелы.
38 /4 Вывод номера года в четырехзначном формате
39
40 Стандартный набор ключей можно записать в переменную среды DIRCMD. Для отмены
41 их действия введите в команде те же ключи с префиксом "--", например: /-W.
42 Копирование одного или нескольких файлов в другое место.
43

```

Рисунок 1.8 – Фрагмент файла info.txt, содержащий сведения о команде DIR

```

44 COPY [/D] [/V] [/N] [/Y | /-Y] [/Z] [/L] [/A | /B] источник [/A | /B]
45   | [+ источник [/A | /B] [+ ...]] [результат [/A | /B]]
46
47 источник Имена одного или нескольких копируемых файлов.
48   | /A Файл является текстовым файлом ASCII.
49   | /B Файл является двоичным файлом.
50   | /D Указывает на возможность создания зашифрованного файла
51 результат Каталог и/или имя для конечных файлов.
52   | /V Проверка правильности копирования файлов.
53   | /N Использование, если возможно, коротких имен при копировании
54   | файлов, чьи имена не удовлетворяют стандарту 8.3.
55   | /Y Подавление запроса подтверждения на перезапись существующего
56   | конечного файла.
57   | /-Y Обязательный запрос подтверждения на перезапись существующего
58   | конечного файла.
59   | /Z Копирование сетевых файлов с возобновлением.
60   | /L Если источник является символьской ссылкой, копирование
61   | ссылки вместо реального файла, на который указывает ссылка.
62
63 Ключ /Y можно установить через переменную среды COPYCMD.
64 Ключ /-Y командной строки переопределяет такую установку.
65 По умолчанию требуется подтверждение, если только команда COPY
66 не выполняется в пакетном файле.
67
68 Чтобы объединить файлы, укажите один конечный и несколько исходных файлов,
69 используя подстановочные знаки или формат "файл1+файл2+файл3+...".
70 Копирует файлы и деревья папок.
71

```

Рисунок 1.9 – Фрагмент файла info.txt, содержащий сведения
о команде COPY

```

XCOPY источник [destination] [/A | /M] [/D[:date]] [/P] [/S [/E]] [/V] [/W]
[ /C ] [ /I ] [ /Q ] [ /F ] [ /L ] [ /G ] [ /H ] [ /R ] [ /T ] [ /U ]
[ /K ] [ /N ] [ /O ] [ /X ] [ /Y ] [ /-Y ] [ /Z ] [ /B ] [ /J ]
[ /EXCLUDE:file1[+file2][+file3]... ] [ /COMPRESS ]

source      Копируемые файлы.
destination  Расположение или имена новых файлов.
/A          Копирует только файлы с установленным атрибутом архивации;
           сам атрибут при этом не изменяется.
/M          Копирует только файлы с установленным атрибутом архивации;
           после копирования атрибут снимается.
/D:m-d-y   Копирует файлы, измененные не ранее указанной даты.
           Если дата не указана, заменяются только конечные файлы
           с более ранней датой, чем у исходных файлов.
/EXCLUDE:file1[+file2][+file3]...
           Список файлов, содержащих строки. Каждая строка
           должна располагаться в отдельной строке в файлах. Если какая-либо
           из строк совпадает с любой частью абсолютного пути к копируемому
           файлу, такой файл исключается из операции копирования. Например,
           при указании строки \obj\ или .obj будут исключены
           все файлы из каталога obj или все файлы с расширением
           OBJ соответственно.
/P          Выводит запросы перед созданием каждого конечного файла.
/S          Копирует только пустые каталоги с подкаталогами.
/E          Копирует каталоги с подкаталогами, включая пустые.
           Эквивалент сочетания параметров /S /E. Совместим с параметром /T.
/V          Проверяет размер каждого нового файла.
/W          Выводит запрос на нажатие клавиши перед копированием.
/C          Продолжает копирование вне зависимости от наличия ошибок.
/I          Если назначение не существует и копируется несколько файлов,
           считается, что местом назначения является каталог.
/Q          Запрещает вывод имен копируемых файлов.
/F          Выводит полные имена исходных и конечных файлов во время копирования.
/L          Выводит копируемые файлы.
/G          Копирует зашифрованные файлы в конечную папку,
           не поддерживающую шифрование.
/H          Копирует скрытые и системные файлы (среди прочих).
/R          Разрешает замену файлов, предназначенных только для чтения.
/T          Создает структуру каталогов (кроме пустых каталогов)
           без копирования файлов. Для создания пустых каталогов и подкаталогов
           используйте сочетание параметров /T /E.
/U          Копирует только файлы, уже имеющиеся в конечной папке.
/K          Копирует атрибуты. При использовании команды XCOPY обычно сбрасываются атрибуты "только для чтения".
/N          Использует короткие имена при копировании.
/O          Копирует сведения о владельце и данные ACL.
/X          Копирует параметры аудита файлов (требуется параметр /O).
/Y          Подавляет запрос на подтверждение перезаписи
           существующего конечного файла.
/-Y         Обязательный запрос на подтверждение перезаписи
           существующего конечного файла.
/Z          Копирует сетевые файлы с возобновлением.
/B          Копирует символьную ссылку вместо ее целевого объекта.
/J          Копирует с использованием ввода-вывода без буферизации. Рекомендуется для очень больших файлов.
/COMPRESS   Запрос на сетевое сжатие во время передачи файла, если
           применено.

Параметр /Y можно установить заранее через переменную среды COPYCMD.
Параметр /-Y командной строки переопределяет такую установку.

```

Рисунок 1.10 – Фрагмент файла info.txt, содержащий сведения

о команде XCOPY

5. Выведем содержимое указанного в таблице 1.1 каталога по указанному формату на экран и в файл.

Вывод на экран отсортированных по имени каталогов, доступных только для чтения, из каталога Windows выполняется с помощью команды

DIR C:\Windows /A:RD /O:N

где ключ /A:RD означает вывод файлов, доступных для чтения, ключ /O:N – сортировку выводимых каталогов по именам.

Результат работы данной команды представлен на рисунке 1.11.

Содержимое папки C:\Windows				
19.02.2023	17:23	<DIR>	assembly	
19.02.2023	13:31	<DIR>	Fonts	
19.02.2023	13:31	<DIR>	ImmersiveControlPanel	
07.12.2019	12:31	<DIR>	Media	
19.02.2023	17:23	<DIR>	Microsoft.NET	
07.12.2019	12:14	<DIR>	Offline Web Pages	
04.11.2022	23:33	<DIR>	PrintDialog	
		0 файлов	0 байт	
		7 папок	36 609 191 936 байт свободно	

Рисунок 1.11 – Вывод каталогов папки Windows, доступных только для чтения, отсортированных по именам

Для вывода результатов работы данной команды в файл dir.txt дополним её следующим образом:

```
C:\Windows>dir /A:DR /O:N > D:\GH\university\term6\OS\lab1\dir.txt
```

1	Том в устройстве С не имеет метки.
2	Серийный номер тома: C464-FE5F
3	
4	Содержимое папки C:\Windows
5	
6	19.02.2023 17:23 <DIR> assembly
7	19.02.2023 13:31 <DIR> Fonts
8	19.02.2023 13:31 <DIR> ImmersiveControlPanel
9	07.12.2019 12:31 <DIR> Media
10	19.02.2023 17:23 <DIR> Microsoft.NET
11	07.12.2019 12:14 <DIR> Offline Web Pages
12	04.11.2022 23:33 <DIR> PrintDialog
13	0 файлов 0 байт
14	7 папок 36 608 450 560 байт свободно
15	

Рисунок 1.12 – Содержимое файла dir.txt

6. Скопируем все имеющиеся в каталоге Windows растровые графические файлы в каталог WinGrafika на диске С.

Для этого выполним следующую команду из каталога C:\windows:

```
xcopy *.png C:\Users\TEMP.LAB227.009\Downloads\WinGrafica & xcopy
*.jpg C:\Users\TEMP.LAB227.009\Downloads\WinGrafica & xcopy *.bmp
C:\Users\TEMP.LAB227.009\Downloads\WinGrafica
```

Результат выполнения данной команды представлен на рисунке 1.13.

```
C:\>xcopy *.png C:\Users\TEMP.LAB227.009\Downloads\WinGrafica & xcopy *.jpg  
C:\Users\TEMP.LAB227.009\Downloads\WinGrafica & xcopy *.bmp C:\Users\TEMP.  
LAB227.009\Downloads\WinGrafica  
Что означает C:\Users\TEMP.LAB227.009\Downloads\WinGrafica:  
имя файла или каталога  
(F = файл, D = каталог)? d  
Не найден файл: *.png  
Скопировано файлов: 0.  
Что означает C:\Users\TEMP.LAB227.009\Downloads\WinGrafica:  
имя файла или каталога  
(F = файл, D = каталог)? d  
Не найден файл: *.jpg  
Скопировано файлов: 0.  
Что означает C:\Users\TEMP.LAB227.009\Downloads\WinGrafica:  
имя файла или каталога  
(F = файл, D = каталог)? d  
Не найден файл: *.bmp  
Скопировано файлов: 0.
```

Рисунок 1.13 – Копирование графических файлов

7. Скопируем все имеющиеся в каталоге Windows исполняемые файлы в каталог WinEx на диске С.

Для этого выполним следующую команду:

```
xcopy *.exe C:\Users\zzz\Downloads\WinEx
```

Результат выполнения данной команды представлен на рисунке 1.14.

```
C:\Windows>xcopy *.exe C:\Users\zzz\Downloads\WinEx  
Что означает C:\Users\zzz\Downloads\WinEx:  
имя файла или каталога  
(F = файл, D = каталог)? d  
C:bfsvc.exe  
C:explorer.exe  
C:HelpPane.exe  
C:hh.exe  
C:notepad.exe  
C:py.exe  
C:pyw.exe  
C:regedit.exe  
C:splwow64.exe  
C:winhlp32.exe  
C:write.exe  
Скопировано файлов: 11.
```

Рисунок 1.14 – Копирование исполняемых файлов

Контрольные вопросы и ответы на них

1. Достоинства и недостатки интерфейса командной строки.

Достоинства:

Наиболее кроссплатформенный интерфейс. Интерфейс командной строки поддерживается любой ОС. Возможно встроить интерфейс командной строки в любое другое ПО.

Недостатки:

В терминале (командной строке) нет возможности отображать графическую информацию.

2. Инструменты командной строки для автоматизации работы в ОС Microsoft Windows.

Оболочка командной строки cmd.exe, среда выполнения сценариев Windows Script Host и оболочка Microsoft PowerShell.

3. Настраиваемые свойства интерпретатора.

У утилиты командной строки, которая поставляется в виде стандартного приложения ОС Windows, имеется свой набор опций и параметров настройки. В окне свойств будут доступны четыре вкладки с опциями: общие, шрифт, расположение и цвета.

4. Различие между внутренними и внешними командами. Примеры внешних и внутренних команд.

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

5. Структура команды интерпретатора.

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

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

6. Получение информации о конкретной команде.

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

7. Групповые символы (шаблоны) и их использование.

Используя символ амперсанда &, можно разделить несколько утилит в одной командной строке, при этом они будут выполняться друг за другом.

Условная обработка команд в Windows осуществляется с помощью символов && и || следующим образом. Двойной амперсанд && запускает команду, стоящую за ним в командной строке, только в том случае, если команда, стоящая перед амперсандами была выполнена успешно.

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.

Условная обработка действует только на ближайшую команду.

Несколько утилит можно сгруппировать в командной строке с помощью круглых скобок.

8. Перенаправление ввода/вывода и конвейеризация команд.

Для того, чтобы перенаправить текстовые сообщения, выводимые какой-либо командой, в текстовый файл, нужно использовать конструкцию команда > имя_файла.

Если при этом заданный для вывода файл уже существовал, то он перезаписывается, если не существовал — создается. Можно также не

создавать файл заново, а дописывать информацию, выводимую командой, в конец существующего файла. Для этого команда перенаправления вывода должна быть задана так: команда >> имя_файла.

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

9. Условное выполнение и группировка команд.

Условная обработка команд в Windows осуществляется с помощью символов && и || следующим образом. Двойной амперсанд && запускает команду, стоящую за ним в командной строке, только в том случае, если команда, стоящая перед амперсандами была выполнена успешно.

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.

Условная обработка действует только на ближайшую команду.

Несколько утилит можно сгруппировать в командной строке с помощью круглых скобок.

10. Назначение символов &, &&, || и () .

Используя символ амперсанда &, можно разделить несколько утилит в одной командной строке, при этом они будут выполняться друг за другом.

Двойной амперсанд && запускает команду, стоящую за ним в командной строке, только в том случае, если команда, стоящая перед амперсандами была выполнена успешно.

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.

Несколько утилит можно сгруппировать в командной строке с помощью круглых скобок.

11. Команды для работы с файловой системой – названия и возможности.

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

Для копирования одного или нескольких файлов используется команда COPY.

Команда XCOPY используется для копирования файлов и каталогов с сохранением их структуры. По сравнению с командой COPY имеет более широкие возможности и является наиболее гибким средством копирования в командной строке Windows.

Команда: DIR [диск:][путь][имя_файла][ключи] используется для вывода информации о содержимом дисков и каталогов.

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).

Удалить один или несколько файлов можно с помощью команды DEL [диск:][путь]имя_файла [ключи].

Переименовать файлы и каталоги можно с помощью команды RENAME (REN).

Команда для перемещения одного или более файлов имеет вид: MOVE [/Y|/-Y] [диск:][путь]имя_файла1[...] результирующий_файл

Команды для переименования папки имеет вид: MOVE [/Y|/-Y] [диск:][путь]каталог1 каталог2.

12. Достоинства и недостатки команд COPY и XCOPY.

Команда COPY имеет 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.

Указанные недостатки COPY можно решить с помощью команды XCOPY, которая предоставляет намного больше возможностей при

копировании. XCOPY может работать только с файлами и каталогами, но не с устройствами.

13. Назначение команды ECHO и примеры ее использования.

Команда ECHO применяется для вывода текстовых сообщений на стандартный вывод и для переключения режима отображения команд на экране.

С помощью команды ECHO OFF можно отключить дублирование команд, идущих после нее (сама команда ECHO OFF при этом все же дублируется).

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 @.

14. Команда DIR и ее возможности.

Команда: DIR [диск:][путь][имя_файла][ключи] используется для вывода информации о содержимом дисков и каталогов. Параметр [диск:][путь] задает диск и каталог, содержимое которого нужно вывести на экран. Параметр [имя_файла] задает файл или группу файлов, которые нужно включить в список. С помощью ключей команды DIR можно задать различные режимы расположения, фильтрации и сортировки.

15. В какой кодировке интерпретатор выводит информацию и как получить читаемую твердую копию?

При создании текстового файла интерпретатор командной строки использует кодировку кириллица (DOS). Поэтому рекомендуется переназначить вывод в файл с расширением .txt, а для просмотра содержимого файла использовать Internet Explorer, указав вид кодировки кириллица (DOS).

Цель работы: Знакомство с языком интерпретатора командной строки ОС MS Windows и командными файлами.

Ход работы:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания.
3. Ответить на контрольные вопросы.

Задание:

1. Вывод на экран имен всех файлов с указанным расширением, находящихся в каталоге, имя которого задается при запуске командного файла первым параметром. Расширение файлов задается вторым параметром.
2. Среди введенных с клавиатуры целых чисел (использовать SET /P) найти наибольшее и наименьшее. Признак конца ввода – знак «-».
3. В заданном каталоге и его подкаталогах найти общее количество подкаталогов. На экран вывести только требуемый результат.
4. В каталогах, имена которых заданы первым и вторым параметрами командного файла, найти и вывести на экран имена файлов (расширения могут быть любые), присутствующие, как в первом, так и во втором каталоге. Следует использовать только один оператор FOR.
5. Вычисление и вывод на экран значения факториала целого числа, задаваемого при запуске КФ. Предусмотреть проверку заданного значения и при задании отрицательного значения или значения, превышающего максимально возможную величину, выводить соответствующие сообщения. Для проверки правильности вычислений использовать калькулятор.
6. Просмотр содержимого каталога, указанного первым параметром КФ. Необходимо: 1) создать подкаталоги с именами EXE, TXT, CMD, DOC и OTHER. 2) В каждый подкаталог скопировать файлы с соответствующими расширениями. 3) Пустые подкаталоги удалить.
7. В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти файлы наибольшего и наименьшего размеров. Вывести имена файлов, их размеры и даты создания.

Описание выполнения работы

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

Создадим файл Task1.bat. Напишем код, который будет выполнять данную задачу.



```
*Task1.bat – Блокнот
Файл Правка Формат Вид Справка
@echo off
for %%f in (%1\%~N%2) do echo %%~N%F
```

Рисунок 2.1 – Листинг первой задачи

Вызовем программу с помощью командной строки, причём первым параметром укажем нужный нам каталог, а вторым – расширение искомых файлов. Результат выполнения кода, представленного на рисунке 2.1, изображён на рисунке 2.2.

```
PS D:\GH\university\term6\OS\lab2> .\Task1.bat ..\lab2 bat
Task1.bat
Task5.bat
task6.bat
task7.bat
Том в устройстве D имеет метку Local D
Серийный номер тома: 2D7E-8820

Содержимое папки D:\GH\university\term6\OS\lab2

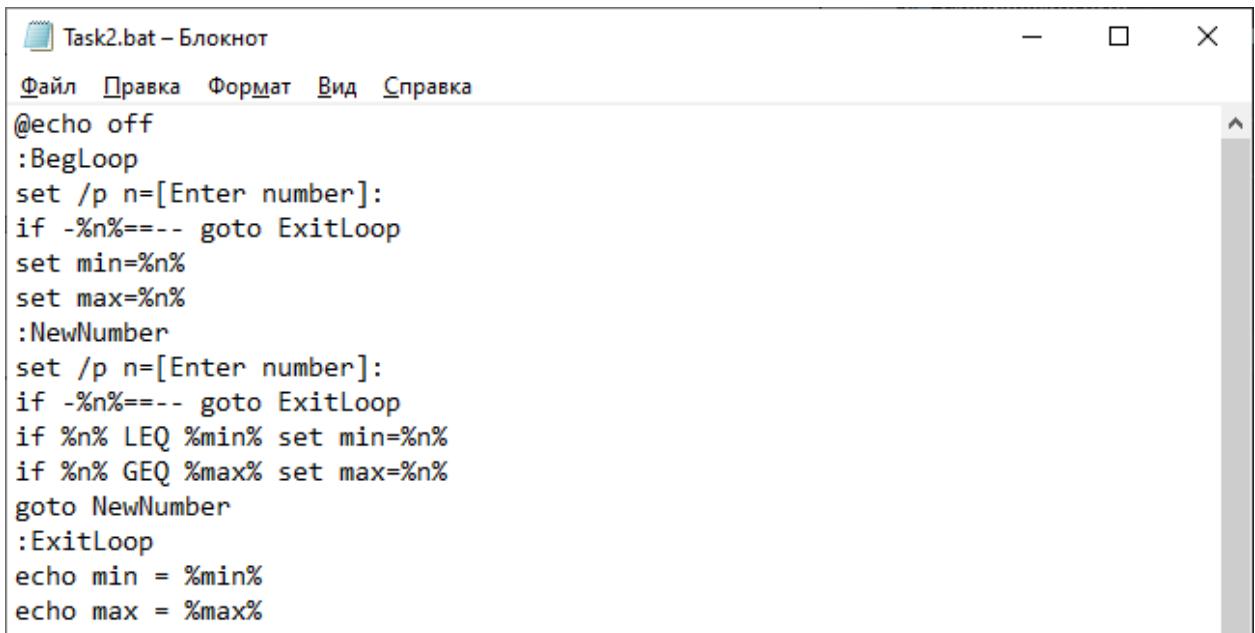
20.03.2023 21:00    <DIR>      .
20.03.2023 21:00    <DIR>      ..
20.03.2023 20:59          336 376 otchet2.docx
20.03.2023 20:55          56 Task1.bat
20.03.2023 21:33          334 Task5.bat
12.03.2023 23:45    <DIR>      task6
12.03.2023 23:45          412 task6.bat
19.03.2023 20:25    <DIR>      task7
19.03.2023 20:40          599 task7.bat
                           5 файлов      337 777 байт
                           4 папок   158 756 016 128 байт свободно
```

Рисунок 2.2 – Выполнение первой программы

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

Код программы нахождения наибольшего и наименьшего целых чисел представлен на рисунке 2.3.

Результат работы программы представлен на рисунке 2.4.



```
Task2.bat - Блокнот
Файл Правка Формат Вид Справка
@echo off
:BegLoop
set /p n=[Enter number]:
if -%n%==-- goto ExitLoop
set min=%n%
set max=%n%
:NewNumber
set /p n=[Enter number]:
if -%n%==-- goto ExitLoop
if %n% LEQ %min% set min=%n%
if %n% GEQ %max% set max=%n%
goto NewNumber
:ExitLoop
echo min = %min%
echo max = %max%
```

Рисунок 2.3 – Листинг второй задачи



```
Командная строка
D:\Lab2>Task2.bat
[Enter number]:2
[Enter number]:4
[Enter number]:7
[Enter number]:12
[Enter number]:34
[Enter number]:-5
[Enter number]:20
[Enter number]:-
min = -5
max = 34
D:\Lab2>
```

Рисунок 2.4 – Выполнение второй программы

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

Листинг программы, выполняющей данное задание, представлен на рисунке 2.5.

Результат выполнения этого кода изображен на рисунке 2.6.

```
Task3.bat – Блокнот
Файл Правка Формат Вид Справка
@echo off
setlocal enabledelayedexpansion
set /p cat=ENTER PATH:
set /a m=-1
for /r %cat% %%f in(.) do set /a m=m+1
echo count=%m%
tree %cat%
```

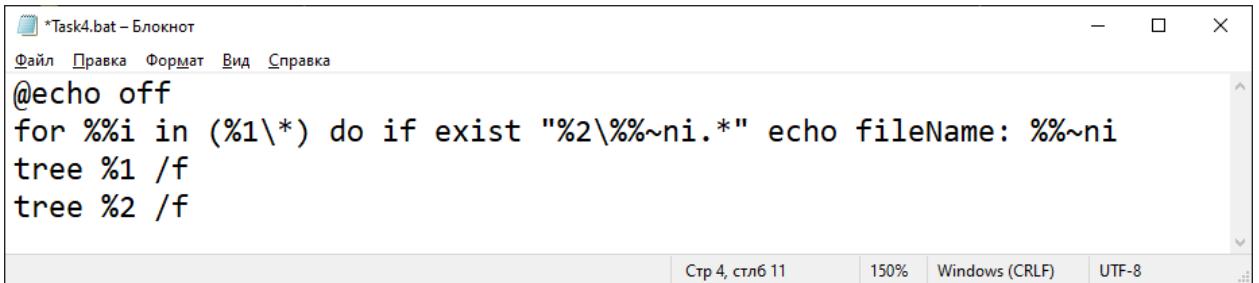
Рисунок 2.5 – Код третьей задачи

```
PS D:\GH\university\term6\OS\lab2> .\Task3.bat
ENTER PATH: ..
count=10
Структура папок тома Local D
Серийный номер тома: 000000D3 2D7E:8820
D:\GH\UNIVERSITY\TERM6\OS
└── lab1
    └── lab2
        ├── task6
        │   ├── cmd
        │   ├── doc
        │   ├── Files
        │   ├── other
        │   └── txt
        └── task7
            └── subCatalog
```

Рисунок 2.6 – Выполнение третьей программы

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

Следует использовать только один оператор FOR. Листинг программы, выполняющей данное задание, представлен на рисунке 2.7.



```
*Task4.bat – Блокнот
Файл Правка Формат Вид Справка
@echo off
for %i in (%1*) do if exist "%2\%~ni.*" echo fileName: %~ni
tree %1 /f
tree %2 /f

Стр 4, столб 11 150% Windows (CRLF) UTF-8
```

Рисунок 2.7 – Листинг четвёртой задачи

Результат выполнения этого кода изображен на рисунке 2.8.

```
PS D:\GH\university\term6\OS\lab2> .\Task4.bat .\task4\subC2\ .\task4\subC1
fileName: 1
fileName: 3
Структура папок тома Local D
Серийный номер тома: 0000006C 2D7E:8820
D:\GH\UNIVERSITY\TERM6\OS\LAB2\TASK4\SUBC2
    1.txt
    3.txt

Подпапки отсутствуют

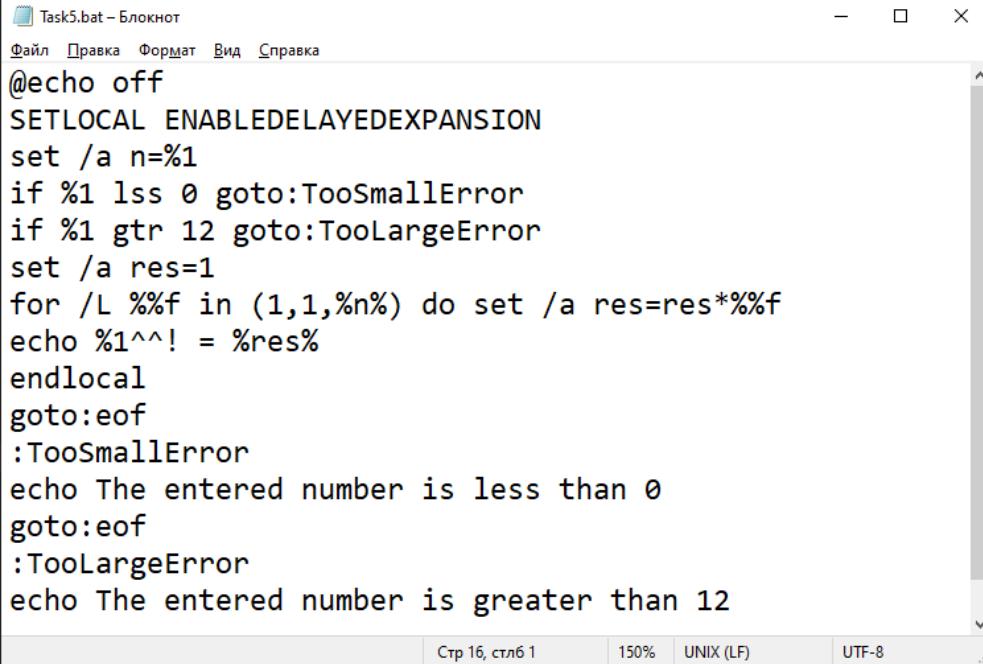
Структура папок тома Local D
Серийный номер тома: 000000FE 2D7E:8820
D:\GH\UNIVERSITY\TERM6\OS\LAB2\TASK4\SUBC1
    1.txt
    2.txt
    3.php
    4.cmd

Подпапки отсутствуют
```

Рисунок 2.8 – Выполнение четвёртой программы

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

Листинг программы, выполняющей данное задание, представлен на рисунке 2.9:



```
@echo off
SETLOCAL ENABLEDELAYEDEXPANSION
set /a n=%1
if %1 lss 0 goto:TooSmallError
if %1 gtr 12 goto:TooLargeError
set /a res=1
for /L %%f in (1,1,%n%) do set /a res=res*%%f
echo %1^^! = %res%
endlocal
goto:eof
:TooSmallError
echo The entered number is less than 0
goto:eof
:TooLargeError
echo The entered number is greater than 12
```

Рисунок 2.9 – Листинг пятой задачи

```
PS D:\GH\university\term6\OS\lab2> .\Task5.bat -3
The entered number is less than 0
PS D:\GH\university\term6\OS\lab2> .\Task5.bat 0
0! = 1
PS D:\GH\university\term6\OS\lab2> .\Task5.bat 1
1! = 1
PS D:\GH\university\term6\OS\lab2> .\Task5.bat 6
6! = 720
PS D:\GH\university\term6\OS\lab2> .\Task5.bat 13
The entered number is greater than 12
PS D:\GH\university\term6\OS\lab2> |
```

Рисунок 2.10 – Выполнение пятой программы

6. Просмотр содержимого каталога, указанного первым параметром КФ. Необходимо: 1. создать подкаталоги с именами EXE, TXT, CMD, DOC и

OTHER. 2. В каждый подкаталог скопировать файлы с соответствующими расширениями. 3. Пустые подкаталоги удалить.

Листинг программы, выполняющей задание 6, представлен на рисунке 2.11.



```
task6.bat – Блокнот
Файл Правка Формат Вид Справка
@echo off

md "%~1\exe" "%~1\doc" "%~1\cmd" "%~1\txt" "%~1\other"

for %%f in (%~1\Files\*.*) do (
    if "%~xf"==".exe" (
        copy %%f "%~1\exe"
    ) else if "%~xf"==".doc" (
        copy %%f "%~1\doc"
    ) else if "%~xf"==".cmd" (
        copy %%f "%~1\cmd"
    ) else if "%~xf"==".txt" (
        copy %%f "%~1\txt"
    ) else copy %%f "%~1\other"
)

FOR /F delims^= %%A IN ('DIR/AD/B/S^|SORT/R') DO RD "%%A"
cls
dir %1
```

Рисунок 2.11 – Код для выполнения 6 задания

Результат запуска КФ с параметром task6 изображен на рисунке 2.12. В папке-источника файлов намеренно не было файлов с расширением .exe, поэтому папки с именем exe не должно быть в результате выполнения КФ.

```
Том в устройстве D имеет метку Local D
Серийный номер тома: 2D7E-8820

Содержимое папки D:\GH\university\term6\OS\lab2\task6

12.03.2023 23:45    <DIR>      .
12.03.2023 23:45    <DIR>      ..
12.03.2023 23:37    <DIR>      cmd
12.03.2023 23:37    <DIR>      doc
12.03.2023 22:10    <DIR>      Files
12.03.2023 23:37    <DIR>      other
12.03.2023 23:37    <DIR>      txt
                           0 файлов          0 байт
                           7 папок   164 296 847 360 байт свободно

D:\GH\university\term6\OS\lab2>task6.bat task6
```

Рисунок 2.12 – Результат запуска КФ для 6 задания

7. В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти файлы наибольшего и наименьшего размеров. Вывести имена файлов, их размеры и даты создания.

Листинг программы, выполняющей задание 7, представлен на рисунке 2.13.

```
@echo off
cls
set maxsize=0
setlocal enabledelayedexpansion
for /f "tokens=* delims=" %%a in ('dir /a-d/b/s "%~1"') do (
    if %%~za gtr !maxsize! (
        set "name=%%~nxa"
        set "maxsize=%%~za"
        set "create=%%~ta"
    )
)
echo File max size
echo. %name% ^| %maxsize% ^| %create%
echo.
set minsize=%maxsize%
for /f "tokens=* delims=" %%a in ('dir /a-d/b/s "%~1"') do (
    if %%~za lss !minsize! (
        set "name=%%~nxa"
        set "minsize=%%~za"
        set "create=%%~ta"
    )
)
echo File min size
echo. %name% ^| %minsize% ^| %create%
pause>nul
exit
```

Рисунок 2.13 – Листинг КФ для выполнения задания 7.

Результат запуска КФ с параметром task7 изображен на рисунке 2.14.

```
File max size
test (2).php | 895776 | 19.03.2023 20:24
```

```
File min size
request.jpg | 863 | 17.03.2023 19:41
```

Рисунок 2.14 – Результат запуска КФ для 7 задания.

Контрольные вопросы

1. Вывод сообщений и дублирование команд. По умолчанию команды пакетного файла перед исполнением дублируются на экране, что иногда излишне. С помощью команды ECHO OFF можно отключить дублирование команд, идущих после нее. Включить дублирование можно аналогично командой: ECHO ON.

2. Использование параметров командной строки.

При запуске пакетных файлов в командной строке можно указывать произвольное число параметров, значения которых можно использовать внутри файла. Это позволяет, например, применять один и тот же командный файл для выполнения команд с различными параметрами. Для доступа из командного файла к параметрам командной строки применяются символы %0, %1, ..., %9 или %*. При этом вместо %0 подставляется имя выполняемого пакетного файла, вместо %1, %2, ..., %9 – значения первых девяти параметров командной строки соответственно, а вместо %* – все аргументы.

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

- WINDIR, которая определяет расположение каталога Windows,
- TEMP, которая определяет путь к каталогу для хранения временных файлов Windows
- PATH, в которой хранится системный путь (путь поиска), то есть список каталогов, в которых система должна искать выполняемые файлы или файлы совместного доступа (например, динамические библиотеки).

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

4. Операции со строковыми и числовыми переменными.

Со строковыми переменными можно выполнять операции конкатенации:

```
SET A=Раз
```

```
SET B=Два
```

```
SET C=%A%%B%
```

Можно выделить подстроку с помощью следующей конструкции: `%имя_переменной:~n1,n2%`, где число n1 определяет смещение (количество пропускаемых символов) от начала (если n1 положительно) или от конца (если n1 отрицательно) соответствующей переменной среды, а число n2 – количество выделяемых символов (если n2 положительно) или количество последних символов в переменной, которые не войдут в выделяемую подстроку (если n2 отрицательно). Если указан только один отрицательный параметр -n, то будут извлечены последние n символов.

Можно выполнять процедуру замены подстрок с помощью конструкции:

```
%имя_переменной:s1=s2% (в результате будет возвращена строка, в которой каждое вхождение подстроки s1 в соответствующей переменной заменено на s2).
```

При включенной расширенной обработке команд (этот режим в Windows используется по умолчанию) имеется возможность рассматривать значения переменных среды как числа и производить с ними арифметические вычисления (используются только целые числа). Для этого используется команда SET с ключом /A. В команде SET с ключом /A могут использоваться операции - (вычитание), * (умножение), / (деление нацело), % (остаток от деления). При использовании знака % в качестве знака операции в командных файлах он должен быть записан дважды.

5. Проверка существования заданного файла и наличия переменной среды.

Проверка существования заданного файла осуществляется с помощью следующей команды:

IF [NOT] EXIST файл команда1 [ELSE команда2]

Условие считается истинным, если указанный файл существует.

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, но принимает в качестве аргумента имя переменной среды и возвращает истинное значение, если эта переменная определена.

6. Выполнение заданной команды для всех элементов указанного множества.

Для выполнения какой-либо заданной команды для всех элементов указанного множества используется следующий вид цикла FOR:

FOR %%переменная IN (множество)

DO команда [параметры]

7. Выполнение заданной команды для всех подходящих имен файлов.

Для выполнения какой-либо заданной команды для всех подходящих имен файлов в качестве множества в цикл FOR передаются искомые имена или шаблон.

8. Выполнение заданной команды для всех подходящих имен каталогов.

Для выполнения какой-либо заданной команды для всех подходящих имен каталогов используется цикл FOR с ключом /D:

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

9. Выполнение заданной команды для определенного каталога, а также всех его подкаталогов.

Для выполнения какой-либо заданной команды для определенного каталога, а также всех его подкаталогов используется цикл FOR с ключом /R:

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

DO команда [параметры]

10. Получение последовательности чисел с заданными началом, концом и шагом приращения.

Получить последовательность чисел с заданными началом, концом и шагом приращения позволяет цикл FOR с ключом /L:

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

11. Чтение и обработка строк из текстового файла.

Чтение и обработка строк из текстового файла осуществляется с помощью цикла 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).

12. Команда Findstr. Назначение. Ключи. Использование регулярных выражений в команде. Задание и использование класса цифр и класса букв через диапазон.

Назначение команды - поиск строк в текстовых файлах.

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

/L-Поиск строк дословно.

/R-Поиск строк как регулярных выражений.

/S-Поиск файлов в текущей папке и всех ее подпапках.

/I-Определяет, что поиск будет вестись без учета регистра.

/X-Печатает строки, которые совпадают точно.

/V-Печатает строки, не содержащие совпадений с искомыми.

/N-Печатает номер строки, в которой найдено совпадение, и ее содержимое.

/M-Печатает только имя файла, в которой найдено совпадение.

/O-Печатает найденный строки через пустую строку.

/P-Пропускает строки, содержащие непечатаемые символы.

/T:файл-Читает список файлов из заданного файла

/C:строка-Использует заданную строку как искомую фразу поиска.

Также с этой командой могут использоваться и регулярные выражения:

. – Любой символ.

* – Повтор: ноль или более вхождений предыдущего символа или класса.

^ – Позиция в строке: начало строки.

\$ – Позиция в строке: конец строки.

[класс] – Класс символов: любой единичный символ из множества.

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

\x – Служебный символ: символьное обозначение служебного символа

x.

\<xuz – Позиция в слове: в начале слова.

xuz\> – Позиция в слове: в конце слова.

13. Операторы перехода и вызова.

Командный файл может содержать метки и команды GOTO перехода к этим меткам:

GOTO метка

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. Для перехода к метке внутри текущего командного файла кроме команды

GOTO можно использовать и 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.

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

Командный файл, выводящий на экран минимальное значения двух числовых аргументов, должен иметь, по крайней мере, 2 строчки (включая @echo off):

```
@ECHO OFF
```

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

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

Командный файл, выводящий на экран минимальное значения двух числовых аргументов, должен иметь, по крайней мере, 2 строчки (включая @echo off):

```
@ECHO OFF
```

```
if %1 GTR %2 (if %1 GTR %3 (echo %1) else (echo %3)) else (if %2 GTR %3 (echo %2) else (echo))
```

Цель работы: Знакомство с основными возможностями оболочки командной строки Windows PowerShell 2.0.

Ход работы:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания.
3. Ответить на контрольные вопросы.

Задание:

1. Ознакомится с теоретическими сведениями
2. Запустить оболочку PowerShell
3. Увеличить ширину окна оболочки до максимальной, увеличить высоту окна и задать цвет фона и цвет шрифта
4. Вывести содержимое каталога Windows по указанному в таблице 5 формату на экран и в текстовый файл.
5. Вывести в текстовый файл список свойств процесса, возвращаемый коммандлетом Get-process и на экран – их общее количество.
6. Создать текстовый файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в таблице 6 параметра. Имена параметров процессов указаны в той же таблице.
7. Создать HTML-файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в таблице 6 параметра. Имена параметров процессов указаны в той же таблице.
8. Найти суммарный объем всех графических файлов (bmp, jpg), находящихся в каталоге Windows и всех его подкаталогах.
9. Вывести на экран сведения о ЦП компьютера.
10. Найти максимальное, минимальное и среднее значение времени выполнение коммандлетов dir и ps
11. Выполнить индивидуальные задания для студентов бригад согласно таблице 7.

Описание выполнения работы

4. Вывести содержимое каталога Windows по указанному в таблице 5 формату на экран и в текстовый файл.

Задачу можно выполнить с помощью следующего скрипта:

```
Get-ChildItem -Path "C:\Windows" -Directory | Where-Object {$_ .Name -match "[st]$"} | Sort-Object Name | Select-Object Name, CreationTime, Attributes | Format-Table | Out-File "task4_output.txt"
```

В этом скрипте мы используем командлет `Get-ChildItem` для получения списка всех подкаталогов в каталоге Windows. Затем мы используем командлет `Where-Object` для фильтрации подкаталогов, которые имеют имена, оканчивающиеся на s или t. Далее мы используем командлет `Sort-Object` для сортировки подкаталогов по имени и командлет `Select-Object` для выбора свойств имени, даты создания и атрибутов.

Результаты выполнения скрипта представлен на рисунке 3.1, а содержимое файла `task4_output.txt` на рисунке 3.2

Name	CreationTime	Attributes
AAct_Tools	29.03.2022 17:06:29	Directory
addons	07.12.2019 17:38:43	Directory, NotContentIndexed
appcompat	07.12.2019 12:14:52	Directory, NotContentIndexed
AppReadiness	07.12.2019 12:14:52	Directory, NotContentIndexed
Boot	07.12.2019 12:14:52	Directory
Containers	07.12.2019 12:14:52	Directory, NotContentIndexed
Cursors	07.12.2019 12:14:52	Directory, NotContentIndexed
diagnostics	07.12.2019 12:14:52	Directory
Downloaded Program Files	07.12.2019 12:14:52	System, Directory, NotContentIndexed
en-US	07.12.2019 17:34:32	Directory, NotContentIndexed
Fonts	07.12.2019 12:14:52	ReadOnly, System, Directory, NotContentIndexed
L2Schemas	07.12.2019 12:14:52	Directory, NotContentIndexed
LiveKernelReports	07.12.2019 12:14:52	Directory, NotContentIndexed
Logs	07.12.2019 12:14:52	Directory, NotContentIndexed
Microsoft.NET	07.12.2019 12:14:52	ReadOnly, Directory, NotContentIndexed
ModemLogs	07.12.2019 12:14:52	Directory, NotContentIndexed
Offline Web Pages	07.12.2019 12:14:52	ReadOnly, Directory, NotContentIndexed
PolicyDefinitions	07.12.2019 12:14:52	Directory, NotContentIndexed
pss	12.04.2022 22:30:13	Directory
RemotePackages	07.12.2019 17:37:33	Directory, NotContentIndexed
Resources	07.12.2019 12:14:52	Directory, NotContentIndexed
schemas	07.12.2019 12:14:52	Directory, NotContentIndexed
ServiceProfiles	06.06.2021 22:42:39	Directory, NotContentIndexed
ShellComponents	07.12.2019 12:14:52	Directory, NotContentIndexed
ShellExperiences	07.12.2019 12:14:52	Directory, NotContentIndexed
SystemApps	07.12.2019 12:14:52	Directory, NotContentIndexed
SystemResources	07.12.2019 12:14:52	Directory
Tasks	07.12.2019 12:14:52	Directory, NotContentIndexed
Vss	07.12.2019 12:14:52	Directory, NotContentIndexed
WaaS	07.12.2019 12:14:52	Directory
WinSxS	07.12.2019 12:03:44	Directory

Рисунок 3.1 – Результат выполнения скрипта для 4 задания

Name	CreationTime	Attributes
---	-----	-----
AAct_Tools	29.03.2022 17:06:29	Directory
addons	07.12.2019 17:35:43	Directory, NotContentIndexed
appcompat	07.12.2019 12:14:52	Directory, NotContentIndexed
AppReadiness	07.12.2019 12:14:52	Directory, NotContentIndexed
Boot	07.12.2019 12:14:52	Directory
Containers	07.12.2019 12:14:52	Directory, NotContentIndexed
Cursors	07.12.2019 12:14:52	Directory, NotContentIndexed
diagnostics	07.12.2019 12:14:52	Directory
Downloaded Program Files	07.12.2019 12:14:52	System, Directory, NotContentIndexed
en-US	07.12.2019 17:34:32	Directory, NotContentIndexed
Fonts	07.12.2019 12:14:52	ReadOnly, System, Directory, NotContentIndexed
L2Schemas	07.12.2019 12:14:52	Directory, NotContentIndexed
LiveKernelReports	07.12.2019 12:14:52	Directory, NotContentIndexed
Logs	07.12.2019 12:14:52	Directory, NotContentIndexed
Microsoft.NET	07.12.2019 12:14:52	ReadOnly, Directory, NotContentIndexed
ModemLogs	07.12.2019 12:14:52	Directory, NotContentIndexed
Offline Web Pages	07.12.2019 12:14:52	ReadOnly, Directory, NotContentIndexed
PolicyDefinitions	07.12.2019 12:14:52	Directory, NotContentIndexed
pss	12.04.2022 22:30:13	Directory
RemotePackages	07.12.2019 17:37:33	Directory, NotContentIndexed
Resources	07.12.2019 12:14:52	Directory, NotContentIndexed
schemas	07.12.2019 12:14:52	Directory, NotContentIndexed
ServiceProfiles	06.06.2021 22:42:39	Directory, NotContentIndexed
ShellComponents	07.12.2019 12:14:52	Directory, NotContentIndexed
ShellExperiences	07.12.2019 12:14:52	Directory, NotContentIndexed
SystemApps	07.12.2019 12:14:52	Directory, NotContentIndexed
SystemResources	07.12.2019 12:14:52	Directory
Tasks	07.12.2019 12:14:52	Directory, NotContentIndexed
Vss	07.12.2019 12:14:52	Directory, NotContentIndexed
WaaS	07.12.2019 12:14:52	Directory
WinSxS	07.12.2019 12:03:44	Directory

Рисунок 3.2 – Содержимое файла task4_output.txt

5. Вывести в текстовый файл список свойств процесса, возвращаемый команделетом Get-process и на экран – их общее количество

Введем следующую команду для выполнения первой части задания:

```
Get-Process taskmgr | Out-File "task5_output.txt"
```

Введем следующую команду для выполнения второй части задания.

```
(Get-Process).count
```

Содержимое файла task5_output.txt представлено на рисунке 3.3, а результат выполнения скрипта для второй части на рисунке 3.4

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
747	37	43552	74204	149.42	10768	1	Taskmgr

Стр 7, столб 1 100% Windows (CRLF) UTF-16 LE

Рисунок 3.3 – Содержимое файла task5_output.txt

```
PS D:\GH\university\term6\OS\lab3> (Get-Process).Count
205
PS D:\GH\university\term6\OS\lab3> |
```

Рисунок 3.4 – Результат выполнения скрипта для второй части задания 5

6. Создать текстовый файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в таблице 6 параметра. Имена параметров процессов указаны в той же таблице

Следующий скрипт позволит выполнить данное задание:

```
Get-Process | Where-Object {$_ .Id -gt 100} | Select-Object Name,
PriorityClass, ProductVersion, Id | Sort-Object Name | Out-File
"task06_output.txt"
```

Содержимое файла task06_output.txt представлено на рисунке 3.5.

task6_output.txt – Блокнот

Name	PriorityClass	ProductVersion	Id
amdfendrsr			2172
amdw			5912
AMDRSServ			5164
AMDRSSrcExt	Normal	10.01.01.1862	3792
ApplicationFrameHost	Normal	10.0.19041.746	976
atieclxx			2992
atiesrxx			2184
chrome	Normal	111.0.5563.147	9220
chrome	Normal	111.0.5563.147	9404
chrome	Normal	111.0.5563.147	9020
chrome	AboveNormal	111.0.5563.147	8992
chrome	Normal	111.0.5563.147	9000
chrome	Idle	111.0.5563.147	10192
chrome	Idle	111.0.5563.147	10552
chrome	Normal	111.0.5563.147	9700
chrome	Normal	111.0.5563.147	9416
chrome	Normal	111.0.5563.147	9692
chrome	Normal	111.0.5563.147	8756
chrome	Idle	111.0.5563.147	4984
chrome	Normal	111.0.5563.147	5148
chrome	Idle	111.0.5563.147	2740
chrome	Normal	111.0.5563.147	1368
chrome	Idle	111.0.5563.147	1880
chrome	Idle	111.0.5563.147	7584
chrome	Normal	111.0.5563.147	8716
chrome	Normal	111.0.5563.147	6992
chrome	Idle	111.0.5563.147	6228
chrome	Idle	111.0.5563.147	6932
cncmd	Normal	8.01.01.1501	1196
Code	Normal	1.77.0	8180
Code	Normal	1.77.0	7696
Code	Normal	1.77.0	6668
Code	AboveNormal	1.77.0	8552
Code	Normal	1.77.0	10212
Code	Normal	1.77.0	10020
Code	Normal	1.77.0	9824

Рисунок 3.5 – Содержимое файла task06_output.txt

Создадим файл Task1.bat. Напишем код, который будет выполнять данную задачу.

7. Создать HTML-файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в таблице 6 параметра. Имена параметров процессов указаны в той же таблице.

Скрипт, позволяющий выполнить данное задание, имеет вид:

```
Get-Process | Where-Object {$_.Id -gt 100} | Select-Object Name, PriorityClass, ProductVersion, Id | Sort-Object Name | ConvertTo-HTML > "task7_output.html"
```

Результатом его работы будет содержимое файла task7_output.html. Оно представлено на рисунке 3.7

Name	PriorityClass	ProductVersion	Id
amdfendrsr			2172
amdow			5912
AMDRSServ			5164
AMDRSSrcExt	Normal	10.01.01.1862	3792
ApplicationFrameHost	Normal	10.0.19041.746	976
atieclxx			2992
atiesrxx			2184
chrome	Normal	111.0.5563.147	9220
chrome	Normal	111.0.5563.147	9404
chrome	Normal	111.0.5563.147	9020
chrome	AboveNormal	111.0.5563.147	8992
chrome	Normal	111.0.5563.147	9000
chrome	Idle	111.0.5563.147	10192
chrome	Idle	111.0.5563.147	10552
chrome	Normal	111.0.5563.147	9700
chrome	Normal	111.0.5563.147	9416
chrome	Normal	111.0.5563.147	9692
chrome	Normal	111.0.5563.147	8756

Рисунок 3.7 – Содержимое файла task7_output.html.

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

Скрипт, позволяющий выполнить данное задание, имеет вид:

```
Get-ChildItem -Path "C:\Windows" -Include "*.jpg", "*.bmp" -Recurse |  
Measure-Object -property length -sum
```

Результат его работы представлен на рисунке 3.8.

```
Count      : 158
Average    :
Sum        : 25179962
Maximum    :
Minimum    :
Property   : Length
```

Рисунок 3.8 – Результат работы скрипта для выполнения задания 8

9. Вывести на экран сведения о ЦП компьютера.

Следующий скрипт позволит выполнить задание:

```
Get-WmiObject Win32_Processor
```

Результат его работы представлен на рисунке 3.9

```
PS D:\GH\university\term6\OS\lab3> Get-WmiObject Win32_Processor

Caption : AMD64 Family 21 Model 2 Stepping 0
DeviceID : CPU0
Manufacturer : AuthenticAMD
MaxClockSpeed : 3800
Name : AMD FX(tm)-4300 Quad-Core Processor
SocketDesignation : CPU 1
```

Рисунок 3.9 –Информация о ЦП компьютера

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

Для нахождения времени работы команды в миллисекундах необходимо использовать команду:

```
(Measure-Command { dir }).TotalMilliseconds
```

Для выполнения задания необходимо сделать несколько измерений, для сохранения результатов которых воспользуемся массивом, который создается следующим образом:

```
$c = New-Object System.Collections.ArrayList
```

Добавление элемента происходит при помощи функции `Add`.

Для вывода минимального, максимального и среднего значения массива воспользуемся функцией:

```
measure -Maximum -Minimum -Average
```

Таким образом, получаем следующий код:

```
$c = New-Object System.Collections.ArrayList
for ($i = 1; $i -le 10; $i++) {
    $c.Add((Measure-Command { Get-ChildItem }).TotalMilliseconds) >
$cnull
}
Write-Output "results for command dir: "
$c | Measure-Object -Maximum -Minimum -Average
```

```

$c = New-Object System.Collections.ArrayList
for ($i = 1; $i -le 10; $i++) {
    $c.Add((Measure-Command { Get-Process }).TotalMilliseconds) >
>null
}
Write-Output "results for command ps: "
$c | Measure-Object -Maximum -Minimum -Average

```

Результат его выполнения отображен на рисунке 3.10

Вывод: по результатам проведенного эксперимента на текущей конфигурации компьютера под управлением ОС Windows команда dir работает быстрее команды ps.

```

results for command dir:

Count      : 10
Average    : 1.86862
Sum        :
Maximum   : 6.3995
Minimum   : 1.0877
Property  :

results for command ps:
Count      : 10
Average    : 5.48177
Sum        :
Maximum   : 7.6594
Minimum   : 3.9262
Property  :

```

Рисунок 3.10 – Время работы командлетов dir и ps

11. Выполнить индивидуальные задания:

11.1 Проверить наличие в текущем каталоге файлов одинакового размера. Если такие файлы есть – вывести их имена.

Задание можно выполнить, используя этот скрипт:

```

Get-ChildItem | Group-Object Length | Where-Object { $_.Count -gt 1 }
| ForEach-Object { $_.Group | Select-Object -ExpandProperty Name }

```

Результат его выполнения отображен на рисунке 3.11.

```

PS D:\GH\university\term6\OS\lab3> Get-ChildItem | Group-Object Length | Where-Object { $_.Count -gt 1 } | ForEach-Object { $_.Group | Select-Object -ExpandProperty Name }
fortask11_1.txt
fortask11_2.txt

```

Рисунок 3.11 – Список файлов одинакового размера текущего каталога

11.2 Найти среди выполняемых процессов имен процессов с наибольшим значением приоритета.

Данный скрипт покажет эти процессы:

```
Get-Process | Sort-Object Priority -Descending | Select-Object -First 50 Name
```

Результат его выполнения отображен на рисунке 3.12.

```
PS D:\GH\university\term6\OS\lab3> Get-Process | Sort-Object Priority -Descending | Select-Object -First 50 Name

Name
----
svchost
smss
sihost
ShellExperienceHost
StartMenuExperienceHost
sqlwriter
spoolsv
SearchProtocolHost
SearchIndexer
SearchFilterHost
```

Рисунок 3.12 – Список процессов с наивысшим приоритетом

Контрольные вопросы

1. Типы команд PowerShell (PS).

В оболочке PowerShell поддерживаются команды четырех типов: командлеты, функции, сценарии и внешние исполняемые файлы. Первый тип – так называемые командлеты (cmdlet). Этот термин используется пока только внутри PowerShell. Командлет – аналог внутренней команды интерпретатора командной строки – представляет собой класс .NET, порожденный от базового класса Cmdlet; разрабатываются командлеты с помощью пакета PowerShell Software Developers Kit (SDK). Единый базовый класс Cmdlet гарантирует совместимый синтаксис всех командлетов, а также автоматизирует анализ параметров командной строки и описание синтаксиса командлетов для встроенной справки. Командлеты рассматриваются в данной работе.

Данный тип команд компилируется в динамическую библиотеку (DLL) и подгружается к процессу PowerShell во время запуска оболочки (то есть сами по себе командлеты не могут быть запущены как приложения, но в них содержатся исполняемые объекты). Командлеты – это аналог внутренних команд традиционных оболочек. Следующий тип команд – функции. Функция – это блок кода на языке PowerShell, имеющий название и находящийся в памяти до завершения текущего сеанса командной оболочки. Функции, как и командлеты, поддерживают именованные параметры. Анализ синтаксиса функции производится один раз при ее объявлении. Сценарий – это блок кода на языке PowerShell, хранящийся во внешнем файле с расширением ps1. Анализ синтаксиса сценария производится при каждом его запуске. Последний тип команд – внешние исполняемые файлы, которые выполняются обычным образом операционной системой.

2. Имена и структура командлетов.

В PowerShell аналогом внутренних команд являются командлеты. Командлеты могут быть очень простыми или очень сложными, но каждый из

них разрабатывается для решения одной, узкой задачи. Работа с командлетами становится по-настоящему эффективной при использовании их композиции (конвейеризации объектов между командлетами).

Команды Windows PowerShell следуют определенным правилам именования: Команды Windows PowerShell состоят из глагола и существительного (всегда в единственном числе), разделенных тире. Глагол задает определенное действие, а существительное определяет объект, над которым это действие будет совершено. Команды записываются на английском языке. Пример: Get-Help вызывает интерактивную справку по синтаксису Windows PowerShell. Перед параметрами ставится символ «-». Например: Get-Help – Detailed. В Windows PowerShell также включены псевдонимы многих известных команд. Это упрощает знакомство и использование Windows PowerShell. Пример: команды help (классический стиль Windows) и man (классический стиль Unix) работают так же, как и Get-Help. Например, Get-Process (получить информацию о процессе), Stop-Service (остановить службу), Clear-Host (очистить экран консоли) и т.д. Чтобы просмотреть список командлетов, доступных в ходе текущего сеанса, нужно выполнить командлет Get-Command. По умолчанию командлет Get-Command выводит сведения в трех столбцах: CommandType, Name и Definition. При этом в столбце Definition отображается синтаксис командлетов (многоточие (...) в столбце синтаксиса указывает на то, что данные обрезаны). Замечание. Косые черты (/ и \) вместе с параметрами в оболочке Windows PowerShell не используются. В общем случае синтаксис командлетов имеет следующую структуру: имя_командлета –параметр1 -параметр2 аргумент1 аргумент2. Здесь параметр1 – параметр (переключатель), не имеющий значения; параметр2 – имя параметра, имеющего значение аргумент1; аргумент2 – параметр, не имеющий имени. Например, командлет GetProcess имеет параметр Name, который определяет имя процесса, информацию о котором нужно вывести. Имя этого параметра указывать необязательно. Таким образом, для получения сведений о процессе

Far можно ввести либо команду Get-Process -Name Far, либо команду Get-Process Far.

3. Псевдонимы команд.

Механизм псевдонимов, реализованный в оболочке PowerShell, дает возможность пользователям выполнять команды по их альтернативным именам (например, вместо команды Get-Childitem можно пользоваться псевдонимом dir). В PowerShell заранее определено много псевдонимов, можно также добавлять собственные псевдонимы в систему. Псевдонимы в PowerShell делятся на два типа. Первый тип предназначен для совместимости имен с разными интерфейсами. Псевдонимы этого типа позволяют пользователям, имеющим опыт работы с другими оболочками (Cmd.exe или Unix-оболочки), использовать знакомые им имена команд для выполнения аналогичных операций в PowerShell, что упрощает освоение новой оболочки, позволяя не тратить усилий на запоминание новых команд PowerShell. Например, пользователь хочет очистить экран. Если у него есть опыт работы с Cmd.exe, то он, естественно, попробует выполнить команду cls. PowerShell при этом выполнит командлет Clear-Host, для которого cls является псевдонимом и который выполняет требуемое действие – очистку экрана. Для пользователей Cmd.exe в PowerShell определены псевдонимы cd, cls, copy, del, dir, echo, erase, move, popd, pushd, ren, rmdir, sort, type; для пользователей Unix – псевдонимы cat, chdir, clear, diff, h, history, kill, lp, ls, mount, ps, pwd, r, rm, sleep, tee, write. Узнать, какой именно командлет скрывается за знакомым псевдонимом, можно с помощью командлета Get-Alias.

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

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

4. Просмотр структуры объектов.

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

```
PS C:\> Get-Process | Get-Member
```

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

```
PS C:\> Get-Process | Get-Member -MemberType Property
```

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

5. Фильтрация объектов в конвейере. Блок сценария.

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

объект удовлетворяет условиям проверки. Например, для вывода информации о «зависших» процессах (объекты, возвращаемые командлетом Get-Process, у которых свойство Responding равно False) можно использовать следующий конвейер:

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

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

```
Get-Process | Where-Object { $_.Id -gt 1000 } 67
```

В блоках сценариев командлета Where-Object для обращения к текущему объекту конвейера и извлечения нужных свойств этого объекта используется специальная переменная `$_`, которая создается оболочкой PowerShell автоматически. Данная переменная используется и в других командах, производящих обработку элементов конвейера. Условие проверки в Where-Object задается в виде блока сценария – одной или нескольких команд PowerShell, заключенных в фигурные скобки `{}`. Результатом выполнения данного блока сценария должно быть значение логического типа: True (истина) или False (ложь). Как можно понять из примеров, в блоке сценария используются специальные операторы сравнения.

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

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

Операторы сравнения можно соединять друг с другом с помощью логических операторов (см. таблице 2).

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

6. Какую информацию выводит команда Get-Help * ?

Get-Help * перечисляет все команды Windows PowerShell.

7. Командлеты для форматирования выводимой информации.

В традиционных оболочках команды и утилиты сами форматируют выводимые данные. Некоторые команды (например, dir в интерпретаторе Cmd.exe) позволяют настраивать формат вывода с помощью специальных параметров. В оболочке PowerShell вывод форматируют только четыре специальных командлета Format (таблица 4). Это упрощает изучение, так как не нужно запоминать средства и параметры форматирования для других команд (остальные командлеты вывод не форматируют).

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

Как уже отмечалось, если ни один из командлетов Format явно не указан, то используется модуль форматирования по умолчанию, который определяется по типу отображаемых данных.

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

При использовании формата списка выводится больше сведений о каждой службе, чем в формате таблицы (вместо трех столбцов данных о каждой службе в формате списка выводятся девять строк данных). Однако это вовсе не означает, что командлет Format-List извлекает дополнительные сведения о службах. Эти данные содержатся в объектах, возвращаемых командлетом Get-Service, однако командлет FormatTable, используемый по умолчанию, отбрасывает их, потому что не может вывести на экран более трех столбцов. При форматировании вывода с помощью командлетов FormatList и Format-Table можно указывать имена свойства объекта, которые должны быть отображены (напомним, что просмотреть список свойств, имеющихся у объекта, позволяет рассмотренный ранее командлет Get-Member).

Вывести все имеющиеся у объектов свойства можно с помощью параметра *.

8. Перенаправление выводимой информации.

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

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

По умолчанию выводимая информация передается командлету OutDefault, который, в свою очередь, делегирует всю работу по выводу строк на экран командлету Out-Host. Для понимания данного механизма нужно учитывать, что архитектура PowerShell подразумевает различие между собственно ядром оболочки (интерпретатором команд) и главным приложением (host), которое использует это ядро. В принципе, в качестве главного может выступать любое приложение, в котором реализован ряд специальных интерфейсов, позволяющих корректно интерпретировать получаемую от PowerShell информацию. В нашем случае главным приложением является консольное окно, в котором мы работаем с оболочкой, и командлет Out-Host передает выводимую информацию в это консольное окно. Параметр Paging командлета Out-Host, подобно команде more интерпретатора Cmd.exe, позволяет организовать постраничный вывод информации, например: Get-Help Get-Process –Full | Out-Host –Paging.

9. Управляющие инструкции PS.

A) Инструкция If ... ElseIf ... Else

В общем случае синтаксис инструкции If имеет вид

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

При выполнении инструкции If проверяется истинность условного выражения условие1. Если условие1 имеет значение \$True, то выполняется блок_кода1, после чего выполнение инструкции if завершается. Если условие1 имеет значение \$False, проверяется истинность условного выражения условие2. Если условие2 имеет значение \$True, то выполняется

блок_код2 и выполнение инструкции if завершается. Если и условие1, и условие2 имеют значение \$False, то выполняется блок_код3 и выполнение инструкции if завершается.

Б) Циклы While и Do ... While

Самый простой из циклов PS – цикл While, в котором команды выполняются до тех пор, пока проверяемое условие имеет значение \$True. Инструкция While имеет следующий синтаксис: While (условие) {блок_команд} Цикл Do ... While похож на цикл While, однако условие в нем проверяется не до блока команд, а после: Do {блок_команд} While (условие).

В) Цикл For Обычно цикл For применяется для прохождения по массиву и выполнения определенных действий с каждым из его элементов.

Синтаксис инструкции For:

For (инициация; условие; повторение) {блок_команд}

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

ForEach (\$элемент in \$коллекция) {блок_команд}

При выполнении цикла ForEach автоматически создается переменная \$элемент. Перед каждой итерацией в цикле этой переменной присваивается значение очередного элемента в коллекции. В разделе блок_команд содержатся команды, выполняемые на каждом элементе коллекции. Инструкция ForEach может также использоваться совместно с командлетами, возвращающими коллекции элементов.

10. Назначение регулярных выражений.

Регулярные выражения (или сокращенно “регэкспы” (regexp, regular expressions)) обладают огромной мощью, и способны сильно упростить

жизнь системного администратора или программиста. В PowerShell регулярные выражения легко доступны, удобны в использовании и максимально функциональны. PowerShell использует реализацию регулярных выражений .NET.

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

Знакомство с регулярными выражениями начнем с более простой технологии, служащей подобным целям – с подстановочных символов. Наверняка вы не раз выполняли команду dir, указывая ей в качестве аргумента маску файла, например *.exe. В данном случае звёздочка означает “любое количество любых символов”. Аналогично можно использовать и знак вопроса, он будет означать “один любой символ”, то есть dir ???.exe выведет все файлы с расширением .exe и именем из двух 71 символов.

В PowerShell можно применять и еще одну конструкцию – группы символов. Так например [a-f] будет означать “один любой символ от a до f, то есть (a,b,c,d,e,f)”, а [smw] любую из трех букв (s, m или w). Таким образом команда get-childitem [smw]???.exe выведет файлы с расширением .exe, у которых имя состоит из трех букв, и первая буква либо s, либо m, либо w.

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

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

```
True
```

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

```
PS C:\> "Shell" -match "Power"
```

False

```
PS C:\> "PowerShell" -match "rsh"
```

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

11. Сохранение данных в текстовом файле и html-файле.

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

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

Для просмотра содержимого файла можно использовать командлет Invoke-Item «имя документа» Например Invoke-Item “processes.htm”

12. Получение справочной информации в PS.

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

Get-Help выводит на экран справку об использовании справки.

Get-Help * перечисляет все команды Windows PowerShell.

Get-Help команда выводит справку по соответствующей команде.

Get-Help команда -Detailed выводит подробную справку с примерами команды.

Использование команды help для получения подробных сведений о команде help: Get-Help Get-Help -Detailed.

Команда Get-Help позволяет просматривать справочную информацию не только о разных командлетах, но и о синтаксисе языка PowerShell, о псевдонимах и т. д. Например, чтобы прочитать справочную информацию об использовании массивов в PowerShell, нужно выполнить следующую команду: Get-Help about_array.

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

13. Как создать массив в PS?

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

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

Можно создать и инициализировать массив, используя оператор диапазона (...). Например, команда PS C:\> \$b=10..15 создает и инициализирует массив \$b, содержащий 6 значений 10, 11, 12, 13, 14 и 15.

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

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

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

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

14. Как объединить два массива?

Можно объединить два массива, например \$b и \$c в один с помощью операции конкатенации +. Например: PS C:\> \$d=\$b+\$c.

15. Как увеличить размер созданного в PS массива?

Имеется способ увеличения первоначально определенной длины массива. Для этого можно воспользоваться оператором конкатенации + или +=.

```
PS C:\>$a+=5,6
```

16. Как ввести данные в массив?

Следующая команда создаст массив \$a из трех элементов:

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

Можно инициализировать массив, используя оператор диапазона (...). Например, команда PS C:\> \$b=10..15 создает и инициализирует массив \$b, содержащий 6 значений 10, 11, 12, 13, 14 и 15.

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

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

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

17. Использование командлета Out-Null.

Командлет Out-Null служит для поглощения любых своих входных данных. Это может пригодиться для подавления вывода на экран ненужных сведений, полученных в качестве побочного эффекта выполнения какой-либо команды.

18. Оператор PowerShell –match.

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

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

True

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

```
PS C:\> "Shell" -match "Power"
```

False

```
PS C:\> "PowerShell" -match "rsh"
```

True

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

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

False

19. Использование символа ^ в командлетах.

"Крышка" в качестве первого символа группы символов означает именно отрицание. То есть на месте группы может присутствовать любой символ кроме перечисленных в ней.

Для того чтобы включить отрицание в символьных группах (\d, \w, \s), не обязательно заключать их в квадратные скобки, достаточно перевести их в верхний регистр.

^ как символ отрицания используется лишь в начале группы символов, а вне группы – этот символ является уже якорем.

20. Использование символа \$ в командлетах.

\$ (знак доллара) - обозначает конец строки.

21. Количественные модификаторы (квантификаторы).

В регулярных выражениях существует специальная конструкция – «количественные модификаторы» (квантификаторы). Эти модификаторы приписываются к любой группе справа, и определяют количество вхождений этой группы.

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

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

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

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

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

22. Использование групп захвата.

Как следует из названия, группы можно использовать для группировки. К группам захвата, как и к символам и символьным группам, можно применять количественные модификаторы.

Например, следующее выражение означает «Первая буква в строке – S, затем одна или больше групп, состоящих из «знака – (минус) и любого количества цифр за ним» до конца строки»:

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

True

23. Командлеты для измерения свойств объектов.

Для измерения времени выполнения командлетов PS служит командлет Measure-Command. В качестве примера можно рассмотреть получение времени выполнения командлета dir, выполнив команду (Measure-Command {dir}).TotalSeconds.

Для получения статистических данных используют командлет Measure-Object. Для числовых массивов с его помощью можно получить максимальное, минимальное, среднее значение элементов массива и их сумму. Если имеется инициализированный массив ms, для указанной цели используется команделт \$ms | measure-object –maximum –minimum –average –sum.

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

Ход работы:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания.
3. Ответить на контрольные вопросы.

Задание:

1. Построить графики изменения количества потоков приложений Sublime Text 4 и Word при создании документа, содержащего текст из одного слова
2. Для приложения Калькулятор построить 2-3 наиболее динамично изменяющихся графика изменения текущего приоритета потоков при вычислении значения арифметического выражения, перемещении калькулятора по экрану, перемещении курсора мыши по экрану в области окна калькулятора
3. Для приложения Word построить график изменения объема используемого файла подкачки при последовательном открытии 3-4 файлов увеличивающегося размера
4. Выполнить индивидуальное задание: для каждого ядра процессора выяснить, в каком режиме ядро работает больше времени – пользовательском или системном?

Описание выполнения работы

1 Построить графики изменения количества потоков приложений Siblime (Блокнот) и Word при создании документа, содержащего текст из одного слова.

Для выполнения данного задания необходимо запустить программу для работы со счетчиками производительности. В ОС Windows для этого используется встроенная программа Performance Monitor (perfmon.exe). Для ее запуска необходимо нажать сочетание клавиш «Win»+«R» и ввести название программы – «perfmon» (см. Рисунок 4.1).

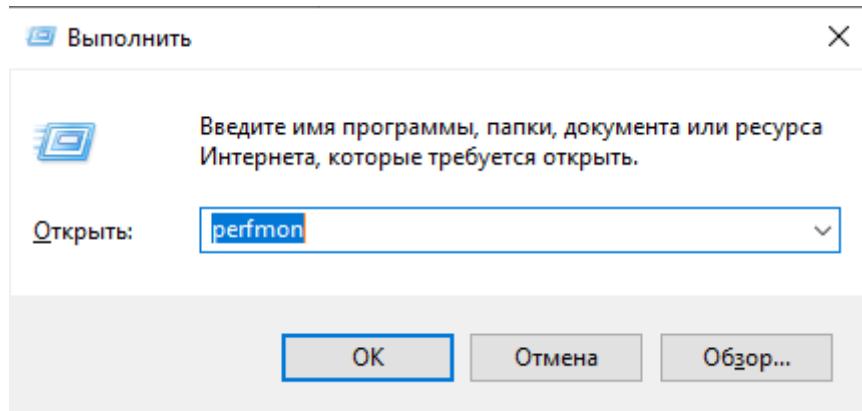


Рисунок 4.1 – Запуск программы Performance Monitor

После запуска программы отобразится область для построения графиков (Рисунок 4.2). Для добавления счетчиков необходимо нажать на кнопку с символом «+» зеленого цвета на панели инструментов программы, в результате чего отобразиться окно выбора необходимых счетчиков.

Так как нам необходимо отобразить изменение количества потоков для приложений Sublime (Блокнот) и Word, то следует добавить «Счетчик потоков», находящийся в группе «Процессы», и выбрать процессы «sublime_text» и «WinWord» (см. Рисунок 4.3).

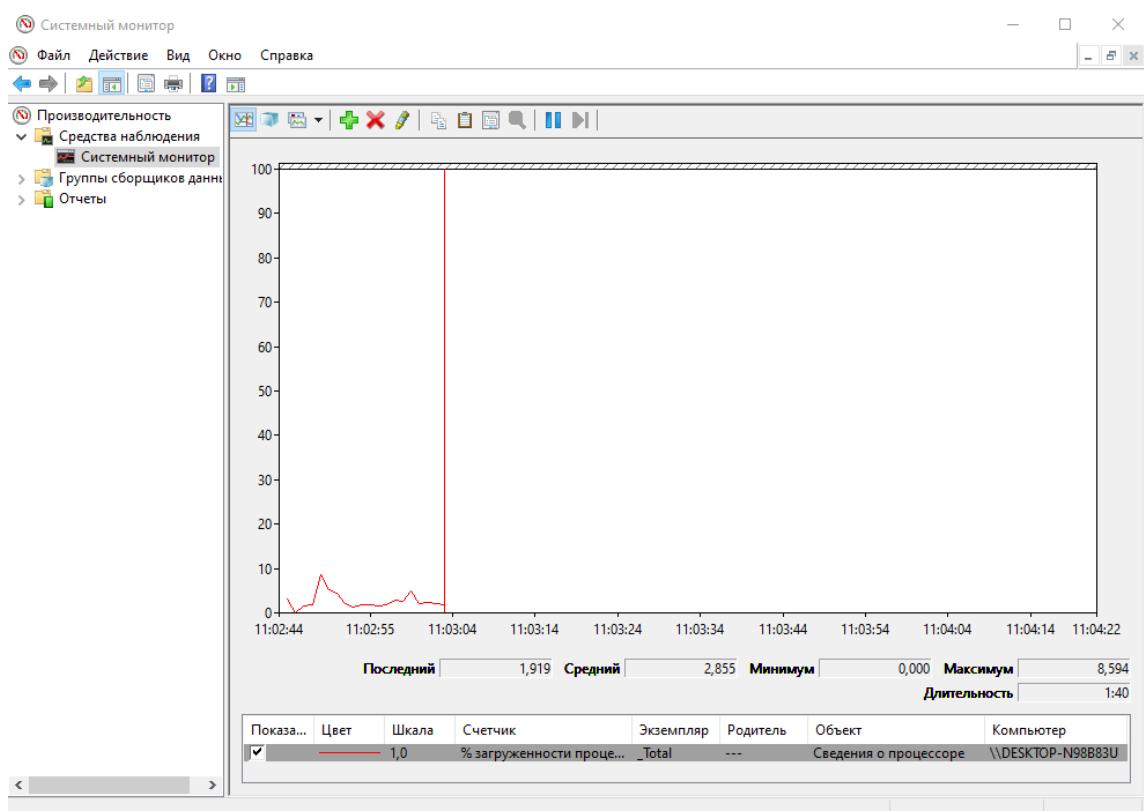


Рисунок 4.2 – Программа Performance Monitor (Системный монитор)

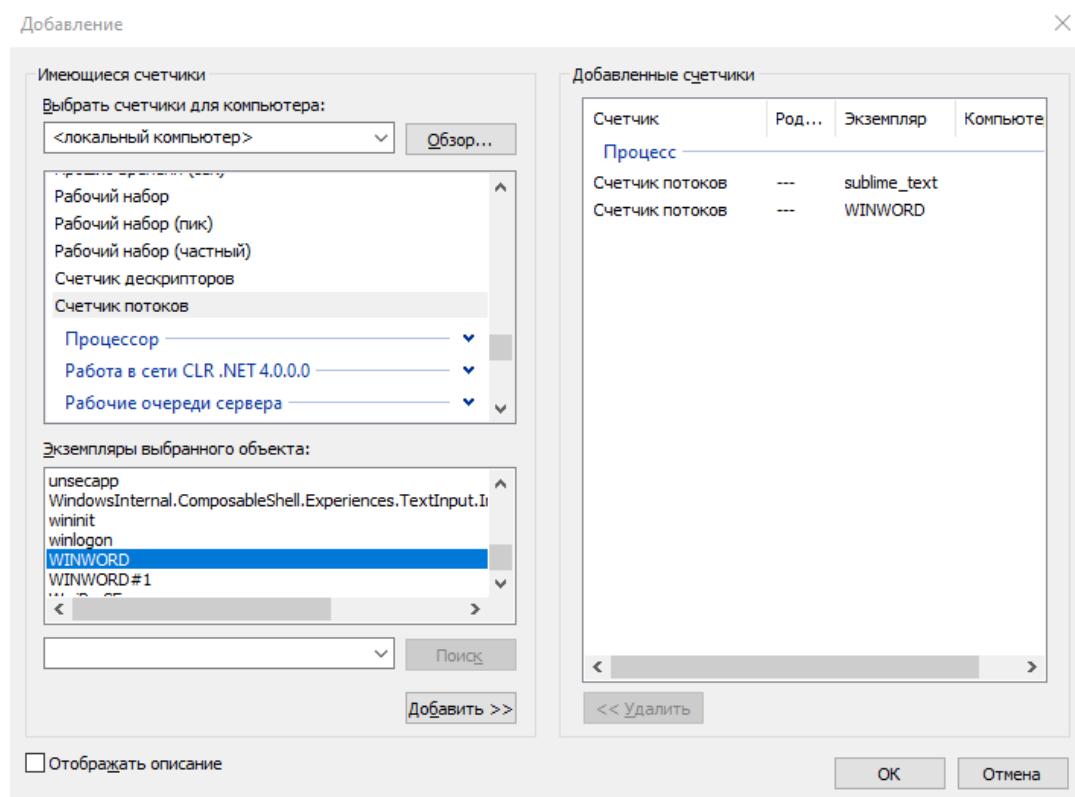


Рисунок 4.3 – Добавление счетчиков потоков приложений Sublime, WordPad

После добавления счетчиков в область для построения графиков в ней появятся графики, отображающие изменения количества потоков для выбранных процессов (**Error! Reference source not found.**). Наберем в каждом из текстовых редакторов по одному слову и сохраним соответствующие файлы, после чего закроем программу. Согласно полученному графику (**Error! Reference source not found.**) сохранение файла сопровождается существенным увеличением количества потоков. Также можно отметить, что приложение «Word» оперирует большим количеством потоков чем «Sublime». Возможно, это связано с большими возможностями этого текстового редактора.

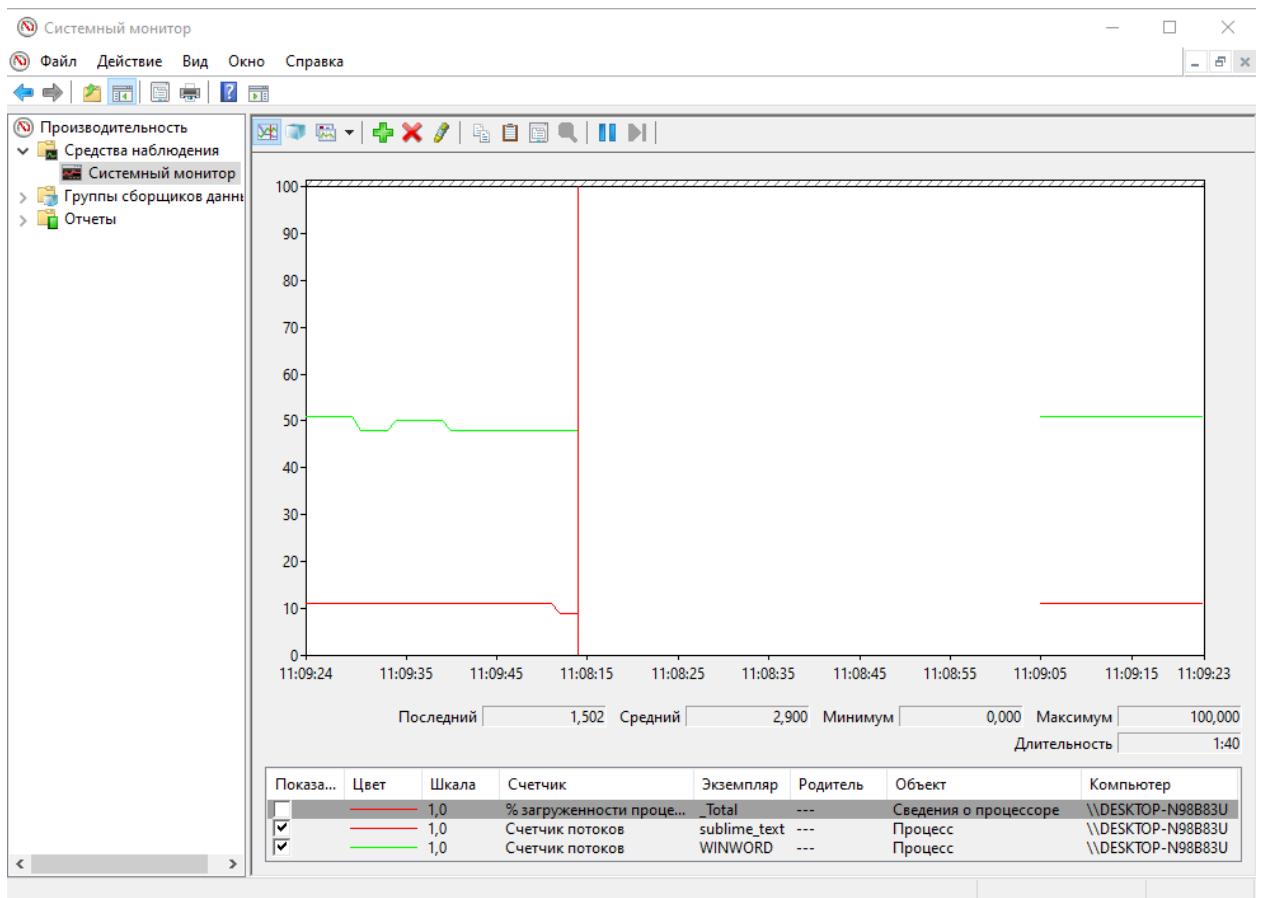


Рисунок 4.4 – Графики количества потоков приложений Sublime, Word

2 Для приложения Калькулятор построить 2-3 наиболее динамично изменяющихся графика изменения текущего приоритета потоков при вычислении значения арифметического выражения, перемещении

калькулятора по экрану, перемещении курсора мыши по экрану в области окна калькулятора.

Для отображения требуемой в задании информации добавим счетчик «Текущий приоритет» для каждого потока процесса «Calculator» (Рисунок 4.4). Выполним несколько произвольных вычислений в приложении «Калькулятор» и выведем соответствующие графики изменения приоритетов потоков (**Error! Reference source not found.**).

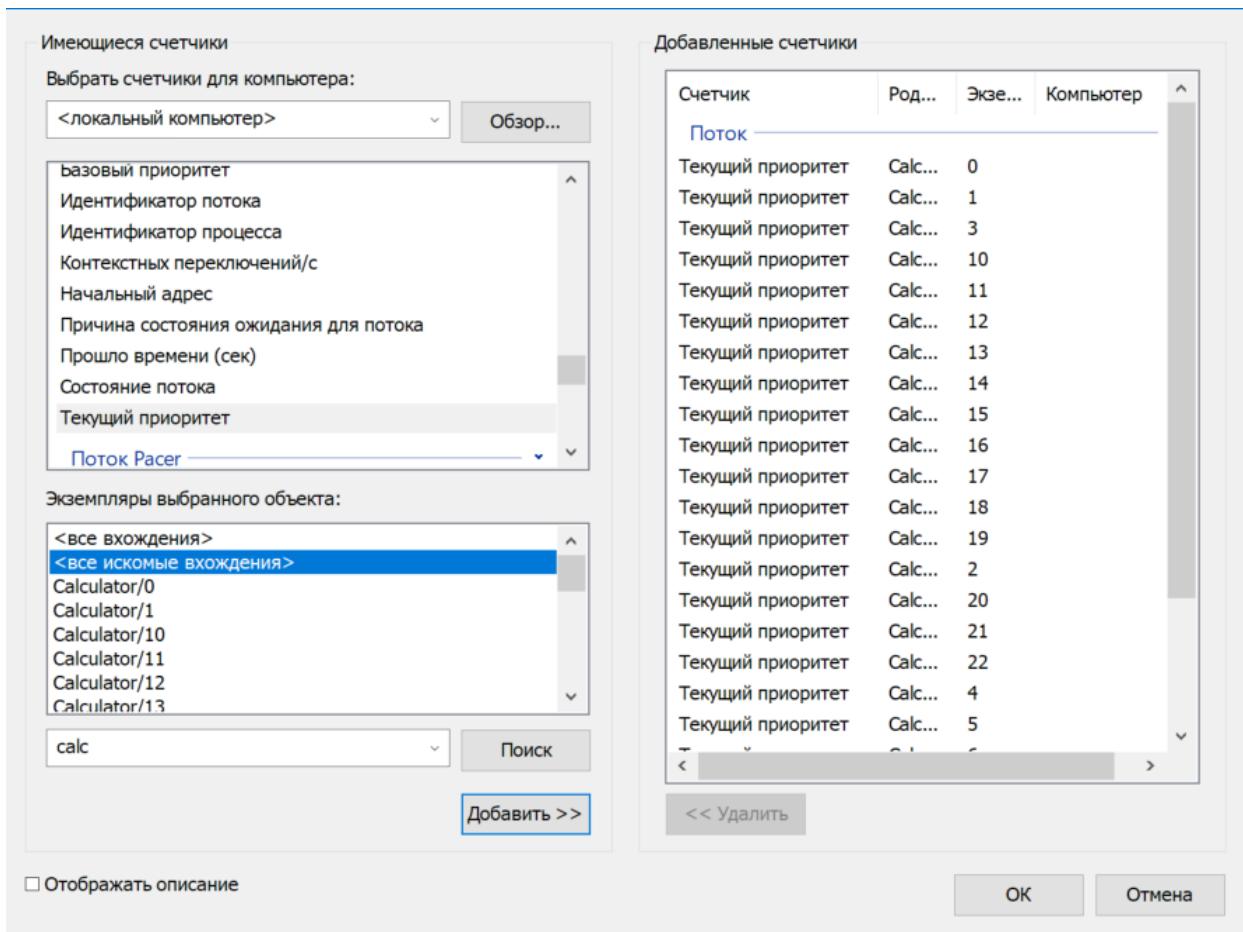


Рисунок 4.4 – Добавление счетчиков «Текущий приоритет» при вычислении арифметических выражений в программе «Калькулятор»

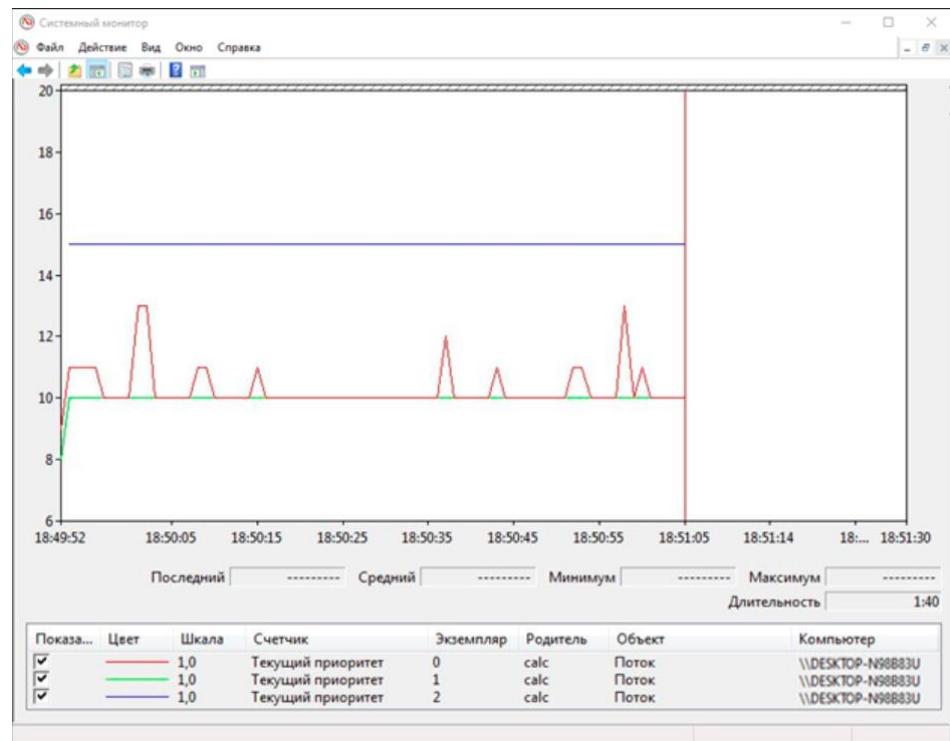


Рисунок 4.6 – График текущего приоритета процессов при вычислении арифметических выражений в программе «Калькулятор»

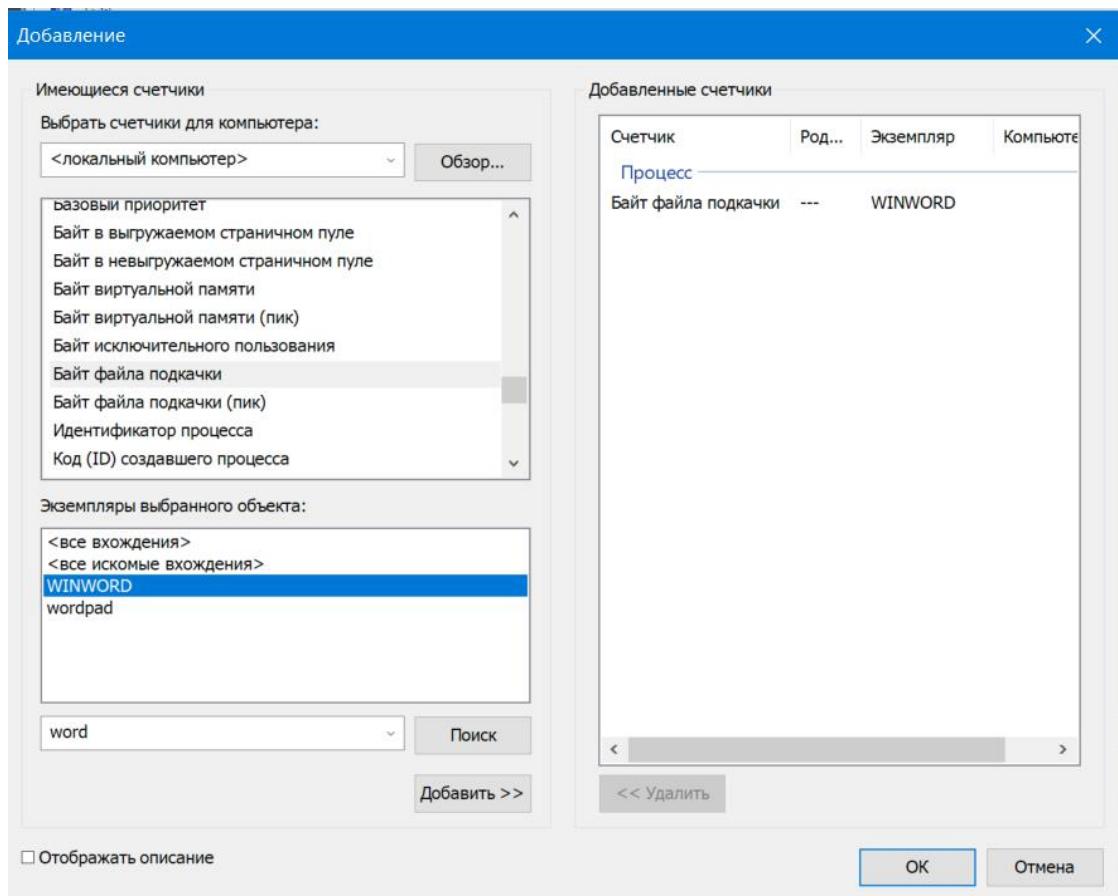


Рисунок 4.7 – Добавление счетчика «Байт файла подкачки»

3 Для приложения Word построить график изменения объема используемого файла подкачки при последовательном открытии 3-4 файлов увеличивающегося размера.

Для построения графика изменения объема используемого файла подкачки, добавим счетчик «Байт файла подкачки» из группы «Процессы» (см. Рисунок 4.).

Последовательно откроем три файла размером 430 Кб, 837 Кб и 1102 Кб соответственно, а затем проанализируем полученный график (Рисунок 4.85). Данный график показывает, что при каждом открытии файла происходит рост объема файла подкачки, однако он слабо зависит от размера файла, даже самого большого. Это говорит об эффективном управлении памятью приложением «Word».

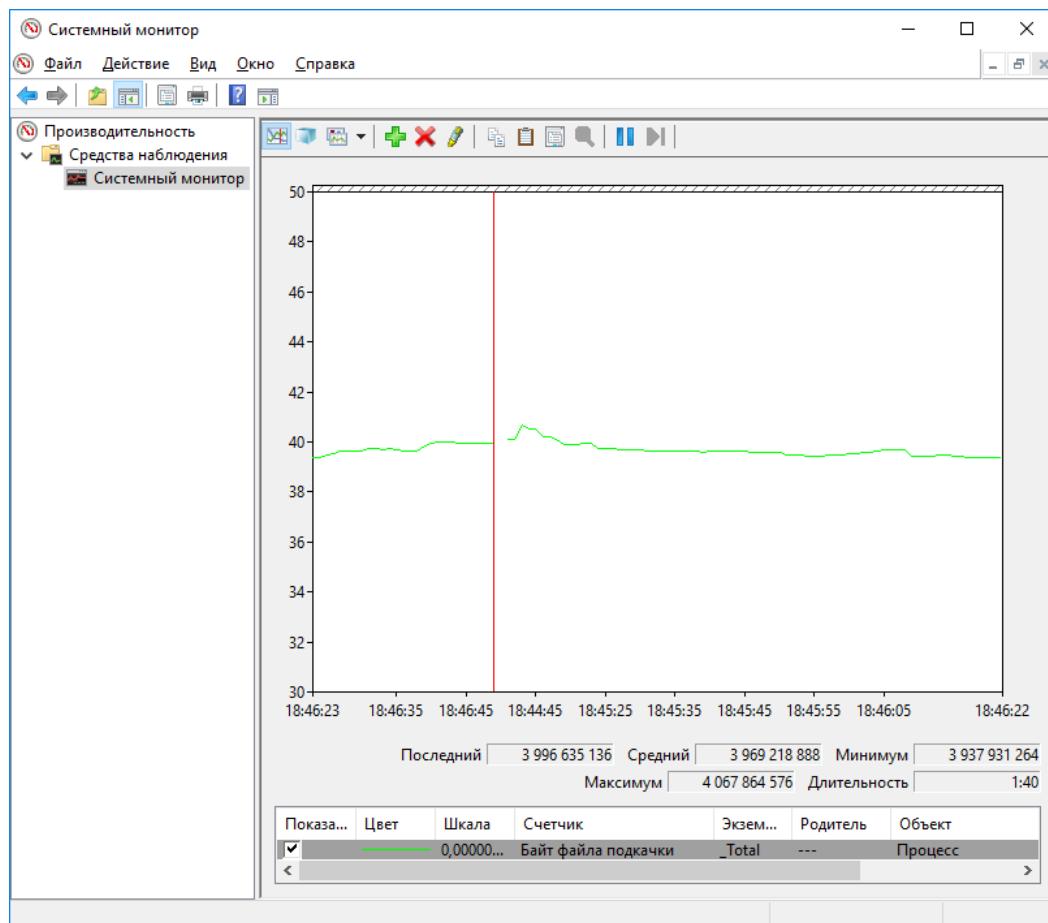


Рисунок 4.85 – График изменения размера файла подкачки при последовательном открытии 3 файлов увеличивающегося размера в приложении «Word»

4 Для каждого ядра процессора выяснить, в каком режиме ядро работает больше времени – пользовательском или системном?

Для этого в Системном мониторе на вкладке добавления счётчиков найдем «Сведения о процессоре» и в качестве источника укажем 2 объекта: «% работы в пользовательском режиме» и «% работы в привилегированном режиме», а экземплярами объекта будут все 4 ядра процессора (рисунок 4.9).

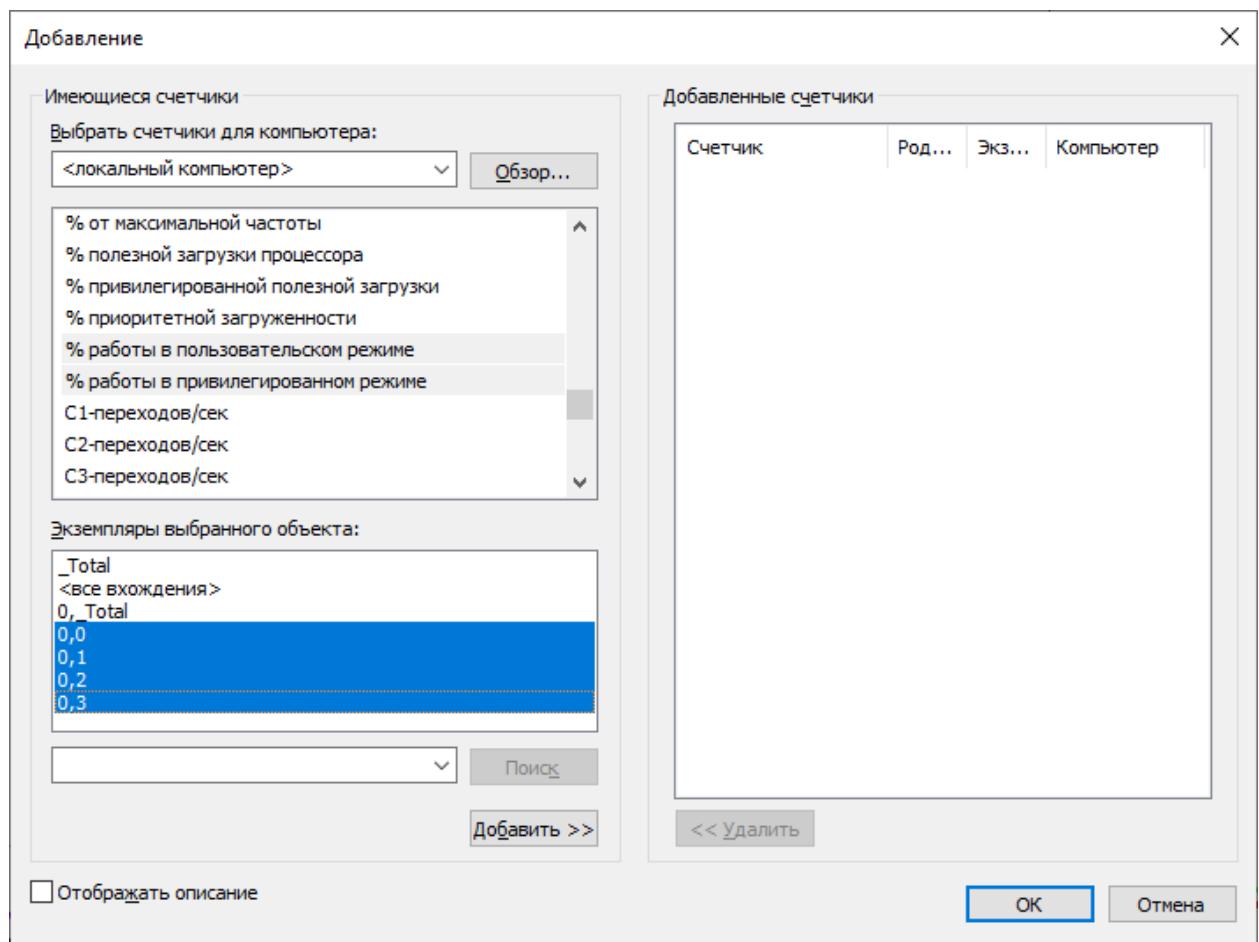


Рисунок 4.9 – выбор данных для выполнения 4 задачи.

Счетчики выставлены и в монитор добавлены 8 данных для отслеживания. После завершения сбора данных можем оценить работу ядер, последовательно отображая в мониторе их графики. Для удобства обозначим красным линию красным для «пользовательского режима» и зеленым для «привилегированного». Результат отображен на рисунке 4.10.

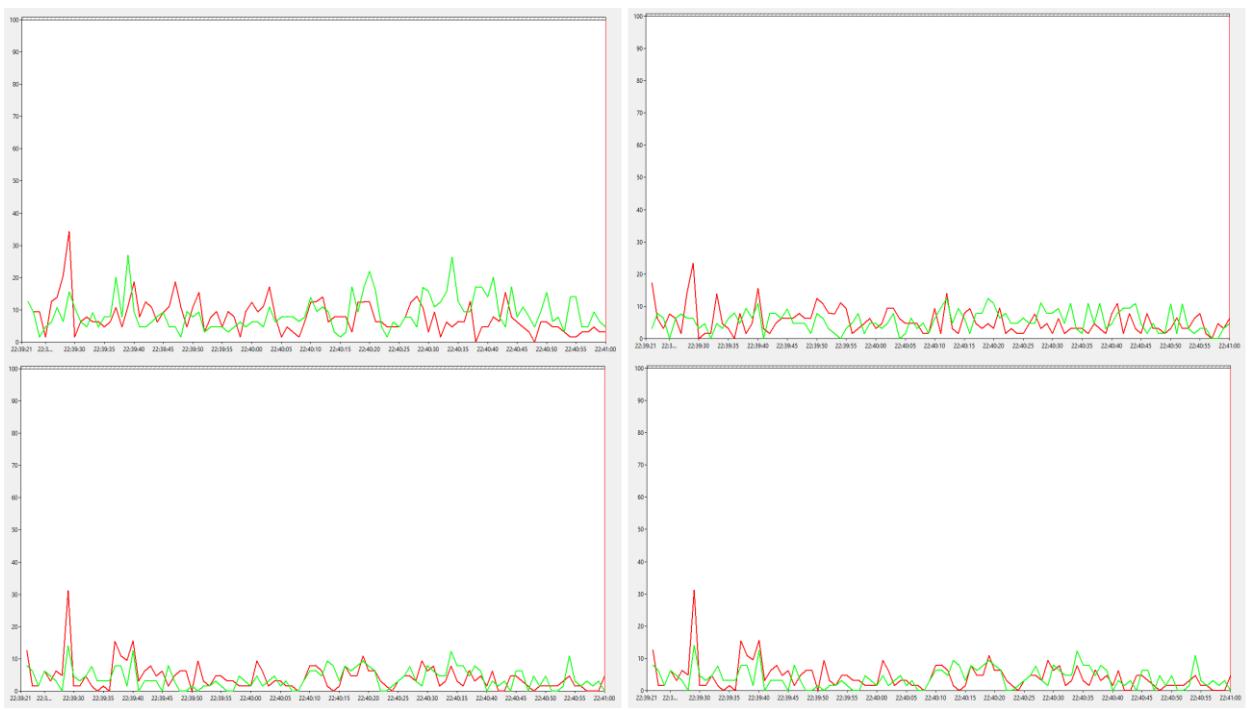


Рисунок 4.10 – результаты исследования режима работы ядер процессора

Вывод: ядра процессора работают практически одинаковое время в разных режимах.

Контрольные вопросы

1. Назначение счетчиков производительности.

В ОС MS Windows имеется ряд объектов производительности, обычно соответствующих аппаратным компонентам, таким как память, процессоры, внешние устройства и т. д. Каждый объект производительности предоставляет счетчики, которые собирают данные производительности (performance counters).

Счетчик производительности представляет собой механизм, с помощью которого в MS Windows производится сбор сведений о производительности различных системных ресурсов. В MS Windows имеется предопределенный набор счетчиков производительности, с которыми можно взаимодействовать. Каждый счетчик относится к определенной области функций системы.

2. Категории и экземпляры счетчиков.

Счетчик производительности следит за поведением объектов производительности компьютера. Эти объекты включают в себя физические компоненты, такие как процессоры, диски, память и системные объекты, такие как процессы, потоки и задания.

Системные счетчики, относящиеся к одному и тому же объекту производительности, группируются в категории, отражающие их общую направленность. При создании экземпляра компонента PerformanceCounter сначала указывается категория, с которой будет взаимодействовать компонент, затем внутри этой категории выбирается счетчик, с которым будет осуществляться взаимодействие.

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

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

3. Управление параметрами создаваемых графиков (масштаб, цвет и толщина линий).

Управление формой представления графиков производится с помощью окна свойств, которое открывается с помощью кнопки Свойства. Диапазон значений вертикальной шкалы задается в окне Свойства: системный монитор

4. Влияние активности окна приложения на текущий приоритет его потоков.

При перемещении окна приложения по экрану текущий приоритет изменяется только у потока с номером 0.

Цель работы: Практическое знакомство с командами, используемыми для контроля использования ресурсов и виртуальной файловой системой.

Ход работы:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания.
3. Ответить на контрольные вопросы.

Задание:

1. Вывести список всех процессов в системе.
2. Вывести дерево процессов.
3. С помощью команды top получить список процессов, потребляющих наибольшее количество процессорного времени.
4. Найти 2 процесса, имеющих более ДВУХ потоков.
5. Используя команду top, изменить приоритеты 2 процессов.
6. Получить список открытых файлов текущего пользователя.
7. Получить текущее состояние системной памяти.
8. Получить справку об использовании дискового пространства.
9. Вывести информацию о каком-либо процессе, используя содержимое каталога /proc.
10. Вывести информацию о процессоре ПК, используя содержимое каталога /proc.
11. Вывести список модулей, используемых в настоящий момент ядром ОС.

Описание выполнения работы

1. Вывести список всех процессов в системе.

Для вывода списка всех выполняющихся на компьютере в текущий момент процессах используется команда: `ps aux`

Значения используемых опций: a - all – процессы всех пользователей; u – ориентированная на пользователей (отображение информации о владельце); x – процессы, не контролируемые ttys.

Результат выполнения данной команды, представлен на рисунке 5.1.

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

student@student-VirtualBox:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.5  0.5 167480 11644 ?
root         2  0.0  0.0     0     0 ?
root         3  0.0  0.0     0     0 ?
root         4  0.0  0.0     0     0 ?
root         6  0.0  0.0     0     0 ?
root         8  0.0  0.0     0     0 ?
root         9  0.0  0.0     0     0 ?
root        10  0.0  0.0     0     0 ?
root        11  0.0  0.0     0     0 ?
root        12  0.0  0.0     0     0 ?
root        14  0.0  0.0     0     0 ?
root        15  0.0  0.0     0     0 ?
root        16  0.0  0.0     0     0 ?
root        17  0.0  0.0     0     0 ?
root        18  0.0  0.0     0     0 ?
root        19  0.0  0.0     0     0 ?
root        20  0.0  0.0     0     0 ?
root        21  0.0  0.0     0     0 ?
root        22  0.0  0.0     0     0 ?
root        23  0.0  0.0     0     0 ?
root        24  0.0  0.0     0     0 ?
root       116  0.0  0.0     0     0 ?
root       117  0.0  0.0     0     0 ?
root       118  0.0  0.0     0     0 ?
```

Рисунок 5.1 – Фрагмент вывода списка процессов системы

2. Вывести дерево процессов.

Для построения дерева процессов используются команды (результат работы каждой из них идентичен):

`pstree`, `psejH`, `psaxjt`

Результат выполнения первой из приведенных выше команд, представлен на рисунке 5.2.

```
student@student-VirtualBox:~$ pstree
systemd─ ModemManager─2*[{ModemManager}]
      └─ NetworkManager─2*[{NetworkManager}]
        ├ accounts-daemon─2*[{accounts-daemon}]
        └─ acpid
        ├ avahi-daemon─ avahi-daemon
        ├ colord─2*[{colord}]
        └─ cron
        ├ cups-browsed─2*[{cups-browsed}]
        ├ cupsd─dbus
        └─ dbus-daemon
        ├ gdm3─gdm-session-wor─gdm-x-session─Xorg─{Xorg}
        │   └─ gnome-session-b─ssh-agent
        │       └─2*[{gnome+}]
        │       └─2*[{gdm-x-session}]
        └─2*[{gdm3}]
        ├ gnome-keyring-d─3*[{gnome-keyring-d}]
        ├ ibus-daemon─ibus-engine-sim─2*[{ibus-engine-sim}]
        │   ├ ibus-extension─3*[{ibus-extension-}]
        │   ├ ibus-memconf─2*[{ibus-memconf}]
        │   └─ibus-ui-gtk3─3*[{ibus-ui-gtk3}]
        └─2*[{ibus-daemon}]
        ├ ibus-x11─2*[{ibus-x11}]
        └─2*[{kerneloops}]
        ├ networkd-dispat
        └─ polkitd─2*[{polkitd}]
        ├ rsyslogd─3*[{rsyslogd}]
        └─ rtkit-daemon─2*[{rtkit-daemon}]
```

Рисунок 5.2 – Фрагмент вывода списка процессов системы в виде дерева

3. С помощью команды top получить список 5 процессов, потребляющих наибольшее количество процессорного времени.

Для того чтобы вывести список процессов, отсортированных по потреблению наибольшего количества процессорного времени, необходимо воспользоваться командой: `top -o %CPU`.

Однако, данная команда выведет полный список всех процессов. Для того чтобы выбрать только первые 5 процессов, необходимо воспользоваться командами `tail` и `head`, которые отсекают от выходного результата команды строки с конца и с начала соответственно. В качестве параметра, определяющего количество отсекаемых строк, используется параметр `n`.

Таким образом, получаем команду:

```
top -o %CPU | tail -n +7 | head -n 6
```

Результат выполнения данной команды, представлен на рисунке 5.3.

student@student-VirtualBox:~\$ top -o %CPU tail -n+7 head -n 6											
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1002	student	20	0	323796	9924	8376	S	6,7	0,5	0:00.62	ibus-d+
1	root	20	0	167480	11644	8604	S	0,0	0,6	0:03.61	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.00	kthrea+
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_pa+

Рисунок 5.3 – Список 5 процессов, потребляющих наибольшее количество процессорного времени

4. Найти 2 процесса, имеющих более ДВУХ потоков.

Для того чтобы выбрать все процессы воспользуемся ключом «-e», который выводим список всех процессов. Получится команда:

```
ps -e
```

Для того чтобы получить необходимые нам колонки воспользуемся ключом «-o», параметры которого – необходимые для вывода колонки: pid – id процесса, cmd – команда запуска процессора, nlwp (Number Light-Weight Process) – число легковесных процессов в операционной системе, так как этот параметр позволяет пользователю использовать собственный формат ввода:

```
ps -e -o pid,cmd,nlwp
```

Отсортируем полученный результат по убыванию количества потоков

```
ps -e -o pid,cmd,nlwp --sort=-nlwp
```

Для того, чтобы получить только первые 2 процесса воспользуемся командой head -n 3. В конечном итоге получим команду:

```
ps -e -o pid,cmd,nlwp --sort=-nlwp | head -n 3
```

Результат выполнения данной команды, представлен на рисунке 5.4.

```
student@student-VirtualBox:~$ ps -e -o pid,cmd,nlwp --sort=-nlwp |head -n 3
    PID CMD                               NLWP
  2003 /usr/lib/firefox/firefox -n      51
  2081 /usr/lib/firefox/firefox -c      19
student@student-VirtualBox:~$
```

Рисунок 5.4 – Два процесса, имеющих более ДВУХ потоков

5. Используя команду `top`, изменить приоритеты 2 процессов. Изменим приоритеты двух процессов с PID 22452 и PID 2324.

Для просмотра информации об этих двух процессах воспользуемся командой `top: top -p22452,2324`

Результат выполнения данной команды, представлен на рисунке 5.5.

Как видим, значение приоритета обоих процессов равно 20. Повысим приоритет его у обоих процессов на 1. Это можно сделать интерактивно из утилиты `top`. Для этого запускаем утилиту `top` и нажимаем клавишу `<R>`. Нам будет предложено ввести уровень приоритета и PID процесса, аналогично вводу в режиме команд (рисунки 5.6 и 5.7).

```
top - 09:26:07 up 4:15, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.0 us, 0.8 sy, 0.0 ni, 98.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018084 total, 74500 free, 941756 used, 1001828 buff/cache
KiB Swap: 2094076 total, 2089712 free, 4364 used. 836532 avail Mem
Renice PID 22452 to value -1
  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
22452 student   20   0  690184  48536  34560 S  1.3  2.4   0:28.11 gnome-sys+
  2324 student   20   0 1629452 217140  66868 S  0.3 10.8   1:41.10 compiz
```

Рисунок 5.5 – Информация о процессах

```
top - 09:26:07 up 4:15, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.0 us, 0.8 sy, 0.0 ni, 98.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018084 total, 74500 free, 941756 used, 1001828 buff/cache
KiB Swap: 2094076 total, 2089712 free, 4364 used. 836532 avail Mem
Renice PID 22452 to value -1
  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
22452 student   20   0  690184  48536  34560 S  1.3  2.4   0:28.11 gnome-sys+
  2324 student   20   0 1629452 217140  66868 S  0.3 10.8   1:41.10 compiz
```

Рисунок 5.6 – Изменение приоритета процесса с PID 22452

```

top - 09:26:37 up 4:16, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.8 us, 0.7 sy, 0.0 ni, 98.5 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018084 total, 74360 free, 941872 used, 1001852 buff/cache
KiB Swap: 2094076 total, 2089712 free, 4364 used. 836408 avail Mem
Renice PID 2324 to value -1

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
22452	student	19	-1	690184	48536	34560	S	1.7	2.4	0:28.62	gnome-sys+
2324	student	20	0	1629444	217140	66868	S	0.3	10.8	1:41.30	compiz

Рисунок 5.7 – Изменение приоритета процесса с PID 2324

Теперь мы можем видеть, что приоритет обоих процессов изменен и равен значению 19 (рисунок 5.8.).

```

top - 09:26:59 up 4:16, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 2 total, 0 running, 2 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.3 us, 0.7 sy, 0.0 ni, 98.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2018084 total, 73740 free, 942492 used, 1001852 buff/cache
KiB Swap: 2094076 total, 2089712 free, 4364 used. 835788 avail Mem


```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
22452	student	19	-1	690184	48536	34560	S	1.7	2.4	0:29.10	gnome-sys+
2324	student	19	-1	1630016	217712	66868	S	1.0	10.8	1:41.64	compiz

Рисунок 5.8 – Информация о процессах

6. Получить список открытых файлов текущего пользователя

Команда `lsof` (ListOpenFiles) без параметров выводит полный список открытых файлов. При этом пользователь-администратор получит несколько тысяч строк текста.

Для получения списка файлов, открытых конкретным пользователем, служит команда: `lsof -u имя_пользователя`

Таким образом, в нашем случае (имя пользователя: `student`) необходимо выполнить команду:

`lsof -u student`

Результат выполнения данной команды, представлен на рисунке 5.9.

```

ipe      2809 student   6r    FIFO          0,13     0t0    48635 p
lsof     2810 student cwd      DIR          8,5      4096    295337 /
/home/student
lsof     2810 student rtd      DIR          8,5      4096    2 / 
lsof     2810 student txt      REG          8,5    175744   787140 /
/usr/bin/lsof
lsof     2810 student mem     REG          8,5    51832   797987 /
/usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
lsof     2810 student mem     REG          8,5   14537584   794276 /
/usr/lib/locale/locale-archive
lsof     2810 student mem     REG          8,5    157224   798128 /
/usr/lib/x86_64-linux-gnu/libpthread-2.31.so
lsof     2810 student mem     REG          8,5    18816   797360 /
/usr/lib/x86_64-linux-gnu/libdl-2.31.so
lsof     2810 student mem     REG          8,5    584392   798071 /
/usr/lib/x86_64-linux-gnu/libpcre2-8.so.0.9.0
lsof     2810 student mem     REG          8,5   2029224   797225 /
/usr/lib/x86_64-linux-gnu/libc-2.31.so
lsof     2810 student mem     REG          8,5   163200   798225 /
/usr/lib/x86_64-linux-gnu/libselinux.so.1
lsof     2810 student mem     REG          8,5   191472   797012 /
/usr/lib/x86_64-linux-gnu/ld-2.31.so
lsof     2810 student 4r    FIFO          0,13     0t0    48634 p
ipe
lsof     2810 student 7w    FIFO          0,13     0t0    48635 p
ipe
student@student-VirtualBox:~$ █

```

Рисунок 5.9— Фрагмент списка открытых файлов пользователя

7. Получить текущее состояние системной памяти

Текущее состояние системной памяти позволяет получить команда `free`.

По умолчанию все выводимые значения представлены в килобайтах. Значения в мегабайтах позволяет получить опция `-m`.

Таким образом, получаем команду: `free -m`

Результат выполнения данной команды, представлен на рисунке 5.10.

```

student@student-VirtualBox:~$ free -m
      всего      занято      свободно      общая  буф./врем.      дост
упно
Память:       1987        1085         119          27        782        732
Подкачка:       687          0         686
student@student-VirtualBox:~$ █

```

Рисунок 5.10— Текущее состояние системной памяти

8. Получить справку об использовании дискового пространства.

Команда `df` выводит данные об объеме доступного дискового пространства (в килобайтах). Опция `-h` улучшает восприятие результатов (данные выводятся в табличной форме).

Команда du дает возможность узнать объем дисковой памяти, занимаемой каталогами и файлами.

Таким образом, получаем команду: df -h

Результат выполнения данной команды, представлен на рисунке 5.11.

```
student@student-VirtualBox:~$ df -h
Файл.система    Размер Использовано   Дост Использовано% Смонтировано в
udev           967M      0  967M          0% /dev
tmpfs          199M     1,3M 198M          1% /run
/dev/sda5       15G     6,3G 7,3G         47% /
tmpfs          994M     11M 984M          2% /dev/shm
tmpfs          5,0M     4,0K 5,0M          1% /run/lock
tmpfs          994M      0 994M          0% /sys/fs/cgroup
/dev/loop0       55M     55M  0          100% /snap/core18/1705
/dev/loop2       241M    241M  0          100% /snap/gnome-3-34-1804/24
/dev/loop4       28M     28M  0          100% /snap/snapd/7264
/dev/loop3       50M     50M  0          100% /snap/snap-store/433
/dev/loop1       63M     63M  0          100% /snap/gtk-common-themes/
1506
/dev/sda1       511M     4,0K 511M          1% /boot/efi
tmpfs          199M     36K 199M          1% /run/user/1000
student@student-VirtualBox:~$
```

Рисунок 5.11 – Справка об использовании дискового пространства

9. Вывести информацию о каком-либо процессе, используя содержимое каталога /proc

В каталоге /proc содержится информация о каждом процессе, приведем информацию об основных файлах в этом каталоге.

В качестве процесса возьмем процесс с PID 1.

cmdline: этот (псевдо-) файл содержит полную командную строку, использованную для вызова процесса.

Результат выполнения данной команды cat /proc/1/cmdline , представлен на рисунке 5.12.

```
student@student-VirtualBox:~$ cat /proc/1/cmdline
/sbin/initstudent@student-VirtualBox:~$
```

Рисунок 5.12 – Информация о процессе с PID 1, используя содержимое каталога /proc

cwd: эта символическая ссылка указывает на текущий рабочий каталог процесса (как следует из её имени).

Получим адрес ссылки при помощи команды: `readlink -f /proc/1/cwd`

Результат выполнения данной команды, представлен на рисунке 5.13.

```
student@student-VirtualBox:~$ readlink -f /proc/1/cwd
student@student-VirtualBox:~$
```

Рисунок 5.13 – Текущий рабочий каталог процесса с PID 1

`environ`: этот файл содержит все переменные окружения, определенные для этого процесса, в виде ПЕРЕМЕННАЯ = значение. Как и в `cmdline` вывод вообще не отформатирован: нет разделителей строк для отделения различных переменных, и в конце нет разделителя строки.
`cat/proc/1/environ`

Результат выполнения данной команды, представлен на рисунке 5.14.

```
student@student-VirtualBox:~$ cat /proc/1/environ
cat: /proc/1/environ: Отказано в доступе
student@student-VirtualBox:~$ sudo cat /proc/1/environ
[sudo] пароль для student:
HOME=/init=/sbin/initNETWORK_SKIP_ENSLAVED=TERM=linuxBOOT_IMAGE=/boot/vmlinuz-5.4.0-33-genericdrop_caps=PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binPWD=/rootmnt=/rootstudent@student-VirtualBox:~$
```

Рисунок 5.14 – Все переменные окружения, определенные для процесса с PID 1

`exe`: эта символическая ссылка указывает на исполняемый файл, соответствующий запущенному процессу.

Получим адрес ссылки при помощи команды: `readlink -f /proc/1/exe`

Результат выполнения данной команды, представлен на рисунке 5.15.

```
student@student-VirtualBox:~$ readlink -f /proc/1/exe
student@student-VirtualBox:~$
```

Рисунок 5.15 – Исполняемый файл, соответствующий запущенному процессу с PID 1

`fd`: этот подкаталог содержит список файловых дескрипторов, открытых в данный момент процессом.

Результат выполнения команды `ls /proc/1/fd`, представлен на рисунке 5.16.

```
student@student-VirtualBox:~$ ls /proc/1/fd
ls: невозможно открыть каталог '/proc/1/fd': Отказано в доступе
student@student-VirtualBox:~$ sudo ls /proc/1/fd
0   107  117  126  135  144  159  21   30   4    49   58   67   77   86   95
1   108  118  127  136  145  16   22   31   40   5    59   68   78   87   96
10  109  119  128  137  146  160  23   32   41   50   6    69   79   88   97
100 11   12   129  138  147  161  24   33   42   51   60   7    8    89   98
101 110  120  13   139  15   162  25   34   43   52   61   70   80   9   99
102 111  121  130  14   152  17   26   35   44   53   62   71   81   90
103 112  122  131  140  153  18   27   36   45   54   63   72   82   91
104 113  123  132  141  154  19   28   37   46   55   64   73   83   92
105 114  124  133  142  155  2    29   38   47   56   65   74   84   93
106 115  125  134  143  156  20   3    39   48   57   66   75   85   94
student@student-VirtualBox:~$
```

Рисунок 5.16— Список файловых дескрипторов, открытых в данный момент процессом с PID 1

`maps`: когда вы выводите содержимое этого именованного канала (при помощи команды `cat`, например), вы можете увидеть части адресного пространства процесса, которые в текущий момент распределены для файла. Вот эти поля (слева направо): адресное пространство, связанное с этим распределением; разрешения, связанные с этим распределением; смещение от начала файла, где начинается распределение; старший и младший номера (в шестнадцатеричном виде) устройства, на котором находится распределенный файл; номер inode файла; и, наконец, имя самого файла. Результат выполнения данной команды `cat /proc/1/maps`, представлен на рисунке 5.17.

```
student@student-VirtualBox:~$ cat /proc/1/maps
cat: /proc/1/maps: Отказано в доступе
student@student-VirtualBox:~$ sudo cat /proc/1/maps
56130efc9000-56130effb000 r--p 00000000 08:05 796641          /usr/l
ib/systemd/systemd
56130effb000-56130f0b4000 r-xp 00032000 08:05 796641          /usr/l
ib/systemd/systemd
56130f0b4000-56130f10a000 r--p 000eb000 08:05 796641          /usr/l
ib/systemd/systemd
56130f10a000-56130f150000 r--p 00140000 08:05 796641          /usr/l
ib/systemd/systemd
56130f150000-56130f151000 rw-p 00186000 08:05 796641          /usr/l
ib/systemd/systemd
5613100dd000-5613102f0000 rw-p 00000000 00:00 0             [heap]
7f0128000000-7f0128021000 rw-p 00000000 00:00 0
7f0128021000-7f012c000000 ---p 00000000 00:00 0
7f0130000000-7f0130021000 rw-p 00000000 00:00 0
7f0130021000-7f0134000000 ---p 00000000 00:00 0
7f013444e000-7f013444f000 ---p 00000000 00:00 0
7f013444f000-7f0134c4f000 rw-p 00000000 00:00 0
7f0134c4f000-7f0134c50000 ---p 00000000 00:00 0
7f0134c50000-7f0135450000 rw-p 00000000 00:00 0
7f0135450000-7f013545f000 r--p 00000000 08:05 797863          /usr/l
ib/x86_64-linux-gnu/libm-2.31.so
7f013545f000-7f0135506000 r-xp 0000f000 08:05 797863          /usr/l
ib/x86_64-linux-gnu/libm-2.31.so
7f0135506000-7f013559d000 r--p 000b6000 08:05 797863          /usr/l
ib/x86_64-linux-gnu/libm-2.31.so
7f013559d000-7f013559e000 r--p 0014c000 08:05 797863          /usr/l
```

Рисунок 5.17– Фрагмент адресного пространства процесса с PID 1

root: эта символическая ссылка указывает на корневой каталог, используемый процессом. Обычно это будет `readlink -f /proc/1/root`. Результат выполнения данной команды, представлен на рисунке 5.18.

```
student@student-VirtualBox:~$ readlink -f /proc/1/root
student@student-VirtualBox:~$
```

Рисунок 5.18– Корневой каталог, используемый процессом с PID 1

`status`: этот файл содержит разнообразную информацию о процессе: имя исполняемого файла, его текущее состояние, его PID и PPID, его реальные и эффективные UID и GID, его использование памяти и другие данные.

Результат выполнения команды `cat /proc/1/status`, представлен на рисунке 5.19.

```
student@student-VirtualBox:~$ cat /proc/1/status
Name:    systemd
Umask:   0000
State:   S (sleeping)
Tgid:    1
Ngid:    0
Pid:    1
PPid:   0
TracerPid:      0
Uid:     0          0          0          0
Gid:     0          0          0          0
FDSize: 256
Groups:
NSTgid: 1
NSpid:  1
NSpgid: 1
NSsid:  1
VmPeak:  233016 kB
VmSize:  167480 kB
VmLck:    0 kB
VmPin:    0 kB
VmHWM:   11644 kB
VmRSS:   11640 kB
RssAnon:      3036 kB
RssFile:      8604 kB
RssShmem:     0 kB
VmData:   19168 kB
VmStk:    132 kB
VmExe:    940 kB
```

Рисунок 5.19—Фрагмент информации о процессе с PID 1

10. Вывести информацию о процессоре ПК, используя содержимое каталога /proc

/proc/cpuinfo: этот файл содержит, как видно из его имени, информацию о процессорах машины: `cat /proc/cpuinfo`

Результат выполнения данной команды, представлен на рисунке 5.20.

```
student@student-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id      : AuthenticAMD
cpu family     : 21
model          : 2
model name     : AMD FX-4330 Quad-Core Processor
stepping        : 0
microcode      : 0x6000626
cpu MHz         : 4018.036
cache size     : 2048 KB
physical id    : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid : 0
fpu             : yes
fpu_exception   : yes
cpuid level    : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
                  pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt rdtscp lm co
                  nstant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid tsc_known_freq pnpi pclmu
                  lqdq monitor ssse3 cx16 sse4_1 sse4_2 x2apic popcnt aes xsave avx hypervisor la
                  hf_lm cr8_legacy abm sse4a misalignsse 3dnowprefetch ssbd vmmcall arat
bugs            : fxsave_leak sysret_ss_attrs null_seg spectre_v1 spectre_v2 sp
ec_store_bypass
bogomips        : 8036.07
TLB size        : 1536 4K pages
```

Рисунок 5.20— Фрагмент информации о процессоре ПК

Вывести список модулей, используемых в настоящий момент ядром `/proc/modules`: этот файл содержит список модулей, используемых ядром в настоящий момент, вместе со счетчиком использования каждого из модулей. Эта информация используется командой `lsmod`, которая отображает её в более удобной для чтения форме. Для вывода воспользуемся командой `cat`.

Результат выполнения команды `cat/proc/modules`, представлен на рисунке 5.21.

```
student@student-VirtualBox:~$ cat /proc/modules
nls_iso8859_1 16384 1 - Live 0x0000000000000000
snd_intel8x0 45056 2 - Live 0x0000000000000000
snd_ac97_codec 131072 1 snd_intel8x0, Live 0x0000000000000000
edac_mce_amd 32768 0 - Live 0x0000000000000000
ac97_bus 16384 1 snd_ac97_codec, Live 0x0000000000000000
snd_pcm 106496 2 snd_intel8x0,snd_ac97_codec, Live 0x0000000000000000
crct10dif_pclmul 16384 1 - Live 0x0000000000000000
snd_seq_midi 20480 0 - Live 0x0000000000000000
ghash_clmulni_intel 16384 0 - Live 0x0000000000000000
snd_seq_midi_event 16384 1 snd_seq_midi, Live 0x0000000000000000
snd_rawmidi 36864 1 snd_seq_midi, Live 0x0000000000000000
joydev 24576 0 - Live 0x0000000000000000
aesni_intel 372736 0 - Live 0x0000000000000000
snd_seq 69632 2 snd_seq_midi,snd_seq_midi_event, Live 0x0000000000000000
crypto_simd 16384 1 aesni_intel, Live 0x0000000000000000
cryptd 24576 2 ghash_clmulni_intel,crypto_simd, Live 0x0000000000000000
glue_helper 16384 1 aesni_intel, Live 0x0000000000000000
snd_seq_device 16384 3 snd_seq_midi,snd_rawmidi,snd_seq, Live 0x0000000000000000
0
snd_timer 36864 2 snd_pcm,snd_seq, Live 0x0000000000000000
snd_90112 11 snd_intel8x0,snd_ac97_codec,snd_pcm,snd_rawmidi,snd_seq,snd_seq_de-
vice,snd_timer, Live 0x0000000000000000
soundcore 16384 1 snd, Live 0x0000000000000000
input_leds 16384 0 - Live 0x0000000000000000
vboxguest 348160 0 - Live 0x0000000000000000 (0)
mac_hid 16384 0 - Live 0x0000000000000000
serio_raw 20480 0 - Live 0x0000000000000000
sch_fq_codel 20480 2 - Live 0x0000000000000000
```

Рисунок 5.21 – Фрагмент списка модулей, используемых в настоящий момент ядром ОС

Контрольные вопросы

1. Команды вывода списка процессов.

Для вывода списка всех выполняющихся на компьютере в текущий момент процессах используется команда: ps aux

Значения используемых опций: a - all – процессы всех пользователей; u – ориентированная на пользователей (отображение информации о владельце); x – процессы, не контролируемые ttys.

Полезные ключи: -e – вывод сведений обо всех процессах и o – пользовательский вывод.

2. Команда получения списка потоков.

Для получения информации о потоках заданного процесса используется опция -L, например ps -fLC swriter.bin выводит список потоков приложения writer Open Office.

3. Команда для завершения приложений.

Завершение процесса выполняется командой kill сигнал PID.

Сначала процессу посыпается сигнал -15. Если это не помогает, 129, используется крайняя мера -посыпается сигнал -9.

4. Состояния процесса Linux.

Каждый из каталогов содержит одинаковые пункты, краткое описание некоторых из них:

cmdline: этот (псевдо-) файл содержит полную командную строку, использованную для вызова процесса 134 программой и ее аргументами нет пробелов, а в конце строки нет разделителя строки. Чтобы просмотреть его, вы можете использовать: perl -ple 's,\00, ,g' cmdline.

cwd: эта символическая ссылка указывает на текущий рабочий каталог процесса (следует из имени).environ: этот файл содержит все переменные окружения, определенные для этого процесса, в виде ПЕРЕМЕННАЯ=значение. Как и в cmdline вывод вообще не отформатирован: нет разделителей строк для отделения различных

переменных, и в конце нет разделителя строки. Единственным решением для его просмотра будет: perl -pl -e 's,\00,\n,g' environ exe: эта символическая ссылка указывает на исполняемый файл, соответствующий запущенному процессу.

fd: этот подкаталог содержит список файловых дескрипторов, открытых в данный момент процессом.

maps: когда вы выводите содержимое этого именованного канала (при помощи команды cat, например), вы можете увидеть части адресного пространства процесса, которые в текущий момент распределены для файла. Вот эти поля (слева направо): адресное пространство, связанное с этим распределением; разрешения, связанные с этим распределением; смещение от начала файла, где начинается распределение; старший и младший номера (в шестнадцатиричном виде) устройства, на котором находится распределенный файл; номер inode файла; и, наконец, имя самого файла.

root: эта символическая ссылка указывает на корневой каталог, используемый процессом. Обычно это будет /.

status: этот файл содержит разнообразную информацию о процессе: имя исполняемого файла, его текущее состояние, его PID и PPID, его реальные и эффективные UID и GID, его использование памяти и другие данные.

5. Получение информации о потоках процесса.

Как известно, процесс может иметь параллельно выполняющиеся потоки (threads) или облегченные процессы (LWP, Light Weight Process). Для получения информации о потоках заданного процесса используется опция – L, например ps -fLC swriter.bin выводит список потоков приложения writer Open Office. Процессы, использующие более одного потока – редактор звуковых файлов audacity и soffice.bin, а также демоны (службы в по терминологии Windows). Как указано выше, многопоточные процессы помечено символом l в колонке состояния.

6. Примеры многопоточных процессов.

Программные потоки являются базовым элементом многозадачного программного окружения. Программный поток может быть описан как среда выполнения процесса; поэтому каждый процесс имеет как минимум один программный поток. Многопоточность предполагает наличие нескольких параллельно работающих (на многопроцессорных системах) и обычно синхронизируемых сред выполнения процесса.

Программные потоки имеют свои идентификаторы (thread ID) и могут выполняться независимо друг от друга. Они делят между собой одно адресное пространство процесса и используют эту особенность в качестве преимущества, позволяющего не использовать каналы IPC (систем меж процессного взаимодействия - разделяемой памяти, каналов и других систем) для обмена данными. Программные потоки процесса могут взаимодействовать-например, независимые потоки могут получать/изменять значение глобальной переменной. Эта модель взаимодействия исключает лишние затраты ресурсов на вызовы IPC на уровне ядра. Поскольку потоки работают в едином адресном пространстве, переключения контекста для потока быстры и не ресурсоемки.

Многопоточные процессы помечено символом 1 в колонке состояния ps.

Примером может служить браузер Firefox.

7. Необходимость использования потоков.

Выделим основные преимущества использования потоков:

Потоки удобно использовать при необходимости выполнения в процессе нескольких действий сразу (пример: одновременная обработка на сервере запросов нескольких пользователей)

Ускорение работы приложений, использующих ввод, обработку и вывод данных за счет возможности распределения этих операций по отдельным потокам. Это дает возможность не прекращать выполнение программы во время возможных простоев из-за ожидания при чтении/записи данных

Как правило, переключение между потоками происходит быстрее и требует

меньших затрат системных ресурсов, по сравнению с переключением между процессами

8. Процессы-зомби: как они появляются, как их найти и что с ними делать?

Потоки удобно использовать при необходимости выполнения в процессе нескольких действий сразу (пример: одновременная обработка на сервере запросов нескольких пользователей).

Как правило, переключение между потоками происходит быстрее и требует меньших затрат системных ресурсов, по сравнению с переключением между процессами.

9. Содержимое вывода команды top.

Команда top представляет динамически обновляемые сведения о процессах и о том, какой объем системных ресурсов использует каждый из них.

В первых пяти строках команда top отображает подробную информацию о системе, а затем описывает каждый выполняющийся процесс. Выходные данные сортируются по уровню загрузки ЦП данным процессом.

В первой строке программа сообщает текущее время, время работы системы (1 час 15 мин), количество зарегистрированных (login) пользователей (3 users), общая средняя загрузка системы (load average). Общей средней загрузкой системы называется среднее число процессов, находящихся в состоянии выполнения (R) или в состоянии ожидания (D). Общая средняя загрузка измеряется каждые 1, 5 и 15 минут.

Во второй строке вывода программы top сообщается, что в списке процессов находятся 132 процесса, из них 131 спит (состояние готовности или ожидания), 1 выполняется (на виртуальной машине только 1 процессор), 0 процессов зомби и 0 остановленных процессов.

В третьей-пятой строках приводится информация о загрузке процессора CPU в режиме пользователя и системном режиме, использования памяти и файла подкачки.

В таблице отображается различная информация о процессе. Рассмотрим

колонки PID (идентификатор процесса), USER (пользователь, запустивший процесс), S (состояние процесса) и COMMAND (команда, которая была введена для запуска процесса).

Колонка S может содержать следующие значения:

R – процесс выполняется или готов к выполнению (состояние готовности); помечено символом 1 в колонке состояния ps. Примером может служить браузер Firefox.

10. Как получить информацию о процессах системы, используя файловую систему /proc?

Каталог /proc содержит всю информацию о всех процессах в системе, для доступа к информации о процессе используется следующая команда:

/proc/[pid]/stat

Где на месте stat может быть любой другой тип возвращаемой информации.

11. Команды для получения информации об открытых файлах.

Команда lsof (List open files) без параметров выводит полный список открытых файлов. Пользователь-администратор получит несколько тысяч строк текста.

Для получения списка файлов, открытых конкретным пользователем, служит команда lsof -u имя_пользователя

12. Получение информации о состоянии системной памяти.

Текущее состояние системной памяти позволяет получить команда

По умолчанию все значения представлены в килобайтах. Значения в M позволяет получить опция –m.

13. Получение информации об использовании дискового пространства.

Команда df выводит данные об объеме доступного дискового пространства (в Кбайтах). Опция –h улучшает восприятие результатов.

Команда du дает возможность узнать объем дисковой памяти, занимаемой каталогами и файлами.

14. Назначение файловой системы /proc.

Файловая система /proc является механизмом для ядра и его модулей, позволяющим посыпать информацию процессам (отсюда и название /proc). С помощью этой виртуальной файловой системы можно работать с внутренними структурами ядра, получать полезную информацию о процессах и изменять установки (меняя параметры ядра) на лету. Файловая система /proc располагается в памяти в отличие от других файловых систем, которые располагаются на диске.

Цель работы: практическое знакомство со способами эффективной обработки текста при помощи интерфейса командной строки и набора стандартных утилит.

Ход работы:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания.
3. Ответить на контрольные вопросы.

Задание:

1. Используя утилиты hexdump и strings, вывести на экран содержимое одного из перечисленных ниже файлов из каталога /bin. Позиция файла для распечатки определяется номером бригады. Имена файлов для выполнения задания 1: tar, sort, sed, ping, vi, **unlink**, uname, touch, sleep, sty.
2. Подсчитать общее количество файлов (каталогов) в одном из перечисленных ниже каталогов. Каталог для подсчета количества определяется номером бригады. Имена каталогов для выполнения задания 2: /bin, /etc, /lib, /proc, /usr, **/var**, /dev, /sbin, /sys, /root.
3. Найти общее количество процессов, выполняющихся в системе в данный момент.
4. Вывести список выполняющихся процессов, в именах которых присутствует слово manager и отсутствует слово grep.
5. Создать текстовый файл, содержащий набор строк вида:

123
178
176
755
713
873

С помощью утилиты grep найти строки, в которых есть цифра 7, после которой находится одна из цифр — 1, 3 или 5.

6. Создать текстовый файл, содержащий набор строк вида:

starfish
samscripter
stellar

microsrar
ascender
sacrifice
scalar

С помощью утилиты grep найти строки, начинающиеся на букву s и заканчивающиеся на букву r.

7. Создать текстовый файл, содержащий простейшие адреса электронной почты вида username@website.com.

С помощью утилиты grep найти строки, содержащие правильные простейшие адреса. Проверить возможность использования более сложного регулярного выражения для распознавания адресов, содержащих другие допустимые символы.

8. На произвольном примере продемонстрировать работу утилиты tr.

9. Создать текстовый файл, содержащий допустимые и недопустимые IP-адреса, например

127.0.0.1
255.255.255.255
12.34.56
123.256.0.0
1.23.099.255
0.79.378.111

С помощью утилиты grep и руководства man найти строки, содержащие допустимые четырехбайтовые IP адреса.

10. Создать текстовый файл, содержащий корректные и некорректные номера телефонов ведомственной АТС объемом 399 номеров, номера с 000 до 399 – корректные, 0, 400, 900 – некорректные. С помощью утилиты grep и руководства man найти строки, содержащие допустимые четырехбайтовые IP адреса.

.

Описание выполнения работы

1. Используя утилиты hexdump и strings, выведем на экран содержимое файла unlink из каталога /bin, как показано на рисунках 6.1, 6.2.

```
student@student-VirtualBox:~$ hexdump -C /bin/touch
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00 |.ELF.....
00000010  03 00 3e 00 01 00 00 00  70 42 00 00 00 00 00 00 |...>....pB....
00000020  40 00 00 00 00 00 00 00  f8 81 01 00 00 00 00 00 |@.....
00000030  00 00 00 00 40 00 38 00  0d 00 40 00 1e 00 1d 00 |....@.8...@...
00000040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00 |.....@.....
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00 |@.....@.....
00000060  d8 02 00 00 00 00 00 00  d8 02 00 00 00 00 00 00 |.....
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00 |.....
00000080  18 03 00 00 00 00 00 00  18 03 00 00 00 00 00 00 |.....
00000090  18 03 00 00 00 00 00 00  1c 00 00 00 00 00 00 00 |.....
000000a0  1c 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00 |.....
000000b0  01 00 00 00 04 00 00 00  00 00 00 00 00 00 00 00 |.....
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |.....
000000d0  c0 29 00 00 00 00 00 00  c0 29 00 00 00 00 00 00 |..)....).
000000e0  00 10 00 00 00 00 00 00  01 00 00 00 05 00 00 00 |.....
000000f0  00 30 00 00 00 00 00 00  00 30 00 00 00 00 00 00 |..0.....0..
00000100  00 30 00 00 00 00 00 00  61 e0 00 00 00 00 00 00 |..0.....a.....
00000110  61 e0 00 00 00 00 00 00  00 10 00 00 00 00 00 00 |a.....
00000120  01 00 00 00 04 00 00 00  00 20 01 00 00 00 00 00 |.....
00000130  00 20 01 00 00 00 00 00  00 20 01 00 00 00 00 00 |. .....
```

Рисунок 6.1 — Фрагмент результата использования утилиты hexdump

```
student@student-VirtualBox:~$ strings -n3 /bin/ uname
ELF
/lib64/ld-linux-x86-64.so.2
GNU
GNU
GNU
V`I
MB#F-
,cr
libc.so.6
fflush
__printf_chk
setlocale
mbrtowc
fopen
strncmp
optind
strchr
dcgettext
error
__stack_chk_fail
fgets_unlocked
iswprint
realloc
abort
__exit
program_invocation_name
__ctype_get_mb_cur_max
calloc
strlen
memset
strstr
__errno_location
```

Рисунок 6.2 — Результат использования утилиты strings

2. Подсчитаем общее количество файлов (каталогов) в каталоге /var с помощью утилиты find с ключами wc -l, как показано на рисунке 6.3.

```
student@student-VirtualBox:~$ find . /var | wc -l
find: '/var/cache/apparmor/26b63962.0': Отказано в доступе
find: '/var/cache/ldconfig': Отказано в доступе
find: '/var/cache/system-tools-backends/backup': Отказано в доступе
find: '/var/cache/apt/archives/partial': Отказано в доступе
find: '/var/cache/private': Отказано в доступе
find: '/var/cache/cups': Отказано в доступе
find: '/var/lib/NetworkManager': Отказано в доступе
find: '/var/lib/AccountsService/users': Отказано в доступе
find: '/var/lib/fwupd/gnupg': Отказано в доступе
find: '/var/lib/snapd/void': Отказано в доступе
find: '/var/lib/snapd/cookie': Отказано в доступе
find: '/var/lib/colord/.cache': Отказано в доступе
find: '/var/lib/gdm3/.cache/libgweather': Отказано в доступе
find: '/var/lib/gdm3/.local/share/keyrings': Отказано в доступе
find: '/var/lib/gdm3/.local/share/gnome-shell': Отказано в доступе
find: '/var/lib/gdm3/.local/share/gvfs-metadata': Отказано в доступе
```

Рисунок 6.3 – Результат подсчёта общего количества файлов (каталогов) в каталоге /var

3. Найдём общее количество процессов, выполняющихся в системе в данный момент с помощью команды ps, выводящей отчёт о работающих процессах, где ключ a показывает процессы для всех пользователей, ключ u показывает/владельца процесса, ключ i показывает процессы, не подключенные к терминалу, как показано рисунке 6.4.

```
student@student-VirtualBox:~$ ps aux | grep bash | wc -l
2
student@student-VirtualBox:~$ ps aux | grep bash
student      2189  0.0  0.2  19240  5080 pts/0    Ss   13:56   0:00 bash
student      3330  0.0  0.0  17688   724 pts/0    S+   14:03   0:00 grep --color=auto bash
student@student-VirtualBox:~$ █
```

Рисунок 6.4 — Результат подсчёта общего количества процессов, выполняющихся в системе в данный момент

4. Выведем список выполняющихся процессов, в именах которых присутствует слово manager и отсутствует слово grep, как показано рисунке 6.5.

```
student@student-VirtualBox:~$ ps aux | grep bash | grep manager | grep -v grep
```

Рисунок 6.5 — Вывод списка выполняющихся процессов, в именах которых присутствует слово manager и отсутствует слово grep

5. Создадим текстовый файл, содержащий набор строк вида:

```
123  
178  
176  
755  
713  
873
```

Создание и содержание текстового файла с помощью утилиты printf, как показано рисунке 6.6, 6.7.

```
student@student-VirtualBox:~$ printf "123  
> 178  
> 176  
> 755  
> 713  
> 873  
> " > 5.txt  
student@student-VirtualBox:~$
```

Рисунок 6.6 — Создание текстового файла



Рисунок 6.7 — Содержание текстового файла

С помощью утилиты grep найдём строки, в которых есть цифра 7, после которой находится одна из цифр — 1, 3 или 5, как показано рисунке 6.8.

```
student@student-VirtualBox:~$ cat /home/student/5.txt | grep [7][135]
755
713
873
student@student-VirtualBox:~$
```

Рисунок 6.8 — Вывод сток с помощью утилиты grep, в которых есть цифра 7, после которой находится одна из цифр — 1, 3 или 5

6. Создадим текстовый файл, содержащий набор строк вида:

```
Starfish
Starless
Samscripter
Stellar
Microsrar
Ascender
sacrificscalar
```

Создание и содержание текстового файла с помощью утилиты описанной выше в пункте 5, как показано рисунке 6.9 – 6.10.

```
student@student-VirtualBox:~$ printf "starfish
> starless
> samscripter
> stellar
> microstar
> ascender
> sacrifice
> scalar
> " > 6.txt
student@student-VirtualBox:~$
```

Рисунок 6.9 — Создание текстового файла



```
1 starfish
2 starless
3 samscripter
4 stellar
5 microstar
6 ascender
7 sacrifice
8 scalar
```

Рисунок 6.10 — Содержание текстового файла

С помощью команды sort отсортируем строки заданного файла при помощи утилиты grep, с её помощью найдём строки, начинающиеся на букву s и заканчивающиеся на букву r, как показано рисунке 6.11.

```
student@student-VirtualBox:~$ sort /home/student/6.txt | grep '\b[s]\w*[r]\b'
samscriper
scalar
stellar
student@student-VirtualBox:~$
```

Рисунок 6.11 — Нахождение с помощью утилиты grep строк, начинающиеся на букву s и заканчивающихся на букву r

7. Создадим текстовый файл, содержащий простейшие адреса электронной почты вида username@email.com, описанным в пункте 5 методом, как показано рисунках 6.12, 6.13.

```
student@student-VirtualBox:~$ printf 'my@email.cim
> another.my@email.com
> not_my@email.com
> what_is_this@email.com
> where_are_you@email.com
> ' > 7.txt
student@student-VirtualBox:~$
```

Рисунок 6.12 — Создание текстового файла



Рисунок 6.13 — Содержание текстового файла

С помощью утилиты grep найдём строки, содержащие правильные простейшие адреса. Проверим возможность использования более сложного регулярного выражения для распознавания адресов, содержащих другие допустимые символы рисунок 6.14.

```
student@student-VirtualBox:~$ grep -E "(\w+\.)*\w+@\(\w+\.\)+[A-Za-z]+" 7.txt
my@email.cim
another.my@email.com
not_my@email.com
what_is_this@email.com
where_are_you@email.com
student@student-VirtualBox:~$
```

Рисунок 6.14 — Нахождение правильных простейших адресов

8 Для замены одних символов другими предназначена утилита tr.
Продемонстрируем её работу на произвольном примере, как показано
на рисунке 6.15.

```
student@student-VirtualBox:~$ printf '01234
' > 10.txt
student@student-VirtualBox:~$ cat /home/student/10.txt | tr 01234 56789 | head -4
56789
student@student-VirtualBox:~$
```

Рисунок 6.15 — Утилита tr

9. Создадим текстовый файл, содержащий допустимые и недопустимые IP-адреса, например

```
127.0.0.1
255.255.255.255
12.34.56
123.256.0.0
1.23.099.255
0.79.378.111
```

Создание и содержание текстового файла, как показано на
рисунке 6.16.

```
student@student-VirtualBox:~$ printf '127.0.0.1
> 255.255.255.0
> 22.33.44
> 1.22.081.32
> 0.79.333.222
> ' > 8.txt
student@student-VirtualBox:~$
```

Рисунок 6.16 — Создание текстового файла

С помощью утилиты grep и ключей -E -o и руководства man найдём строки, содержащие допустимые четырехбайтовые IP-адреса, как показано на рисунке 6.17.

```
student@student-VirtualBox:~$ grep -E -o "(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)" 8.txt
127.0.0.1
255.255.255.0
1.22.081.32
student@student-VirtualBox:~$
```

Рисунок 6.17 — Нахождение правильных IP-адресов

10 Создадим текстовый файл, содержащий корректные и некорректные номера телефонов ведомственной АТС объемом 399 номеров, номера с 000 до 399 – корректные, 0, 400, 900 – некорректные, как показано на рисунке 6.18.

```
390
391
392
393
394
395
396
396
397
398
399
400
900
' > 9.txt
```

Рисунок 6.18 — Создание текстового файла

С помощью утилиты grep и руководства man найдём строки, содержащие допустимые номера телефонов, как показано на рисунке 6.19.

```
student@student-VirtualBox:~$ grep -E "[0-3][0-9][0-9]" 9.txt
397
398
399
student@student-VirtualBox:~$
```

Рисунок 6.19 — Нахождение правильных номеров телефонов

Контрольные вопросы

1. Вывод на экран содержимого нетекстового файла с помощью утилит hexdump и strings.

На терминал Linux может быть выведен нетекстовый файл, однако пользы в этом будет мало. Во-первых, раз уж файл содержит не текст, то не предполагается, что пользователь сможет что-либо понять из его содержимого. Во-вторых, если в нетекстовых данных, выводимых на терминал, случайно встретится управляющая последовательность, терминал ее выполнит.

Если содержимое нетекстового файла все-таки желательно просмотреть (то есть превратить в текст), можно воспользоваться утилитой hexdump с ключом -C, которая выдает содержимое файла в виде шестнадцатеричных ASCII-кодов, или с помощью утилиты strings, показывающей только те части файла, которые могут быть представлены в виде текста.

2. Стандартный ввод, вывод, стандартный вывод ошибок.

Каждый процесс Linux получает при старте три дескриптора, открытых для него системой. Первый из них (дескриптор 0) открыт на чтение, это стандартный ввод процесса. Именно со стандартным вводом работают все операции чтения, если в них не указан дескриптор файла. Второй (дескриптор 1) – открыт на запись, это стандартный вывод процесса. С ним работают все операции записи, если дескриптор файла не указан в них явно. Наконец, третий поток данных (дескриптор 2) предназначается для вывода диагностических сообщений, он называется стандартный вывод ошибок.

3. Конвейер и канал.

Нередко возникают ситуации, когда нужно обработать вывод одной программы какой-то другой программой. Для решения подобной задачи в bash предусмотрена возможность перенаправления вывода можно не только в

файл, но и непосредственно на стандартный ввод другой программе. В Linux такой способ передачи данных называется конвейер.

В bash для перенаправления стандартного вывода на стандартный ввод другой программе служит символ «|». Можно последовательно обработать данные несколькими разными программами, перенаправляя вывод на ввод следующей программе и организуя сколь угодно длинный конвейер для обработки данных. В результате получаются командные строки вида «cmd1 | cmd2 | ... | cmdN».

Организация конвейера устроена в shell по той же схеме, что и перенаправление в файл, но с использованием особого объекта системы - канала. Можно представить трубу, немедленно доставляющую данные от входа к выходу (английский термин – «rⁱp^e»). Каналом пользуются сразу два процесса: один пишет туда, другой читает. Связывая две команды конвейером, shell открывает канал (заводится два дескриптора – входной и выходной), подменяет стандартный вывод первого процесса на входной дескриптор канала, а стандартный ввод второго процесса - на выходной дескриптор канала. После чего остается запустить по команде в этих процессах, и стандартный вывод первой попадет на стандартный ввод второй.

Канал (rⁱp^e) – неделимая пара дескрипторов (входной и выходной), связанных друг с другом таким образом, что данные, записанные во входной дескриптор, будут немедленно доступны на чтение с выходного дескриптора.

4. Фильтры.

Если программа и вводит данные, и выводит, то ее можно рассматривать как трубу, в которую что-то входит и из которой что-то выходит. Обычно смысл работы таких программ заключается в том, чтобы определенным образом обработать поступившие данные. В Linux такие программы называют фильтрами: данные проходят через них, причем что-то «застревает» в фильтре и не появляется на выходе, а что-то изменяется, что-то проходит сквозь фильтр неизменным. Фильтры в Linux обычно по умолчанию читают

данные со стандартного ввода, а выводят на стандартный вывод. Простейший фильтр - программа cat.

5. Структурные единицы текста. Подсчет количества единиц текста.

Работая с текстом в Linux, нужно принимать во внимание, что текстовые данные, передаваемые в системе, структурированы. Большинство утилит обрабатывает не непрерывный поток текста, а последовательность единиц. В текстовых данных в Linux выделяются следующие структурные единицы:

- Строки
- Поля
- Символы

6. Элементарные регулярные выражения.

Регулярными выражениями называются особым образом составленные наборы символов, выделяющие из текста нужное сочетание слов или символов, которое соответствует признакам, отраженным в регулярном выражении. Иными словами, регулярное выражение – это фильтр для текста.

В Linux регулярные выражения используются командой grep, которая позволяет искать файлы с определенным содержанием либо выделять из файлов строки с необходимым содержимым (например, номера телефонов, даты и т. д.).

7. Знакозаменяющие метасимволы.

Не все символы можно использовать прямо по назначению. Посмотрите, например, на конструкцию, которая описывалась в предыдущем разделе. Допустим, требуется найти в каком-то файле строки, содержащие следующий набор символов: abc[def. Можно предположить, что регулярное выражение будет составлено по принципам, описанным выше, но это неверно. Открывающая квадратная скобка — это один из символов, который несет для программы, работающей с регулярными выражениями, особый смысл (который был рассмотрен ранее). Такие символы называются метасимволами.

В Linux к знакозаменяющим метасимволам относятся:

- . – Предполагает, что в конечном выражении на ее месте будет стоять любой символ.
- \w – Замещает любые символы, которые относятся к буквам, цифрам и знаку подчеркивания.
- \W – Замещает все символы, кроме букв, цифр и знака подчеркивания (то есть является обратным метасимволом \w).
- \d – Замещает все цифры.
- \D – Замещает все символы, кроме цифр.
- \b – Замещает символ перевода курсора на один влево (возврат курсора).
- \r – Замещает символ перевода курсора в начало строки.
- \n – Замещает символ переноса курсора на новую строку.
- \t – Замещает символ горизонтальной табуляции.
- \v – Замещает символ вертикальной табуляции.
- \f – Замещает символ перехода на новую страницу.
- \s – Равнозначен использованию пяти последних метасимволов, то есть вместо метасимвола \s можно написать [\r\n\t\v\f].
- \S – Является обратным метасимволу \s.

8. Метасимволы количества повторений в регулярных выражениях.

В Linux к метасимволам количества повторений в регулярных выражениях относятся:

- ? – Указывает обработчику регулярных выражений на то, что предыдущий символ, метасимвол или конструкция могут вообще не существовать в конечном тексте либо присутствовать, но иметь не более одного вхождения.
- * – Означает, что впереди стоящие символ, метасимвол либо конструкция могут как отсутствовать, так и быть в конечном выражении, причем количество вхождений неограниченно.

· + – Действие схоже с действием предыдущего символа с тем отличием, что впередистоящие метасимвол, символ или конструкция должны повторяться как минимум один раз.

· { min, max } – Указывает минимальное и максимальное количество вхождений.

9. Группировка выражений в регулярных выражениях.

Группировка в регулярных выражениях применяется для соединения нескольких его составляющих в одну единицу.

10. Использование зарезервированных символов в регулярных выражениях.

Некоторые символы имеют для программы, работающей с регулярными выражениями, особый смысл. Это, например, косая черта, точка, круглая, фигурная и квадратная скобки, звездочка и т. д. Однако не исключено, что в целевом выражении также могут быть эти символы, и их наличие нужно будет определить в регулярном. В данном случае эти символы нужно указать особым образом. Для этого перед нужным символом ставят косую черту, то есть, чтобы указать наличие в конечном тексте символа звездочки, в регулярном выражении на соответствующем месте следует написать *.

11. Подсчет количества элементов текстового файла.

Часто бывает необходимо подсчитать количество определенных элементов текстового файла. Для подсчета строк, слов и символов служит стандартная утилита – wc (от англ. «word count» – «подсчет слов»).

12. Назначение утилит head, tail, cut.

Иногда пользователя интересует только часть из тех данных, которые собирается выводить программа. Утилита head нужна, чтобы вывести только первые несколько строк файла. Не менее полезна утилита tail (англ. «хвост»), выводящая только последние строки файла. Если же пользователя интересует только определенная часть каждой строки файла - поможет утилита cut.

13. Назначение и использование утилиты grep.

Зачастую пользователю нужно найти только упоминания чего-то конкретного среди данных, выводимых утилитой. Обычно эта задача сводится к поиску строк, в которых встречается определенное слово или комбинация символов. Для этого подходит стандартная утилита grep, которая может искать строку в файлах, а может работать как фильтр: получив строки со стандартного ввода, она выведет на стандартный вывод только те строки, где встретилось искомое сочетание символов.

14. Выполнение транслитерации.

Удобство работы с потоком не в последнюю очередь состоит в том, что можно не только выборочно передавать результаты работы программ, но и автоматически заменять один текст другим прямо в потоке. Для замены одних символов другими предназначена утилита tr (сокращение от англ. «translate» - «преобразовывать, переводить»), работающая как фильтр.

15. Назначение потокового редактора sed.

Помимо простой замены отдельных символов, возможна замена последовательностей (слов). Специально для этого предназначен потоковый редактор sed (сокращение от англ. «stream editor»). Он работает как фильтр и выполняет редактирование поступающих строк: замену одних последовательностей символов другими, причем можно заменять и регулярные выражения