Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение

высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

им. Н.П. ОГАРЁВА»

(ФГБОУ ВО «МГУ им. Н.П. Огарёва»)

Факультет математики и информационных технологий

Кафедра систем автоматизированного проектирования

ИТОГОВЫЙ ОТЧЕТ

по дисциплине: Операционные системы

Автор отчёта о лабораторной работе  А. Е. Конышев

подпись, дата

Обозначение лабораторной работы ЛР–02069964–02.03.02–08–23

Направление подготовки 02.03.02 Фундаментальная информатика и информационные технологии

Руководитель работы

канд. техн. наук, доц.  А. В. Шамаев

подпись, дата

Саранск 2023

**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 С:. Если диск С: недоступен, использовать любой другой доступный диск.

7. Скопировать все имеющиеся в каталоге Windows исполняемые файлы в каталог WinEx на диске С:. Если диск С: 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).

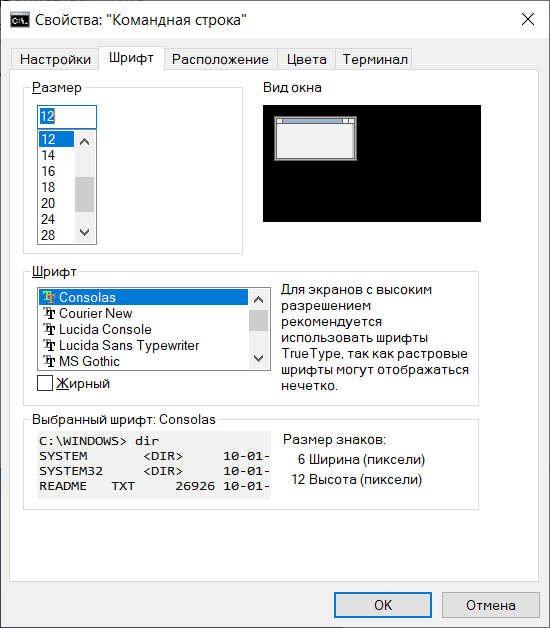


Рисунок 1.2 – Изменение размера окна интерпретатора

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

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

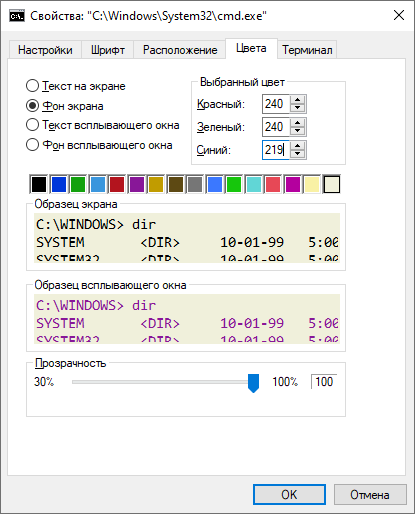


Рисунок 1.3 – Изменение цвета фона окна интерпретатора

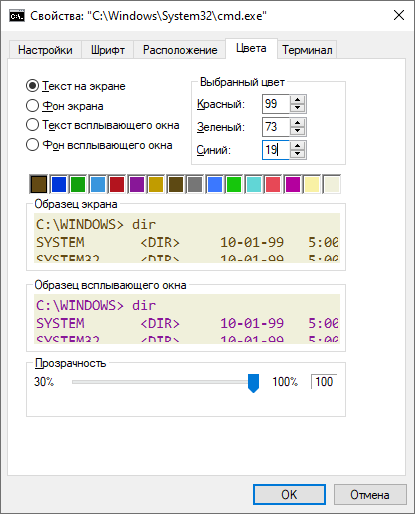


Рисунок 1.4 – Изменение цвета шрифта окна интерпретатора

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.

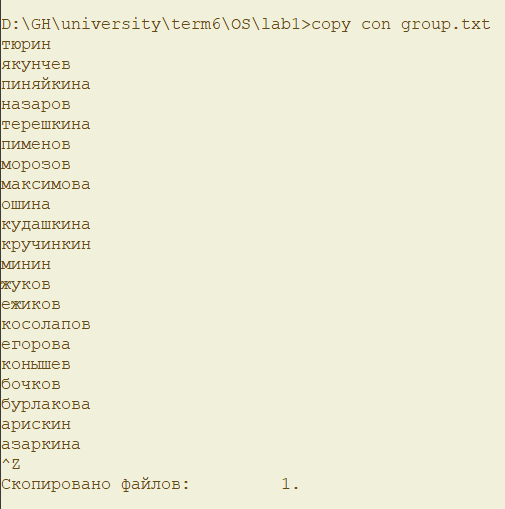


Рисунок 1.5 – Создание списка студентов группы

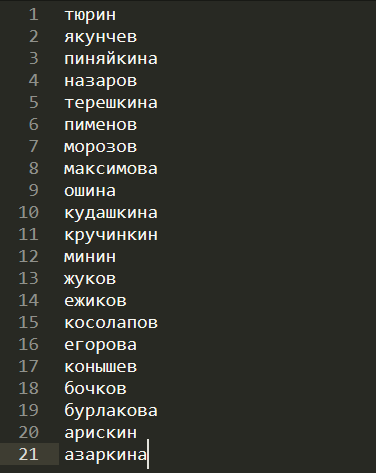


Рисунок 1.6 –Содержимое файла group.txt

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.

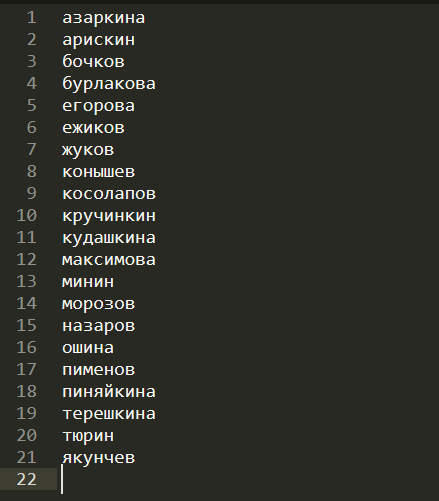


Рисунок 1.7 – Содержимое файла с результатом работы команды SORT

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.

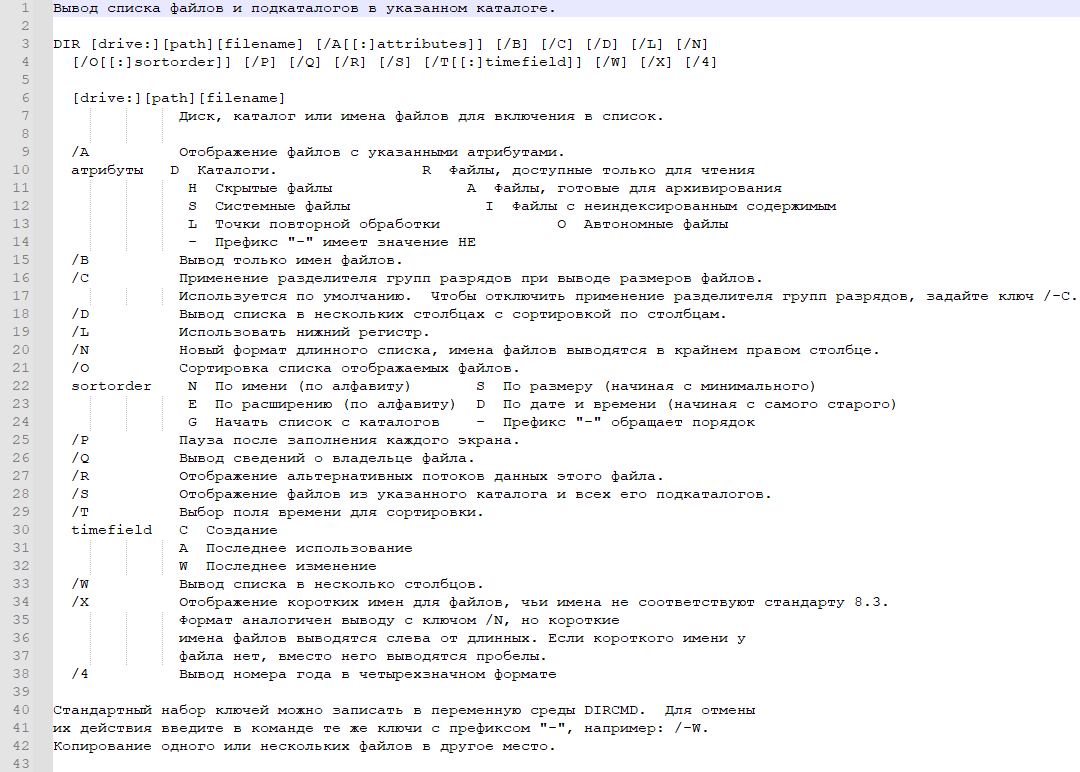


Рисунок 1.8 – Фрагмент файла info.txt, содержащий сведения о команде DIR

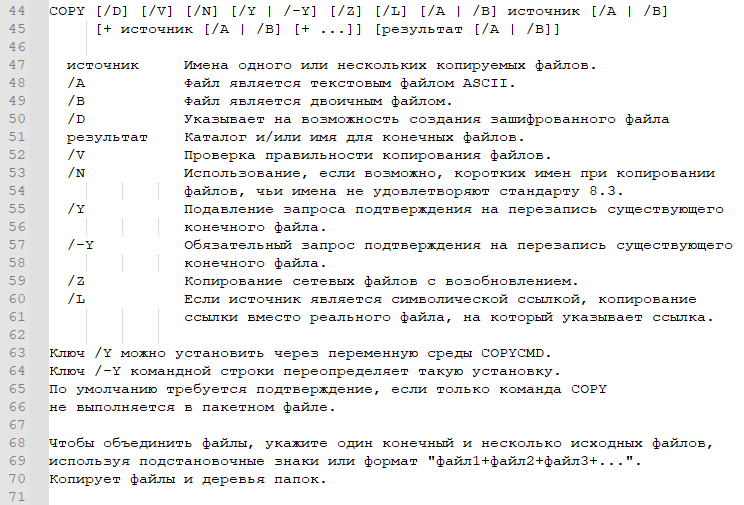


Рисунок 1.9 – Фрагмент файла info.txt, содержащий сведения о команде COPY



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.

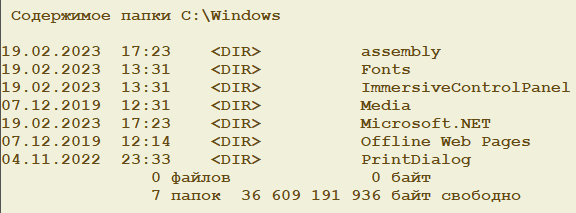


Рисунок 1.11 – Вывод каталогов папки Windows, доступных только для чтения, отсортированных по именам

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

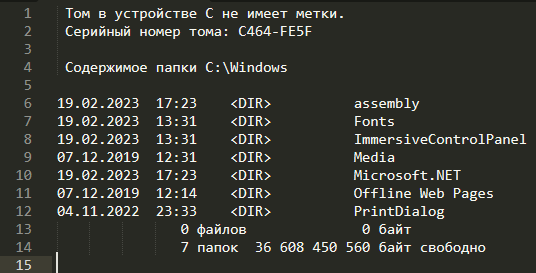


Рисунок 1.12 – Содержимое файла dir.txt

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.

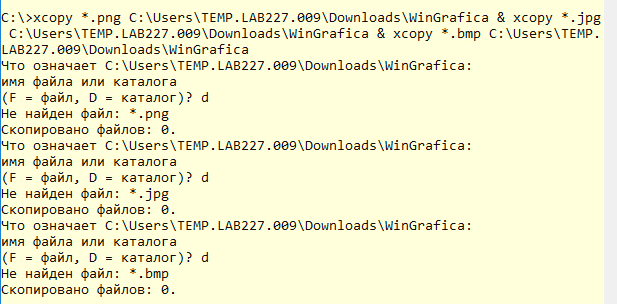


Рисунок 1.13 – Копирование графических файлов

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.

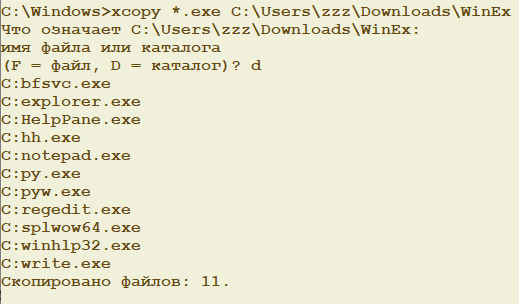


Рисунок 1.14 – Копирование исполняемых файлов

**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 помощь 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.

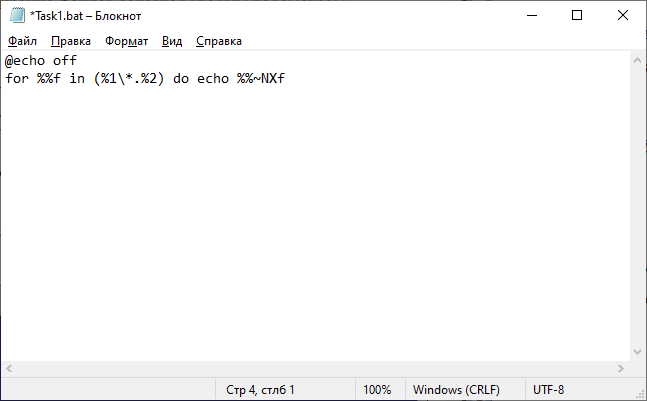


Рисунок 2.1 – Листинг первой задачи

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.

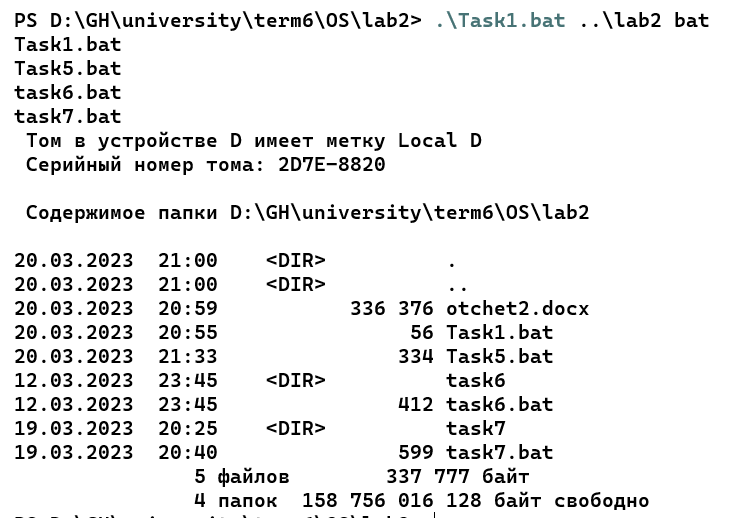


Рисунок 2.2 – Выполнение первой программы

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.

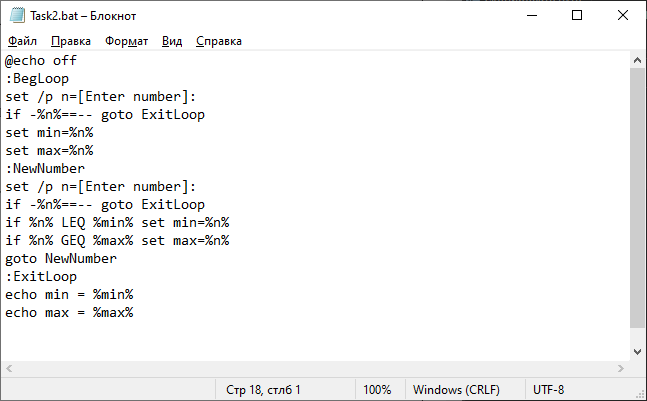


Рисунок 2.3 – Листинг второй задачи

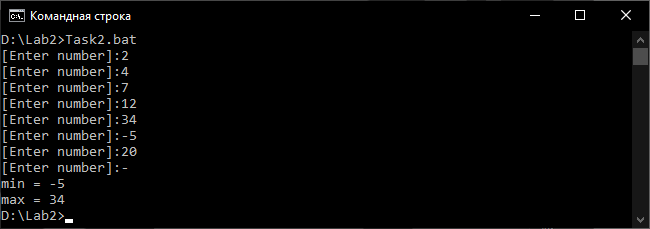


Рисунок 2.4 – Выполнение второй программы

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 это кода изображен на рисунке 2.6.

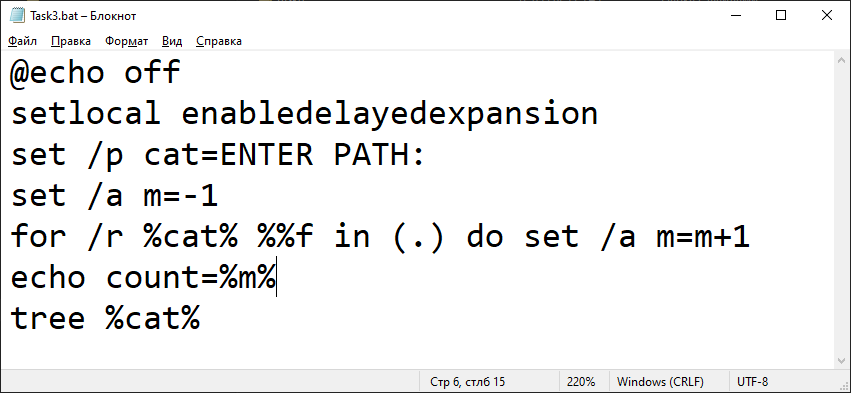


Рисунок 2.5 – Код третьей задачи



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.

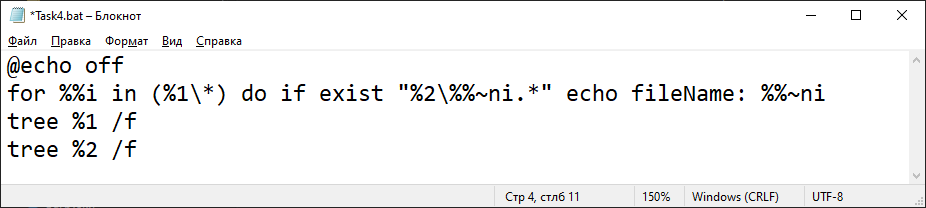


Рисунок 2.7 – Листинг четвёртой задачи

Результат выполнения это кода изображен на рисунке 2.8.



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:

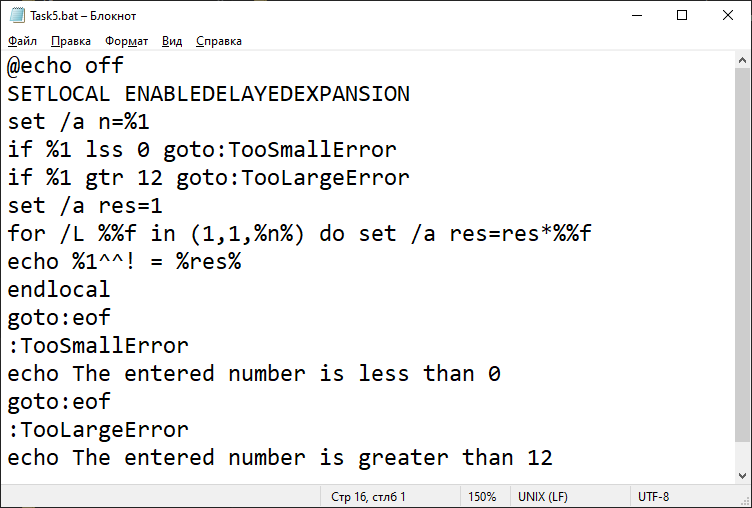


Рисунок 2.9 – Листинг пятой задачи

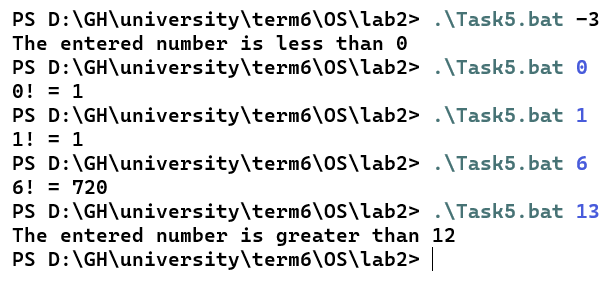


Рисунок 2.10 – Выполнение пятой программы

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.



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.

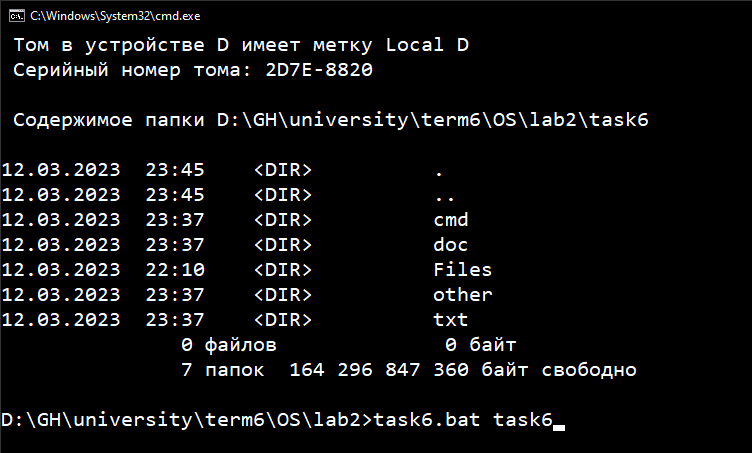


Рисунок 2.12 – Результат запуска КФ для 6 задания

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.

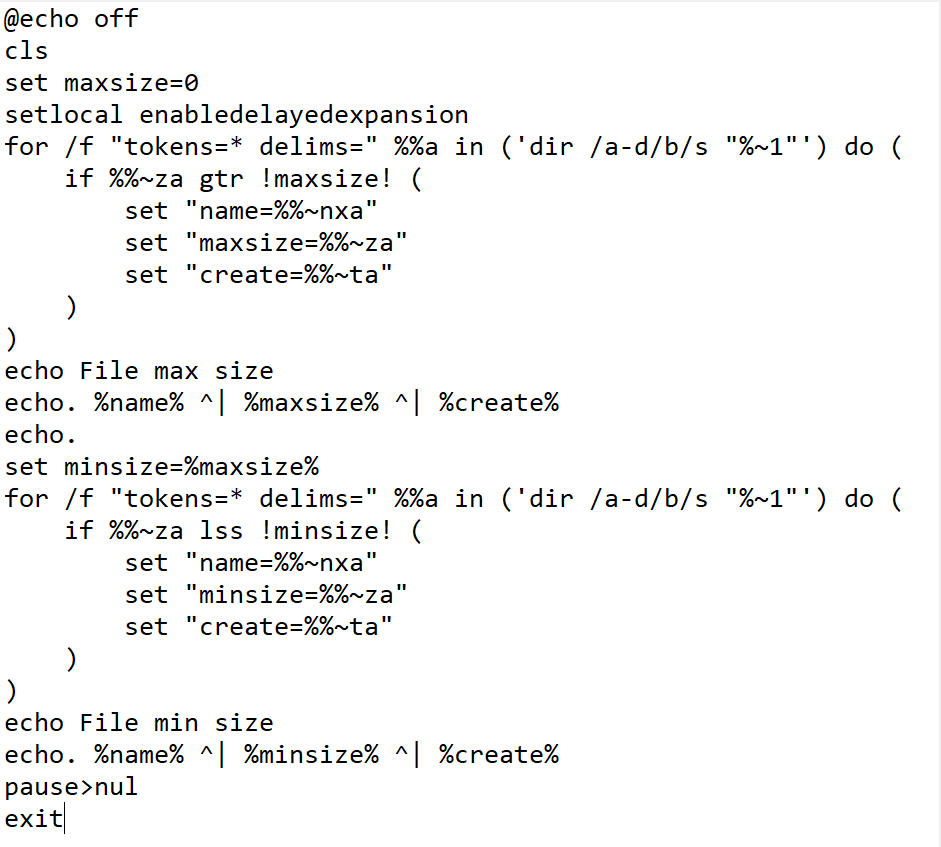


Рисунок 2.13 – Листинг КФ для выполнения задания 7.

Результат запуска КФ с параметром task7 изображен на рисунке 2.14.

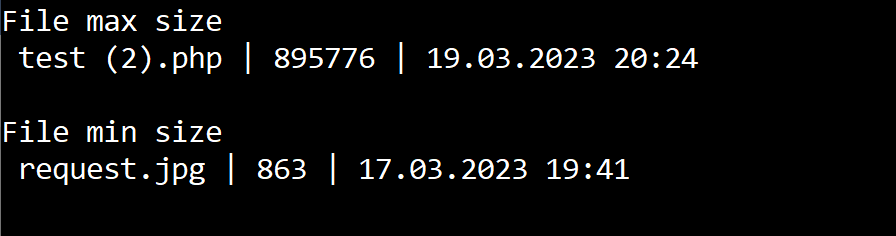


Рисунок 2.14 – Результат запуска КФ для 7 задания.

**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 А=Раз

SET В=Два

SET C=%A%%B%

Можно выделить подстроку с помощью следующей конструкции: %имя\_переменной:~п1,п2%, 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 /А. 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] [/Т: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.

/Т:файл-Читает список файлов из заданного файла

/С: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

\х – Служебный символ: символьное обозначение служебного символа x.

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

xyz\> – 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 онка 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 командлетом 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 командлетов 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

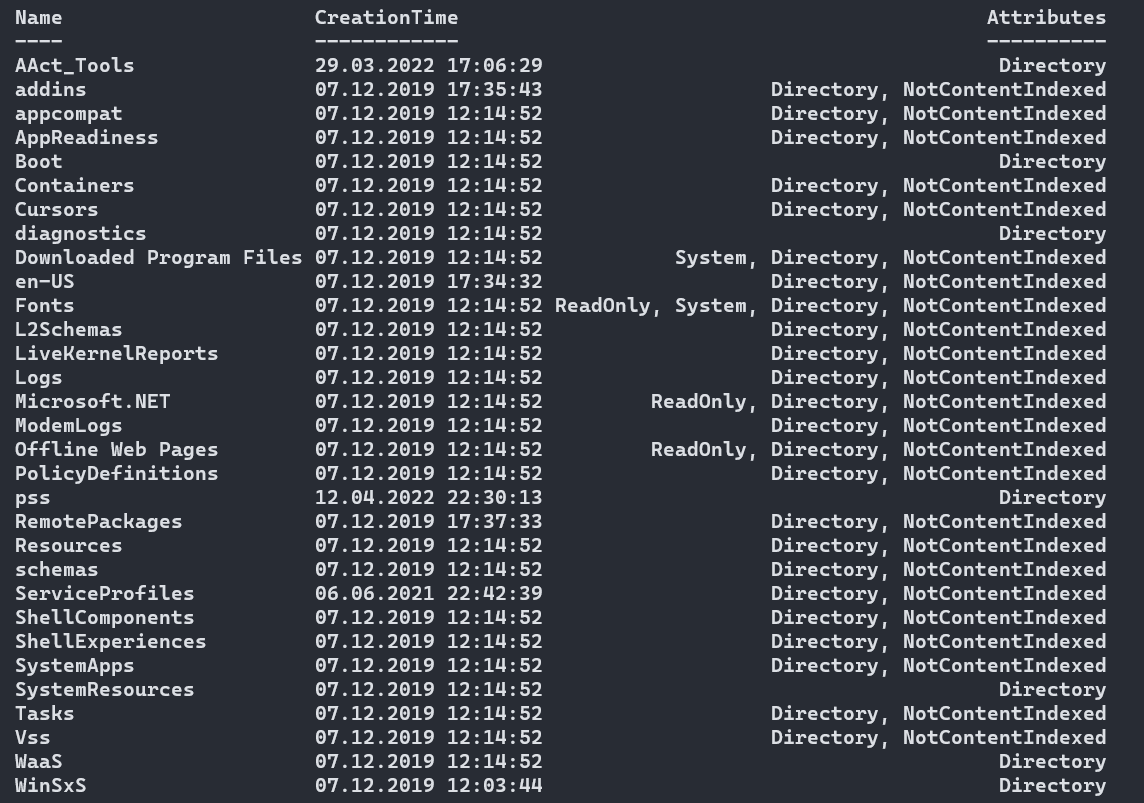


Рисунок 3.1 – Результат выполнения скрипта для 4 задания



Рисунок 3.2 – Содержимое файла task4\_output.txt

5. Вывести в текстовый файл список свойств процесса, возвращаемый командлетом 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

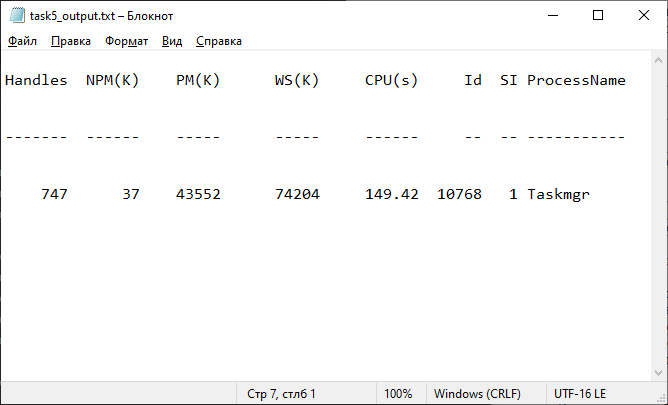


Рисунок 3.3 – Содержимое файла task5\_output.txt

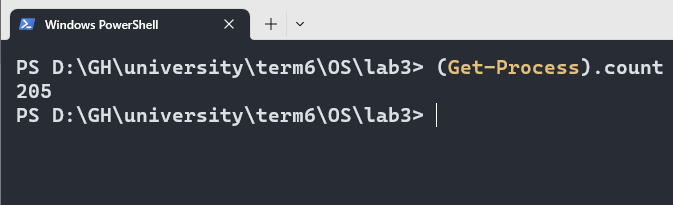


Рисунок 3.4 – Результат выполнения скрипта для второй части задания 5

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.

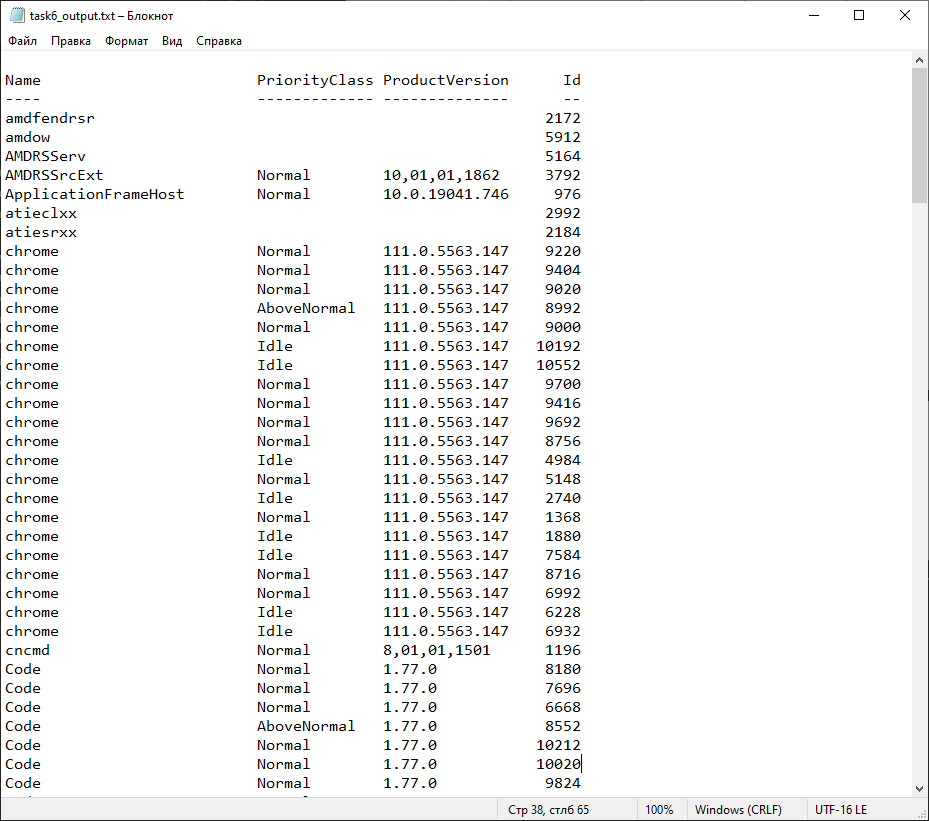


Рисунок 3.5 – Содержимое файла task06\_output.txt

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



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.

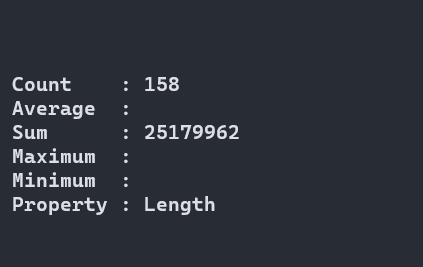


Рисунок 3.8 – Результат работы скрипта для выполнения задания 8

9. Вывести на экран сведения о ЦП компьютера.

Следущий скрипт позволит выполнить задание:

Get-WmiObject Win32\_Processor

Результат его работы представлен на рисунке 3.9

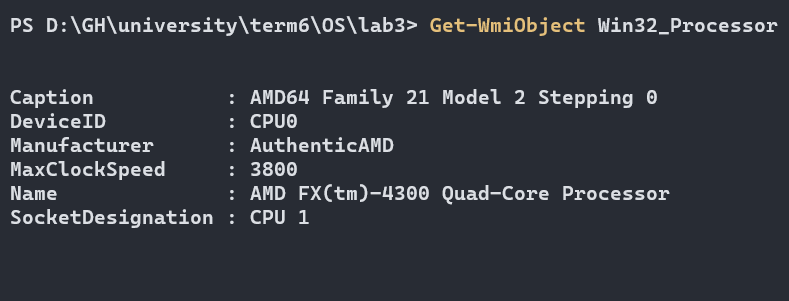


Рисунок 3.9 –Информация о ЦП компьютера

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) > $null

}

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.

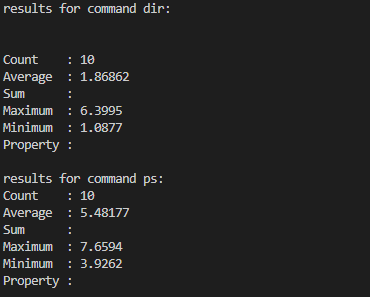


Рисунок 3.10 – Время работы командлетов dir и ps

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.

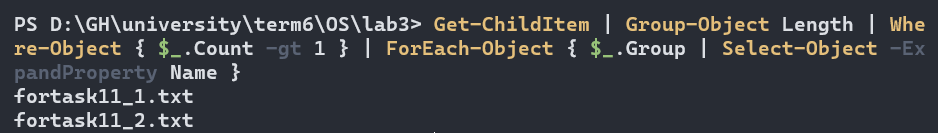


Рисунок 3.11 – Список файлов одинакового размера текущего каталога

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.

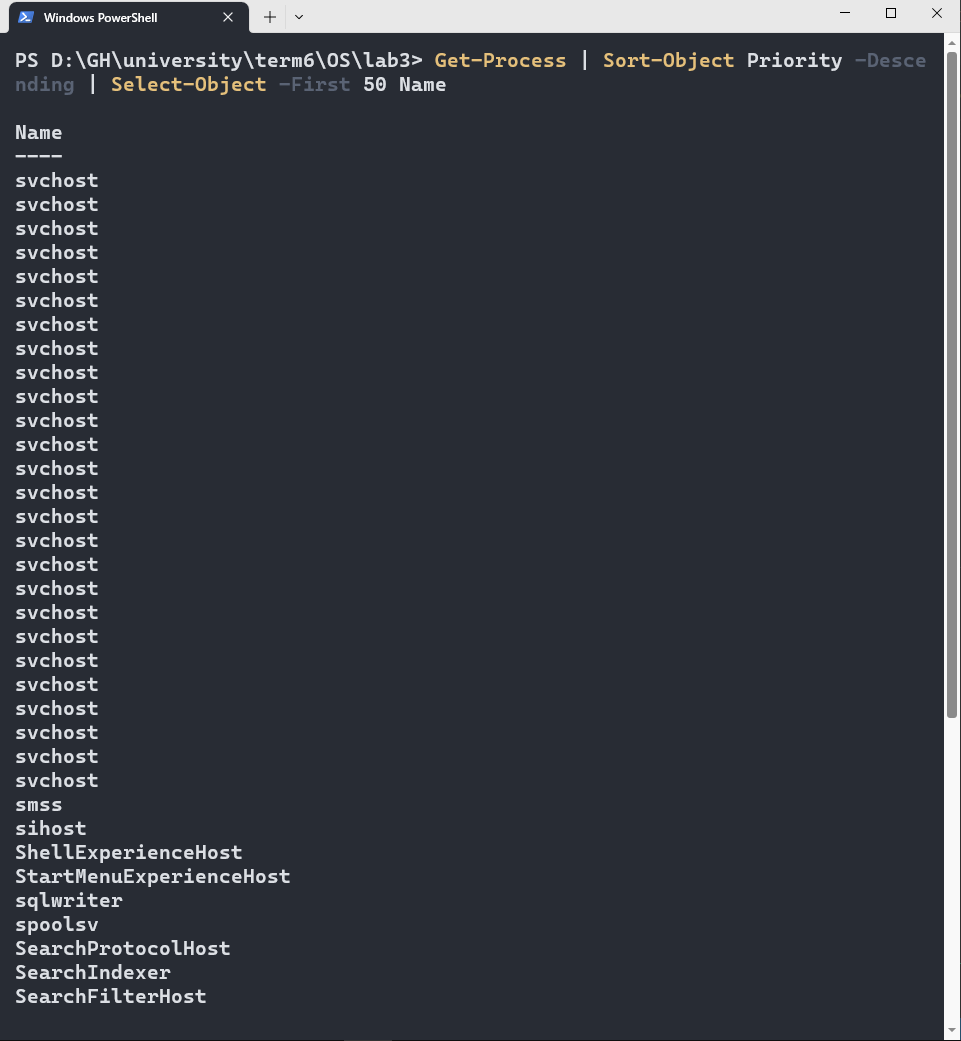


Рисунок 3.12 – Список процессов с наивысшим приоритетом

.

**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 61 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 пседонимов, 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:

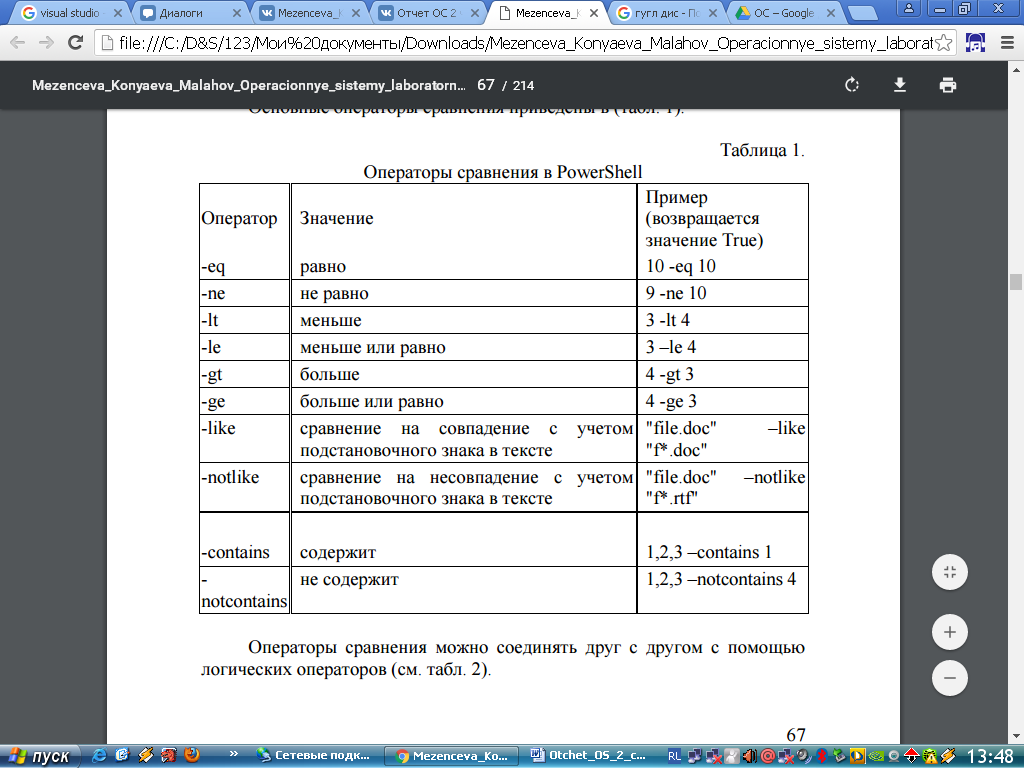
Get-Process | Where-Object {-not $\_.Responding}

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

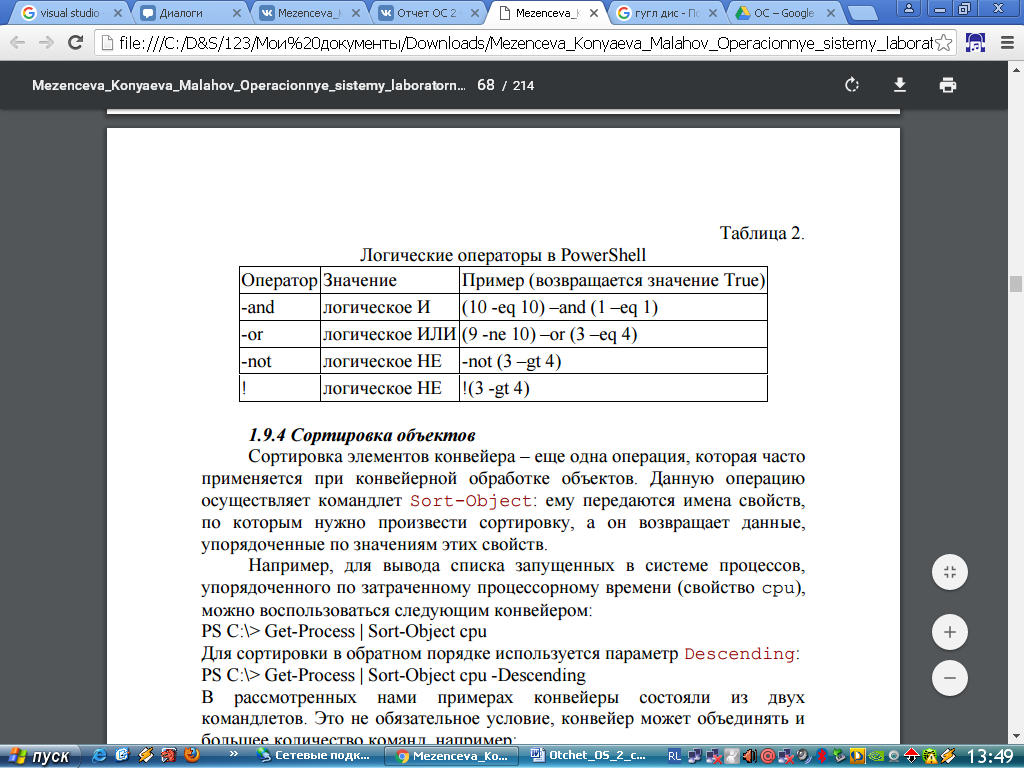
Get-Process | Where-Object {$\_.Id -gt 1000} 67

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

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



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

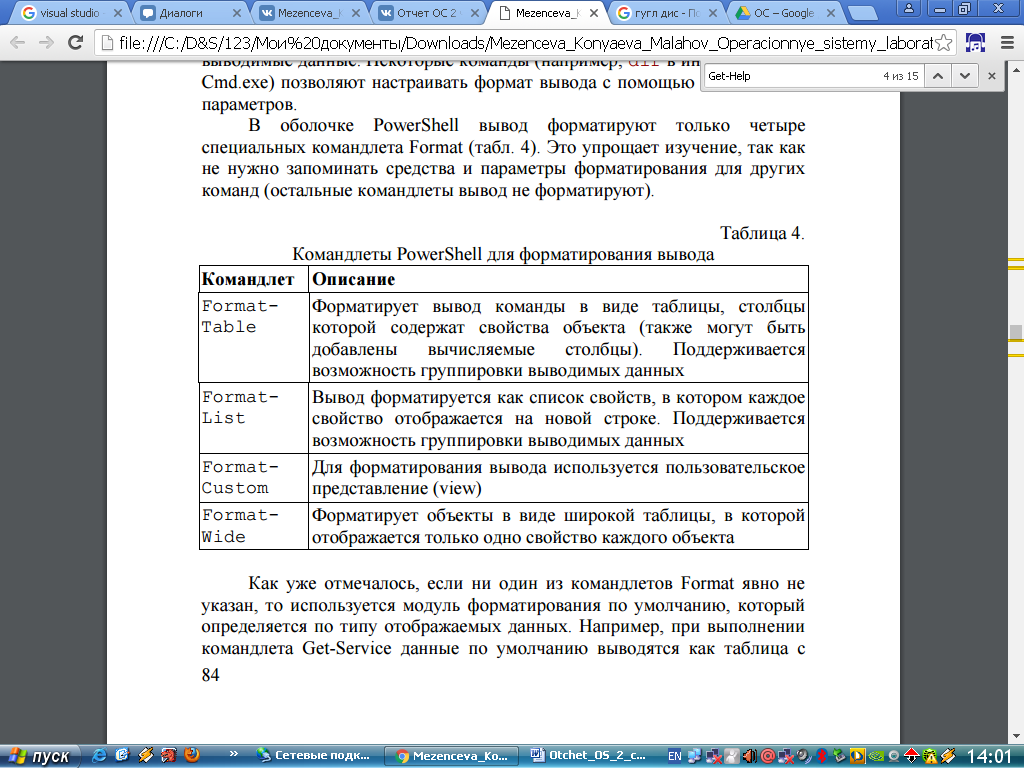


6. Какую информацию выводит команда Get-Help \* ?

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).



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.

А) Инструкция If …ElseIf … Else

В общем случае синтаксис инструкции If имеет вид

If (условие1) {блок\_кода1}

[ElseIf (условие2)] {блок\_кода2}]

[Else {блок\_кода3}].

При выполнении инструкции If проверяется истинность условного выражения условие1. Если условие1 имеет значение $True, то выполняется блок\_кода1, после чего выполнение инструкции if завершается. Если условие1 имеет значение $False, проверяется истинность условного выражения условие2. Если условие2 имеет значение $True, то выполняется блок\_кода2 и выполнение инструкции if завершается. Если и условие1, и условие2 имеют значение $False, то выполняется блок\_кода3 и выполнение инструкции if завершается.

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 эначение 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.

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

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

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 себеиспользоваться 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 командлет $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).

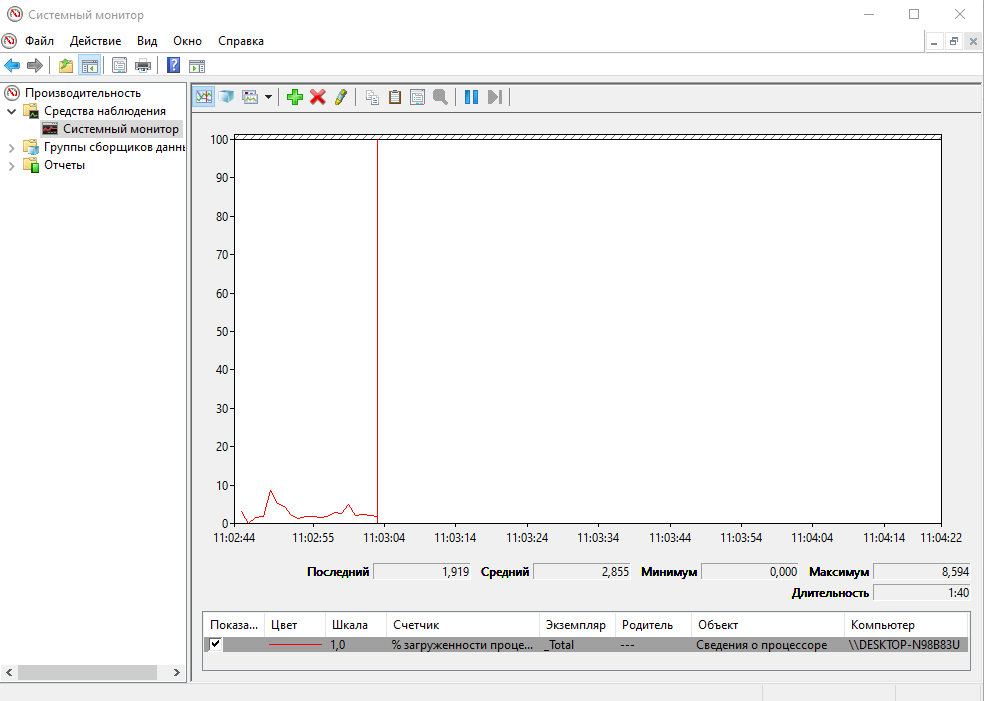
.

Рисунок 4.2 – Программа Performance Monitor (Системный монитор)

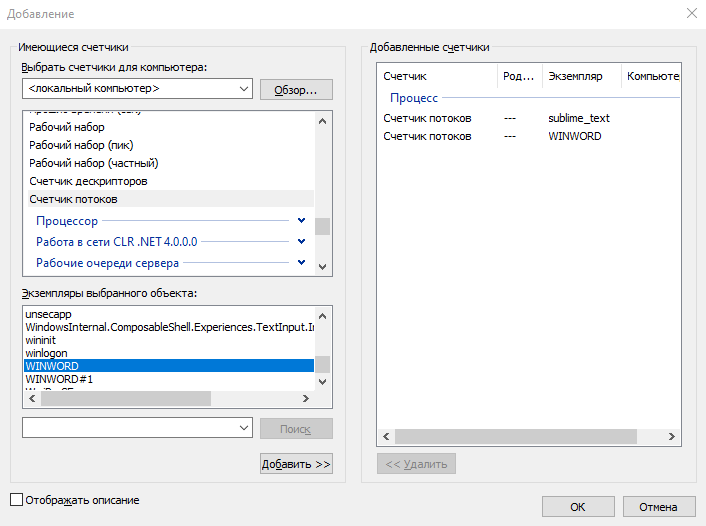


Рисунок 4.3 – Добавление счетчиков потоков приложений Sublime, WordPad

После добавления счетчиков в область для построения графиков в ней появятся графики, отображающие изменения количества потоков для выбранных процессов (рисунок 4.4). Наберем в каждом из текстовых редакторов по одному слову и сохраним соответствующие файлы, после чего закроем программу. Согласно полученному графику (рисунок 5.4) 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.

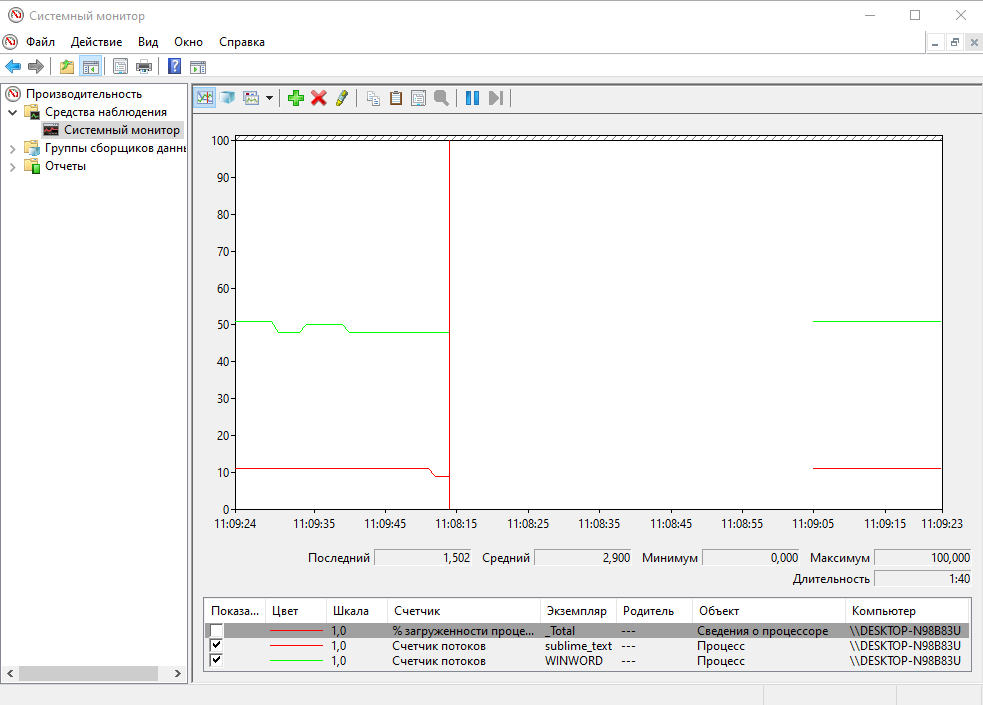


Рисунок 4.4 – Графики количества потоков приложений Sublime, Word

1. 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 4.5). Выполним несколько произвольных вычислений в приложении «Калькулятор» и выведем соответствующие графики изменения приоритетов потоков (рисунок 4.6).

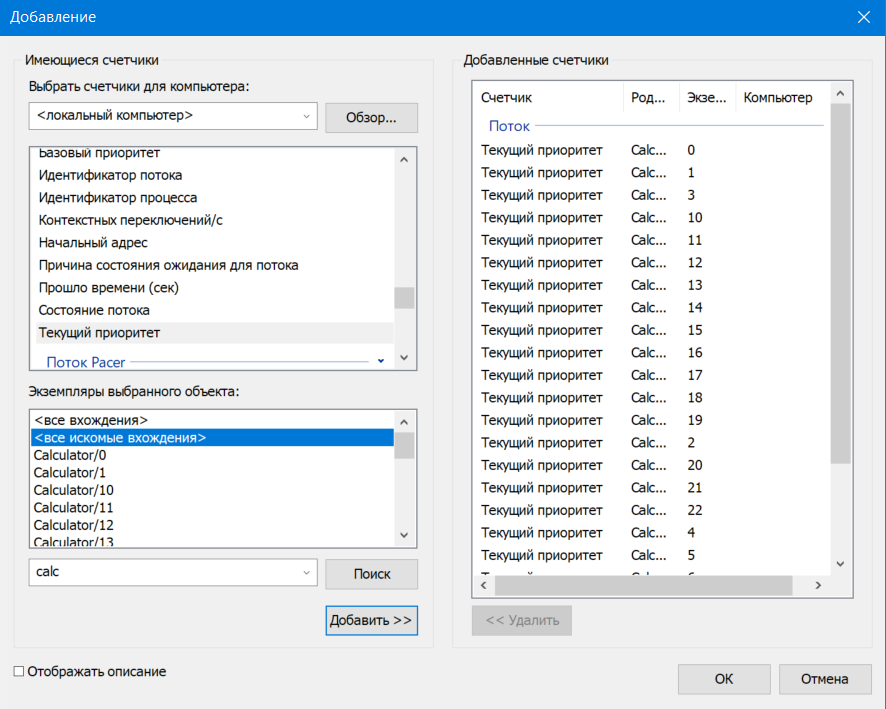


Рисунок 4.5 – Добавление счетчиков «Текущий приоритет» при вычислении арифметических выражений в программе «Калькулятор»

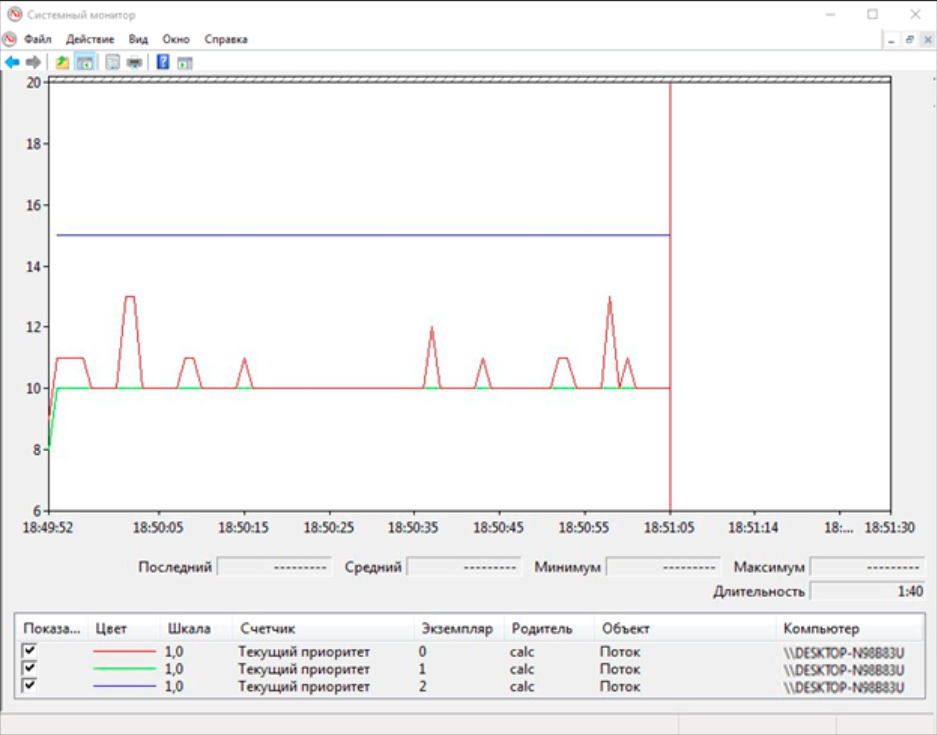


Рисунок 4.6 – График текущего приоритета процессов при вычислении арифметических выражений в программе «Калькулятор»

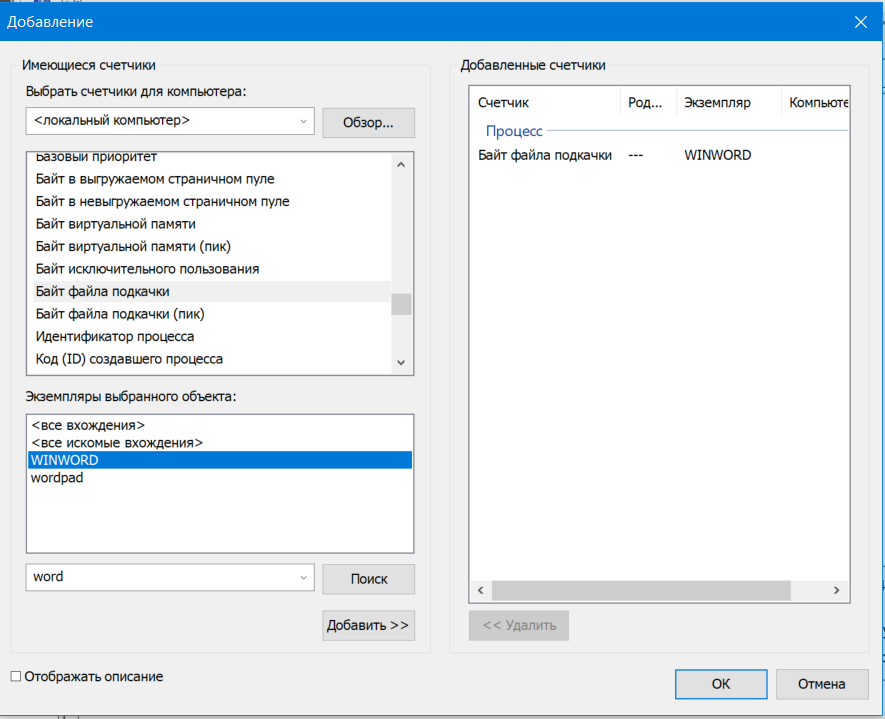


Рисунок 4.7 – Добавление счетчика «Байт файла подкачки»

1. 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 4.7).

Последовательно откроем три файла размером 430 Кб, 837 Кб и 1102 Кб соответственно, а затем проанализируем полученный график (рисунок 4.8). 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».

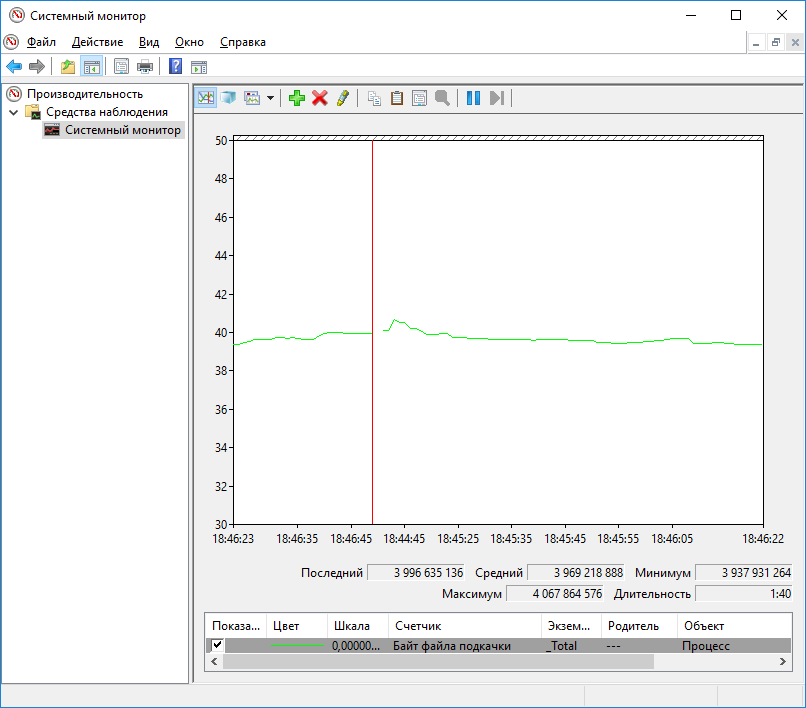


Рисунок 4.8 – График изменения размера файла подкачки при последовательном открытии 3 файлов увеличивающегося размера в приложении «Word»

1. 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).

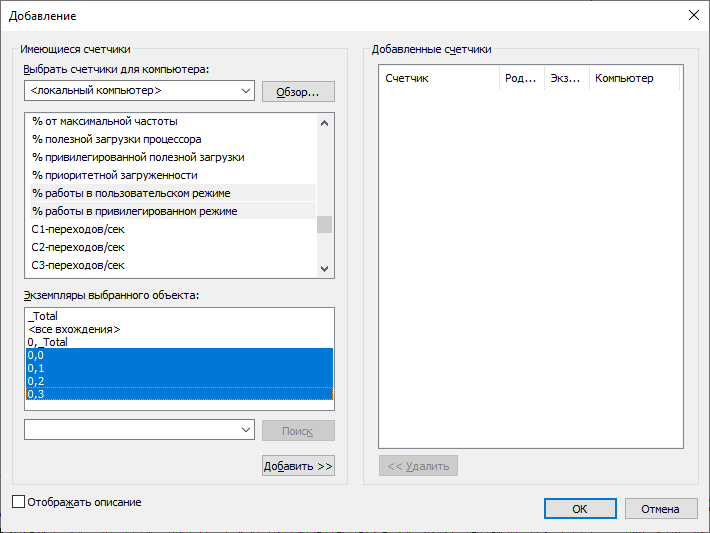


Рисунок 4.9 – выбор данных для выполнения 4 задачи.

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.

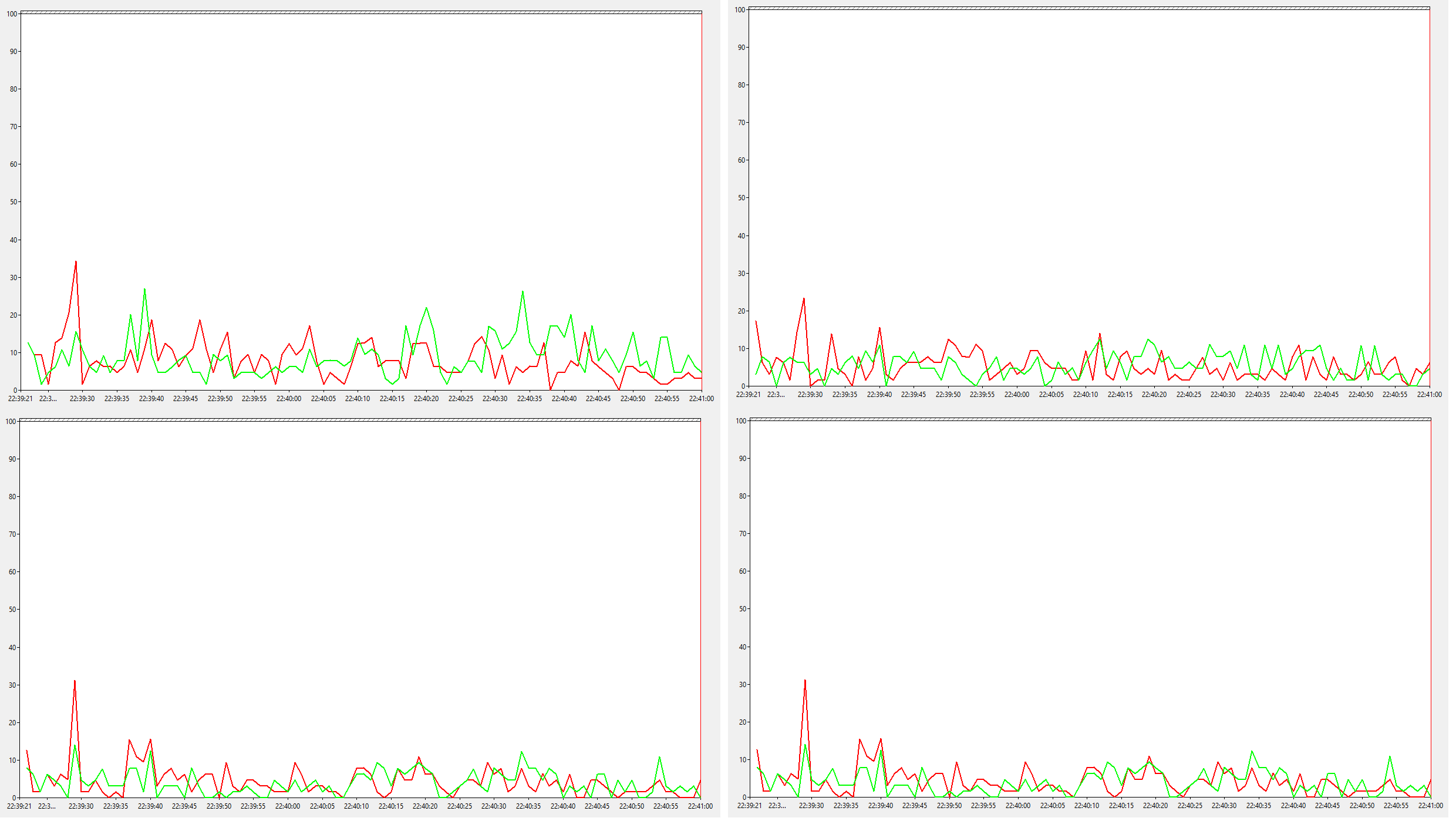


Рисунок 4.10 – результаты исследования режима работы ядер процессора

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.

1. 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.

1. 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

1. 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: а - 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.

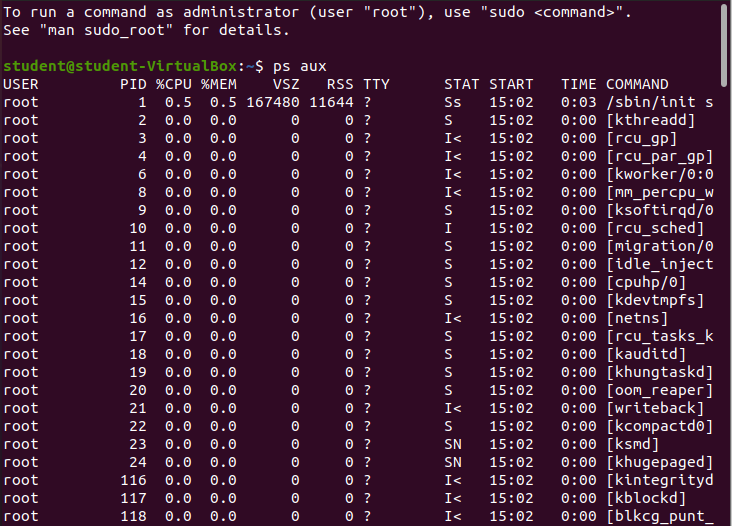


Рисунок 5.1 – Фрагмент вывода списка процессов системы

1. 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.



Рисунок 5.2 – Фрагмент вывода списка процессов системы в виде дерева

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

Для того чтобы вывести список процессов, отсортированных по потреблению наибольшего количество 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.

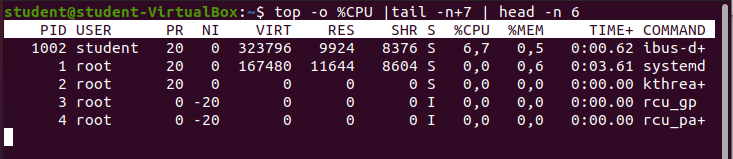


Рисунок 5.3 – Список 5 процессов, потребляющих наибольшее количество процессорного времени

1. 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 -opid,cmd,nlwp --sort=-nlwp | head –n 3

Результат выполнения данной команды, представлен на рисунке 5.4.

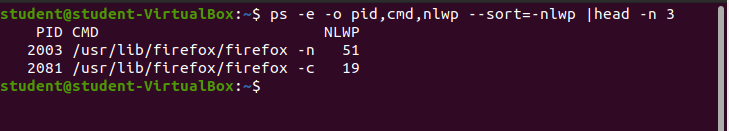


Рисунок 5.4 – Два процесса, имеющих более ДВУХ потоков

1. 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).

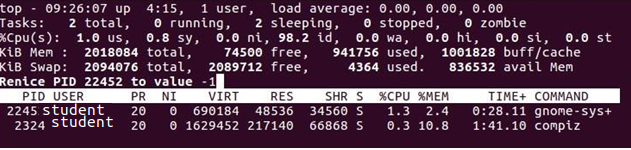


Рисунок 5.5– Информация о процессах



Рисунок 5.6 – Изменение приоритета процесса с PID 22452

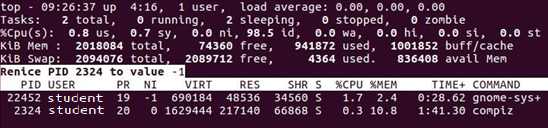


Рисунок 5.7– Изменение приоритета процесса с PID 2324

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

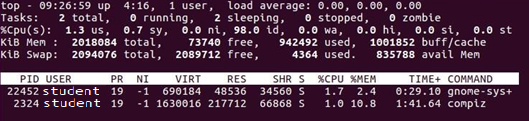


Рисунок 5.8 – Информация о процессах

1. 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.



Рисунок 5.9– Фрагмент списка открытых файлов пользователя

1. 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.

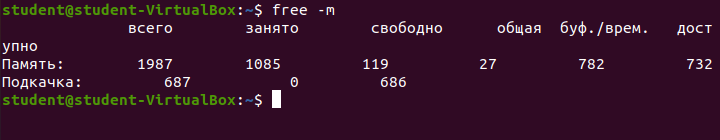


Рисунок 5.10– Текущее состояние системной памяти

1. 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.

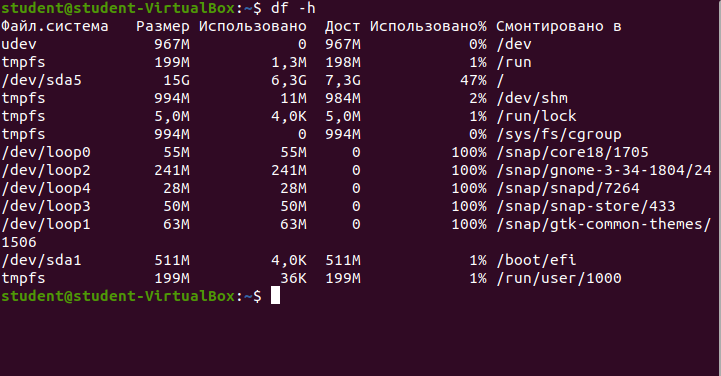


Рисунок 5.11– Справка об использовании дискового пространства

1. 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.



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.



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.

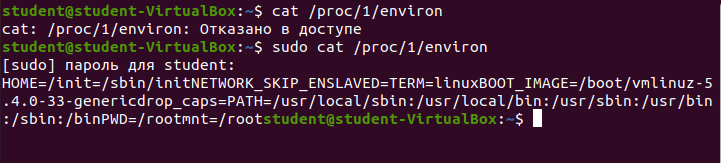


Рисунок 5.14 – Все переменные окружения, определенные для процесса с PID 1

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.



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.

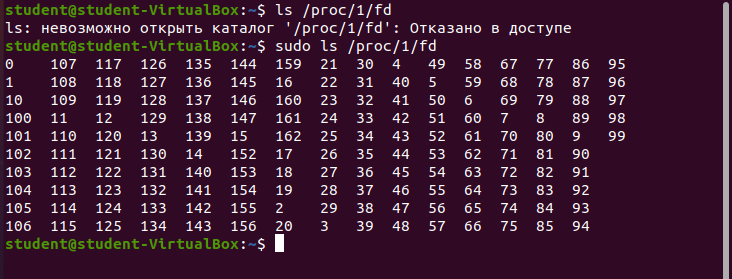


Рисунок 5.16– Список файловых дескрипторов, открытых в данный момент процессом с PID 1

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.

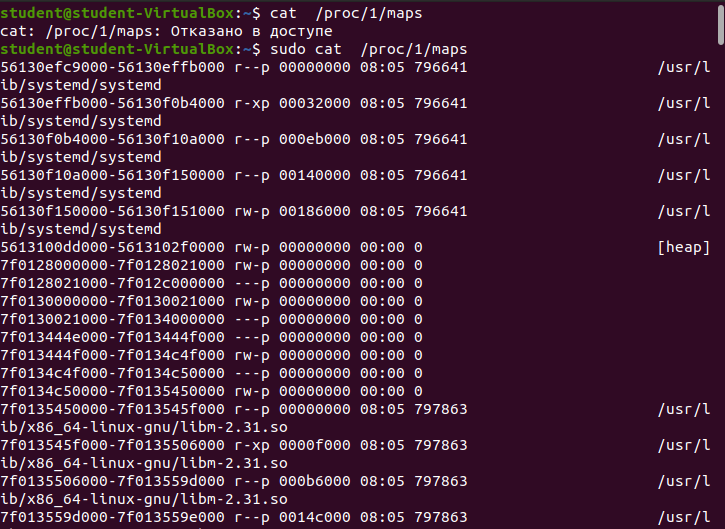


Рисунок 5.17– Фрагмент адресного пространства процесса с PID 1

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.

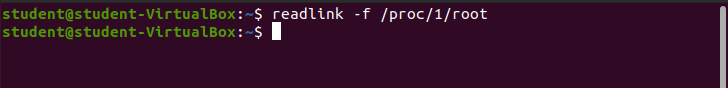


Рисунок 5.18– Корневой каталог, используемый процессом с PID 1

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.

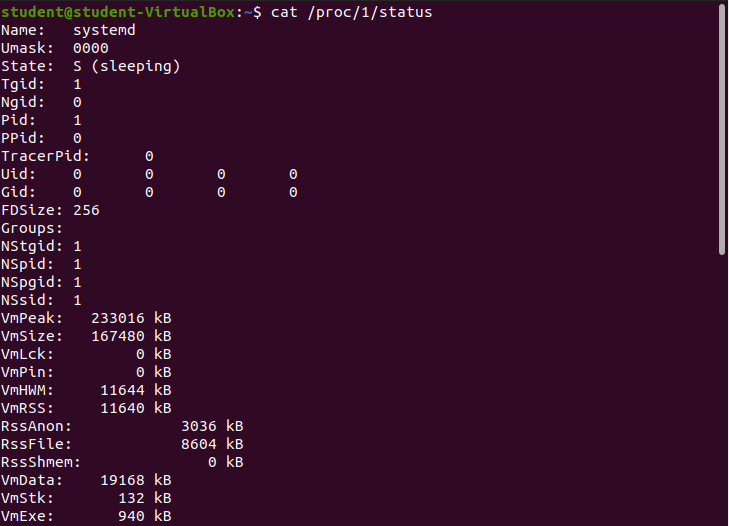


Рисунок 5.19–Фрагмент информации о процессе с PID 1

1. 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.

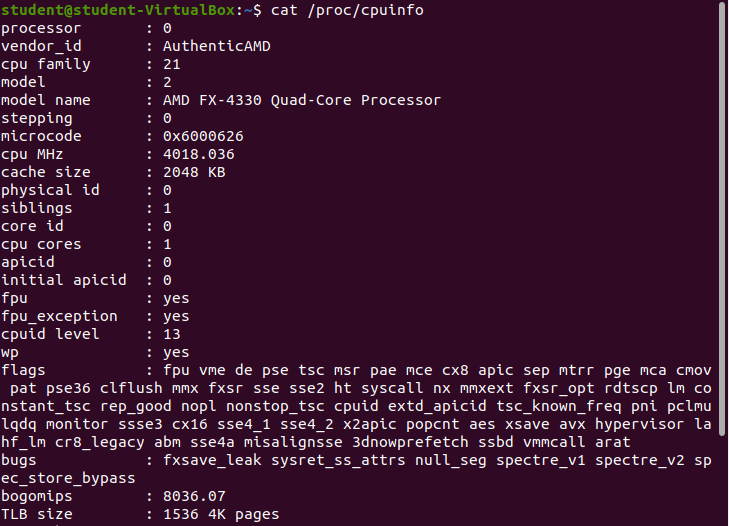


Рисунок 5.20– Фрагмент информации о процессоре ПК

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 формe. Для вывода воспользуемся командой cat.

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

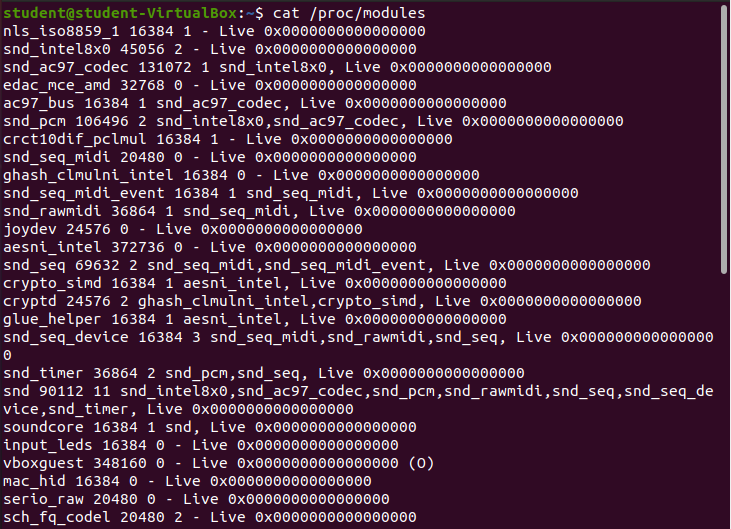


Рисунок 5.21 – Фрагмент списка модулей, используемых в настоящий момент ядром ОС

**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: а - 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.

* 1. Команда получения списка потоков.

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

* 1. Команда для завершения приложений.

Завершение процесса выполняется командой kill сигнал PID.

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

* 1. 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 распределеним; смещение от начала файла, где начинается распределение; старший и младший номера (в шестнадцатиричном виде) устройства, на котором находится распределенный файл; номер inode файла; и, наконец, имя самого файла.

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.

1. 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.

1. 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 l в колонке состояния ps.

Примером может служить браузер Firefox.

1. 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

1. 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.

1. 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 l в колонке состояния ps. Примером может служить браузер Firefox.

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

Каталог /procсодержит 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.

1. 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

1. Получение информации о состоянии системной памяти.

Текущее состояние системной памяти позволяет получить команда

По умолчанию все значения представлены в килобайтах. Значения в М

позволяет получить опция –m.

1. 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.

1. Назначение файловой системы /proc.

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

**Лабораторная работа № 8**

**Симметричная мультипроцессорная обработка**

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

**Ход работы:**

1. В работе оценивается трудоемкость простейших алгоритмов и эффективность их параллельного выполнения на многоядерных процессорах под управлением ОС MS Windows 10, поддерживающей SMP. Для оценки трудоемкости применяется оценка времени выполнения реализующей алгоритм программы на одном или нескольких ядрах ЦП.

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

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

4. Размер обрабатываемого массива следует задавать в пределах 100-500, при этом время выполнения приложения не должно быть менее 500 мсек.

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

6. Полученные результаты обработать: вычислить реальное значение выигрыша по производительности и сравнить со значением выигрыша, найденного по закону Амдала. Сравнить характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива) и асимптотической оценки трудоемкости алгоритма, который реализует исследуемая программа.

7. Выполнить исследование алгоритма умножения матриц и алгоритма быстрой сортировки. Исследование алгоритма состоит в последовательном выполнении пунктов 1-6.

**Описание выполнения работы**

Исследуем алгоритм умножения матриц. Установим размер массива   n  =  500 элементов.

**Алгоритм умножения матриц**

**uses** Arrays;

**uses** Utils;

**procedure** ParallelMult(a,b,c: **array** [,] **of** real; n: integer);

**begin**

{$omp parallel for }

**for var** i:=0 **to** n-1 **do**

**for var** j:=0 **to** n-1 **do**

**begin**

c[i,j]:=0;

**for var** l:=0 **to** n-1 **do**

c[i,j]:=c[i,j]+a[i,l]\*b[l,j];

**end**;

**end**;

**procedure** Mult(a,b,c: **array** [,] **of** real; n: integer);

**begin**

**for var** i:=0 **to** n-1 **do**

**for var** j:=0 **to** n-1 **do**

**begin**

c[i,j]:=0;

**for var** l:=0 **to** n-1 **do**

c[i,j]:=c[i,j]+a[i,l]\*b[l,j];

**end**;

**end**;

**var** n := 500;

**begin**

**var** a := Arrays.CreateRandomRealMatrix(n,n);

**var** b := Arrays.CreateRandomRealMatrix(n,n);

**var** c := **new** real[n,n];

ParallelMult(a,b,c,n);

writeln('Параллельное перемножение матриц:',Milliseconds,' миллисекунд');

**var** d := Milliseconds;

Mult(a,b,c,n);

writeln('Последовательное перемножение матриц: ',Milliseconds-d,' миллисекунд');

**end**.

Откроем системный монитор и добавим счетчик потоков для приложения «PascalABC.Net» (Рисунок 8.1). Зададим соответствие для процесса «PascalABCNet» в диспетчере задач: укажем использование первого ядра при выполнении алгоритма умножения матриц. (Рисунок 8.2).

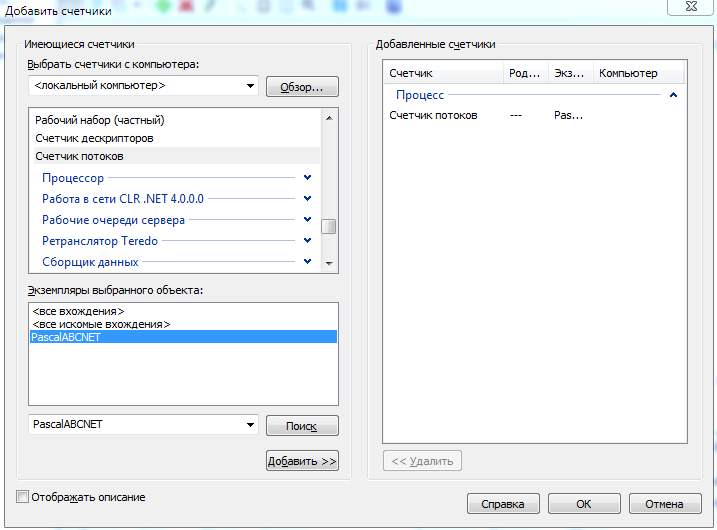


Рисунок 8.1 — Добавление счетчика потоков для приложения «PascalABC.Net»

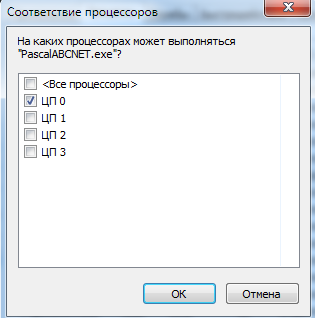


Рисунок 8.2 — Соответствие процессоров для приложения «PascalABC.exe»

Запустим приложение пять раз. Запуски программы и счетчик потоков для первого ядра представлены на рисунках 8.3-8.7.

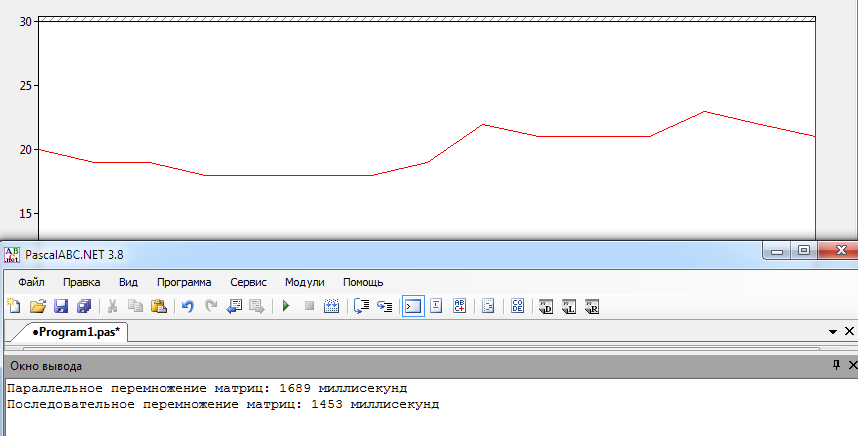


Рисунок 8.3 — Первый запуск программы

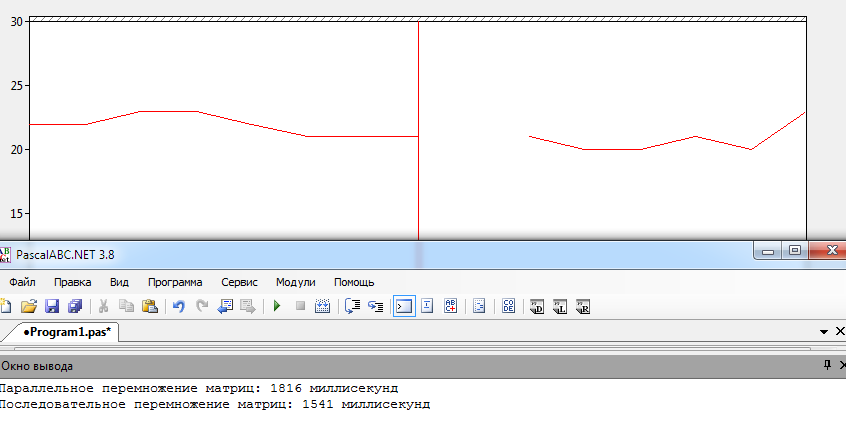


Рисунок 8.4 — Второй запуск программы

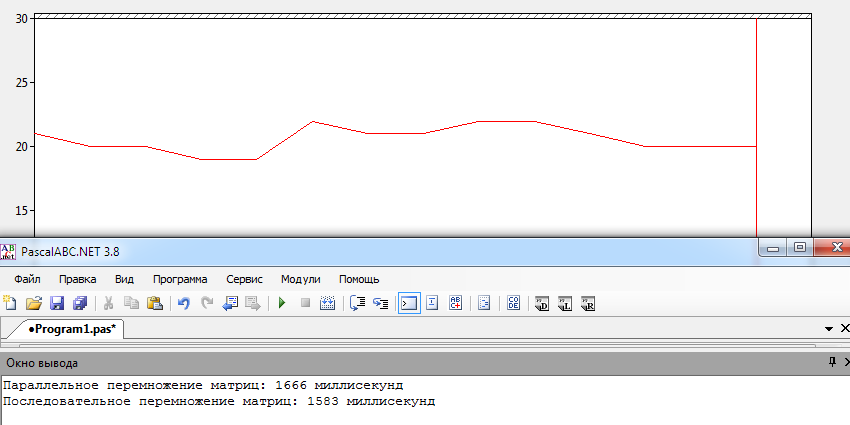


Рисунок 8.5 — Третий запуск программы

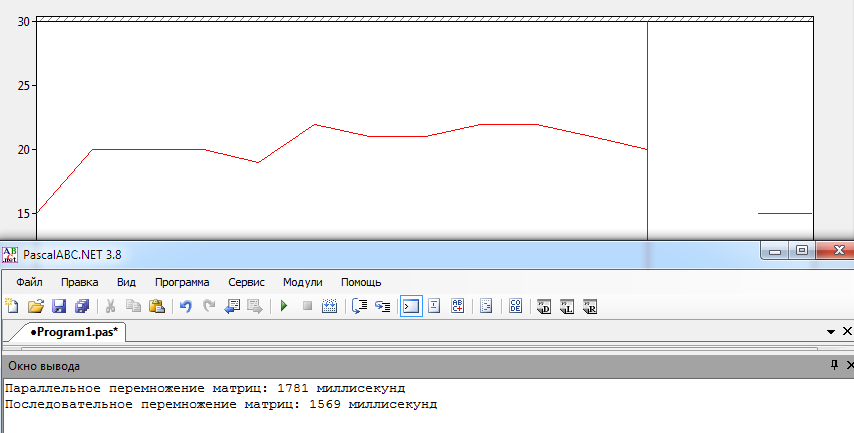


Рисунок 8.6 — Четвертый запуск программы



Рисунок 8.7 — Пятый запуск программы

Выполним аналогичные действия для двух, трех и четырех ядер. Полученные данные занесем в таблицу 8.1 и рассчитаем выигрыш в производительности эмпирически и по закону Амдала.

Таблица 8.1 – Данные выполнения алгоритма умножения матриц

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ядер | Тесты | Номер эксперимента | | | | | Min значение | Выигрыш в производительности | |
|  |  | 1 | 2 | 3 | 4 | 5 |  | По закону Амдала | Эмпирически |
| 1 | Параллельное | 1689 | 1816 | 1666 | 1781 | 2022 | 1666 | 1 | 0,87215 |
|  | Последовательное | 1453 | 1541 | 1583 | 1569 | 1576 | 1453 |  |  |
|  | Число потоков | 23 | 23 | 22 | 22 | 21 | 21 |  |  |
| 2 | Параллельное | 1485 | 1689 | 1497 | 1774 | 1781 | 1485 | 1,6667 | 0,9899 |
|  | Последовательное | 1515 | 1478 | 1470 | 1527 | 1525 | 1470 |  |  |
|  | Число потоков | 24 | 23 | 22 | 24 | 24 | 22 |  |  |
| 3 | Параллельное | 836 | 854 | 860 | 841 | 835 | 835 | 2,14286 | 1,76766 |
|  | Последовательное | 1518 | 1568 | 1476 | 1508 | 1563 | 1476 |  |  |
|  | Число потоков | 23 | 23 | 25 | 24 | 22 | 22 |  |  |
| 4 | Параллельное | 920 | 867 | 806 | 824 | 883 | 806 | 2,5 | 1,92928 |
|  | Последовательное | 1644 | 1579 | 1615 | 1608 | 1555 | 1555 |  |  |
|  | Число потоков | 22 | 22 | 25 | 25 | 24 | 22 |  |  |

На основе полученных экспериментальных данных можно сделать вывод о том, что выигрыш в производительности по закону Амдала приблизительно равен эмпирически полученному значению.

Сравним характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива). Вычисления будем производить на размерах массива от 550 до 800 (шаг 50 элементов) при задействованных четырех ядрах. Запуски программы продемонстрированы на рисунках 8.8-8.13.



Рисунок 8.8 — Запуск программы при размере массива = 550 элементов



Рисунок 8.9 — Запуск программы при размере массива = 600 элементов



Рисунок 8.10 — Запуск программы при размере массива = 650 элементов



Рисунок 8.11 — Запуск программы при размере массива = 700 элементов



Рисунок 8.12 — Запуск программы при размере массива = 750 элементов



Рисунок 8.13 — Запуск программы при размере массива = 800 элементов

Представим полученные данные в виде графика, представленного на рисунке 8.14. Исследуем алгоритм быстрой сортировки. Установим размер массива   n  = 4000000 элементов.

Рисунок 8.14 — Характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива)

Как видно из рисунка 8.14 параллельное выполнение алгоритма умножение матриц дает существенный выигрыш по сравнению с последовательным выполнением программы.

**Алгоритм быстрой сортировки**

**uses** Arrays;

**uses** Utils;

**procedure** ParallelQuickSort(**var** v: **array of** real; l,r:longint);

**var** i,j:longint;

w,q:real;

**begin**

i := l; j := r;

q := v[(l+r) **div** 2];

**repeat**

**while** (v[i] < q) **do** inc(i);

**while** (q < v[j]) **do** dec(j);

**if** (i <= j) **then**

**begin**

w:=v[i]; v[i]:=v[j]; v[j]:=w;

inc(i); dec(j);

**end**;

**until** (i > j);

{$omp parallel sections}

**begin**

{$omp parallel sections}

**if** (l < j) **then**

**begin**

ParallelQuickSort(v, l,j);

**end**;

{$omp parallel sections}

**if** (i < r) **then**

**begin**

ParallelQuickSort(v, i,r);

**end**

**end**

**end**;

**procedure** QuickSort(**var** v: **array of** real; l,r:longint);

**var** i,j:longint;

w,q:real;

**begin**

i := l; j := r;

q := v[(l+r) **div** 2];

**repeat**

**while** (v[i] < q) **do** inc(i);

**while** (q < v[j]) **do** dec(j);

**if** (i <= j) **then**

**begin**

w:=v[i]; v[i]:=v[j]; v[j]:=w;

inc(i); dec(j);

**end**;

**until** (i > j);

**if** (l < j) **then** QuickSort(v,l,j);

**if** (i < r) **then** QuickSort(v,i,r);

**end**;

**var** n := 2500000;

**begin**

**var** a := Arrays.CreateRandomRealArray(n,0,10);

**var** a\_copy := Arrays.CreateRandomRealArray(n,0,10);

**for var** i := 1 **to** length(a) **do**

a\_copy[i - 1] := a[i - 1];

**var** d := Milliseconds;

QuickSort(a, 0, n-1);

writeln(' Последовательная сортировка: ',Milliseconds-d,' миллисекунд');

d := Milliseconds;

ParallelQuickSort(a\_copy, 0, n-1);

writeln(' Параллельная сортировка: ',Milliseconds-d,' миллисекунд');

**end**.

Запустим приложение пять раз. Запуски программы и счетчик потоков для первого ядра представлены на рисунках 8.15-8.19.

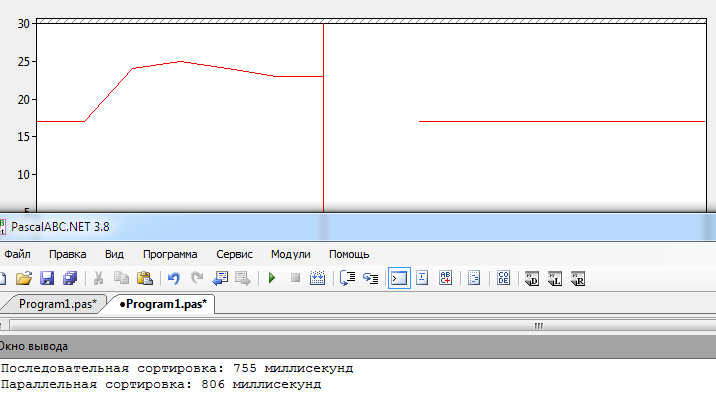


Рисунок 8.15 — Первый запуск программы

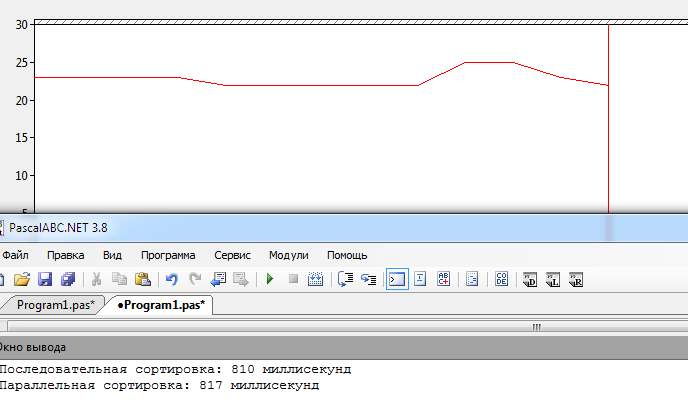


Рисунок 8.16 — Второй запуск программы

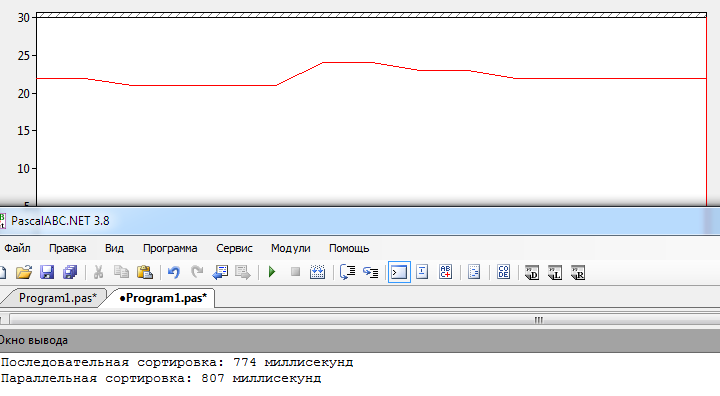


Рисунок 8.17 — Третий запуск программы

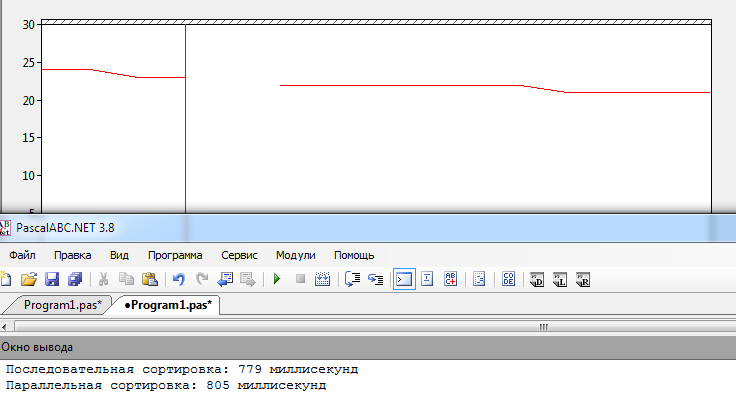


Рисунок 8.18 — Четвертый запуск программы

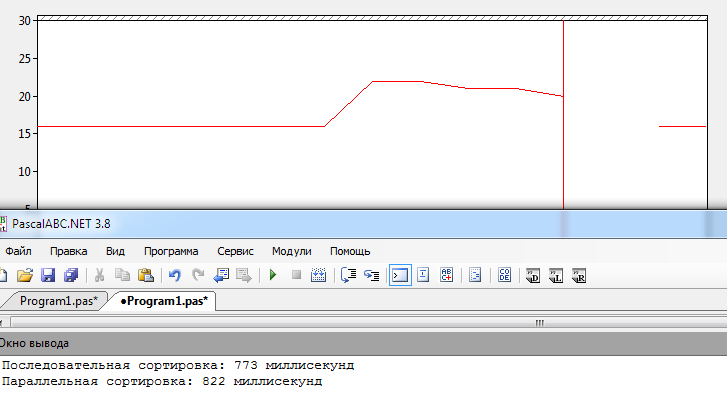


Рисунок 8.19 — Пятый запуск программы

Выполним аналогичные действия для двух, трех и четырех ядер. Полученные данные занесем в таблицу 8.2 и рассчитаем выигрыш в производительности эмпирически и по закону Амдала.

Таблица 8.2 – Данные выполнения алгоритма быстрой сортировки

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ядер | Тесты | Номер эксперимента | | | | | Min значение | Выигрыш в производительности | |
|  |  | 1 | 2 | 3 | 4 | 5 |  | По закону Амдала | Эмпирически |
| 1 | Параллельное | 806 | 817 | 807 | 805 | 822 | 805 | 1 | 0,93789 |
|  | Последовательное | 755 | 810 | 774 | 779 | 773 | 755 |  |  |
|  | Число потоков | 25 | 26 | 25 | 24 | 24 | 24 |  |  |
| 2 | Параллельное | 991 | 821 | 860 | 819 | 750 | 750 | 1,3333 | 1,0173 |
|  | Последовательное | 794 | 767 | 772 | 763 | 768 | 763 |  |  |
|  | Число потоков | 25 | 25 | 26 | 26 | 26 | 25 |  |  |
| 3 | Параллельное | 723 | 762 | 652 | 738 | 756 | 652 | 1,5 | 1,17025 |
|  | Последовательное | 765 | 799 | 773 | 765 | 763 | 763 |  |  |
|  | Число потоков | 25 | 24 | 26 | 26 | 26 | 24 |  |  |
| 4 | Параллельное | 618 | 764 | 626 | 718 | 479 | 479 | 1,6 | 1,58038 |
|  | Последовательное | 760 | 822 | 767 | 759 | 757 | 757 |  |  |
|  | Число потоков | 25 | 26 | 25 | 25 | 26 | 25 |  |  |

На основе полученных экспериментальных данных можно сделать вывод о том, что выигрыш в производительности по закону Амдала приблизительно равен эмпирически полученному значению.

Сравним характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива). Вычисления будем производить на размерах массива от 4000000 до 6000000 (шаг 500000 элементов) при задействованных четырех ядрах. Запуски программы продемонстрированы на рисунках 8.20-8.24.



Рисунок 8.20 — Запуск программы при размере массива = 4000000 элементов



Рисунок 8.21 — Запуск программы при размере массива = 4500000 элементов



Рисунок 8.22 — Запуск программы при размере массива = 5000000 элементов



Рисунок 8.23 — Запуск программы при размере массива = 5500000 элементов



Рисунок 8.24 — Запуск программы при размере массива = 6000000 элементов

Представим полученные данные в виде графика, представленного на рисунке 8.25.

Рисунок 8.25 — Характер изменения оценок реального времени выполнения программы (при различных размерах обрабатываемого массива)

Как видно из рисунка 8.25 параллельное выполнение алгоритма быстрой сортировки дает существенный выигрыш по сравнению с последовательным выполнением программы.

**Контрольные вопросы и ответы на них**

Вопрос 1. Симметричная и асимметричная архитектуры аппаратных и программных средств.

Симметричная архитектура мультипроцессорной системы предполагает однородность всех процессоров и единообразие включения процессоров в общую схему мультипроцессорной системы. Традиционные симметричные мультипроцессорные конфигурации разделяют общую оперативную память между всеми процессорами (ядрами процессоров). Масштабируемость (возможность наращивания числа процессоров) в симметричных системах ограничена вследствие того, что все они пользуются одной оперативной памятью и должны располагаться в одном корпусе. Такая конструкция, называемая масштабируемой по вертикали. В симметричных архитектурах обеспечивается достаточно высокая производительность для тех приложений, в которых несколько задач должны активно взаимодействовать между собой.

В асимметричной архитектуре разные процессоры могут отличаться как своими характеристиками, так и функциональной ролью, которая поручается им в системе. Функциональная неоднородность в асимметричных архитектурах влечет за собой структурные отличия во фрагментах системы, содержащих разные процессоры системы. Масштабирование в асимметричной архитектуре реализуется иначе, чем в симметричной. Так как требование единого корпуса отсутствует, система может состоять из нескольких устройств, каждое из которых содержит один или несколько процессоров. Это масштабирование по горизонтали. Каждое такое устройство называется кластером, а вся 94 мультипроцессорная система - кластерной.

Вопрос 2. Понятие SMP.

Наиболее распространенной целью объединения процессоров является симметричная мультипроцессорная обработка (SMP). В системе SMP каждый процессор решает свою задачу, порученную ему операционной системой. В документации Intel симметрия рассматривается в двух аспектах: - симметрия памяти — все процессоры пользуются общей памятью и одной копией ОС; - симметрия ввода-вывода — все процессоры разделяют общие устройства ввода-вывода и общие контроллеры прерываний.

Вопрос 3. Закон Амдала.

Для количественной оценки выигрыша в производительности ПК при параллельной работе нескольких ядер обычно используется закон Дж. Амдала (1967 г). Закон Амдала описывает максимальный теоретический выигрыш в производительности параллельного решения по отношению к лучшему последовательному.



В данном уравнении V – выигрыш в производительности при использовании n ядер центрального процессора, S – время, потраченное на выполнение последовательной части параллельной версии. При n=1 (одно ядро) ускорения нет. Если используется два ядра, которые половину всей работы выполняют параллельно, S=0,5 и V= 2 / 1,5 = 1,33. В случае выполнения всей работы двумя ядрами параллельно максимально возможный теоретический выигрыш равен 2

Вопрос 4. Трудоемкость алгоритма.

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

Вопрос 5. Трудоемкость алгоритмов умножения матриц, сложения матриц и сортировки массива методом пузырька.

Трудоемкость алгоритма сложения векторов A(n) и B(n) равна O(n), потому что количество операций сложения равно количеству элементов векторов. Трудоемкость алгоритма умножения квадратных матриц равна O(n3). Реализующая алгоритм программа умножения матриц содержит 3 вложенных арифметических цикла.

Вопрос 6. Трудоемкость алгоритма быстрой сортировки.

При оптимальном выборе ведущих элементов, когда разделение каждого блока происходит на равные по размеру части, трудоемкость алгоритма совпадает с быстродействием наиболее эффективных способов сортировки, то есть порядка O(n log n). В среднем случае количество операций, выполняемых алгоритмом быстрой сортировки, определяется выражением T(n) = O(1.4n log n).