Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ОТЧЕТ

к лабораторной работе по дисциплине:

|  |
| --- |
| **Администрирование информационных систем** |
| Язык сценариев ОС Microsoft Windows «Power Shell» |

наименование темы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы: | ИСМб-19-1 |  |  |  | Михейко Ю.А. |
|  |  | подпись |  | Фамилия И.О |
| Проверил: |  |  |  |  | Хритова М.А. |
|  | должность |  | подпись |  | Фамилия И.О |

Иркутск 2022 г.

Содержание

[1 Введение 2](#_Toc119719214)

[2 Краткая информация о языке 4](#_Toc119719215)

[3 Описание всех действий и листинг скриптов 5](#_Toc119719216)

[3.1 Описание используемых команд 5](#_Toc119719217)

[3.1.1 Описание скрипта 1 5](#_Toc119719218)

[3.1.2 Описание скрипта 2 5](#_Toc119719219)

[3.1.3 Описание скрипта 3 6](#_Toc119719220)

[3.2 Листинг кода 6](#_Toc119719221)

[3.2.1 Скрипт 1 6](#_Toc119719222)

[3.2.2 Скрипт 2 7](#_Toc119719223)

[3.2.3 Скрипт 3 9](#_Toc119719224)

[4 Результаты работы скрипта 10](#_Toc119719225)

[4.1 Работа скрипта 1 10](#_Toc119719226)

[4.2 Работа скрипта 2 11](#_Toc119719227)

[4.3 Работа скрипта 3 15](#_Toc119719228)

[5 Заключение 16](#_Toc119719229)

1 Введение

**Цель работы:** Освоить язык сценариев «Power Shell».

**Задание:**

1. Изучить теоретическую информацию о языке сценариев Power Shell.
2. Написать 2-3 простейших сценария, оформив их в виде скрипта. Варианты сценариев студент придумывает сам.

Примеры сценариев:

* Копирование нескольких файлов на сменный носитель в определенное время (учитывать возможность изменения буквы диска).
* Изменения прав доступа по шаблону в папке студента.
* Отобразить список служб на компьютере пользователя.
* Отсортировать все службы на компьютере пользователя на запущенные и остановленные.
* Создать подкаталог текущего каталога и скопировать в него файлы с расширениями PAS, BAS, C и TXT, присвоив им атрибут «только для чтения»;
* В заданном каталоге выдать на экран список имен файлов с расширениями PAS,TXT, BAT.
* Отсортировать указанный файл текущего каталога по убыванию 2-го символа и выдать его на экран.

1. Написать отчет с приложением листинга скрипта.

**Требования к отчету:** Отчетный документ представляется в печатном или электронном виде, в форматах doc(x).

Отчет оформляется согласно действующему нормоконтролю и содержит следующие главы:

1. Введение

2. Краткая информация о языке.

3. Описание всех действий и листинг скрипта с комментариями.

4. Результаты работы скрипта, со скриншотами.

5. Заключение (с анализом возможностей Power Shell).

2 Краткая информация о языке

PowerShell — расширяемое средство автоматизации от Microsoft с открытым исходным кодом, состоящее из оболочки с интерфейсом командной строки и сопутствующего языка сценариев. Впервые публично язык был продемонстрирован на Профессиональной конференции разработчиков в сентябре 2003 года под кодовым названием Monad. Изначально средство выпущено как компонент Windows под названием Windows PowerShell. Открытый исходный код был выпущен с началом разработки PowerShell Core в августе 2016 года.

Команды, исполняемые в Windows PowerShell, могут быть в форме командлетов, которые являются специализированными классами .NET, созданными с целью предоставления функциональности в PowerShell в виде сценариев PowerShell (.PS1) или являются обычными исполняемыми файлами. Если команда является исполняемым файлом, то PowerShell запускает её в отдельном процессе; если это команда, то он исполняется внутри процесса PowerShell. PowerShell предоставляет интерфейс командной строки, в котором можно вводить команды и отображать выводимые ими данные в текстовом виде. Этот пользовательский интерфейс, базирующийся на стандартном механизме консоли Windows, предоставляет настраиваемый механизм автозавершения команд, но не обладает возможностью подсветки синтаксиса, хотя при желании её можно обеспечить. В PowerShell также можно создавать псевдонимы (англ. alias) для командлетов, которые при вызове преобразуются в оригинальные команды. Кроме того, поддерживаются позиционные и именованные параметры для командлетов. При выполнении командлета работа по привязке значений аргументов к параметрам выполняется самим PowerShell, но при вызове внешних исполняемых файлов аргументы передаются им для самостоятельного разбора.

3 Описание всех действий и листинг скриптов

3.1 Описание используемых команд

3.1.1 Описание скрипта 1

Write-Host – записывает пользовательские выходные данные в узел.

-BackgroundColor – определяет цвет фона. Значение по умолчанию отсутствует.

-ForegroundColor – определяет цвет текста. Значение по умолчанию отсутствует.

Start-Sleep – приостанавливает действие сценария или сеанса на указанный период времени.

Add-Type – добавляет класс Microsoft .NET в сеанс PowerShell.

-AssemblyName – задает имя сборки, включающей типы. Add-Type принимает типы из указанной сборки. Этот параметр требуется при создании типов на основе имени сборки.

system.windows.media.mediaplayer – предоставляет воспроизведение мультимедиа.

Open(Uri) – открывает заданный Uri для воспроизведения мультимедиа.

Get-ChildItem – получает элементы и дочерние элементы в одном или нескольких указанных расположениях.

-path – указывает путь к одному или нескольким расположениям.

-include – задает массив из одного или нескольких строковых шаблонов, которые будут сопоставляться при получении командлетом дочерних элементов.

-recurse – получает элементы в указанных расположениях и во всех дочерних элементах местоположений.

Play() – воспроизводит мультимедиа с текущего Position.

Stop() – останавливает воспроизведения мультимедиа.

Start-Sleep – приостанавливает действие сценария или сеанса на указанный период времени.

-Seconds – указывает, как долго ресурс бездействует в секундах.

3.1.2 Описание скрипта 2

Write-Host – записывает пользовательские выходные данные в узел.

-BackgroundColor – определяет цвет фона. Значение по умолчанию отсутствует.

-ForegroundColor – определяет цвет текста. Значение по умолчанию отсутствует.

Switch – передает управление в оператор внутри своего тела.

Read-Host – считывает строку ввода из консоли.

get-process – возвращает процессы, выполняемые на локальном компьютере.

-Id - указывает один или несколько процессов по идентификатору процесса.

Stop-Process – останавливает один или несколько запущенных процессов.

-Id – указывает идентификаторы процессов, которые необходимо остановить.

-Confirm - запрос подтверждения перед выполнением командлета.

3.1.3 Описание скрипта 3

Read-Host – считывает строку ввода из консоли.

Get-ChildItem – получает элементы и дочерние элементы в одном или нескольких указанных расположениях.

-Path – указывает путь к одному или нескольким расположениям.

-recurse – получает элементы в указанных расположениях и во всех дочерних элементах местоположений.

Where-Object – выбирает объекты из коллекции на основании значения их свойств.

-gt – указывает, что этот командлет получает объекты, если значение свойства больше указанного значения.

Sort-Object – сортирует объекты по значениям свойств.

-Property – указывает имена свойств, которые команда Sort-Object использует для сортировки объектов. Length – ключ, указывающий на сортировку по величине.

Out-File – отправляет вывод в файл.

3.2 Листинг кода

3.2.1 Скрипт 1

#Аудио-плеер из папки (по 30 секунд на трек)

$path = Read-Host "Укажите путь к папке с музыкой." #Выводим текст о с просьбой ввода и считываем вводимое значение

#Имитация обработки

Write-Host "Обработка." -BackgroundColor Black -ForegroundColor Green #Вывод в консоль текста

Start-Sleep 1 #Делей на секунду

Write-Host "Обработка.." -BackgroundColor Black -ForegroundColor Green

Start-Sleep 1

Write-Host "Обработка..." -BackgroundColor Black -ForegroundColor Green

Start-Sleep 1

Add-Type -AssemblyName presentationCore #Подключаем окружение .Net для работы с плеером

$mediaPlayer = New-Object system.windows.media.mediaplayer #Инициализируем объект Media Player

$musicPath = $path #Сохраняем в переменную путь к файлу

$mediaPlayer.open([uri]"$($file.fullname)") #Открываем объект Media Player

$musicFiles = Get-ChildItem -path $musicPath -include \*.mp3,\*.wma -recurse #Инициализируем объект аудио-файла с путем к файлам и с расширением .mp3 и .wma

#и параметром рекурсии по вложенным папкам -recurse

foreach($file in $musicFiles) #Перебираем файлы с папке по указанному пути

{

"Сейчас играет: $($file.BaseName)" #Сообщаем в консоль название проигрываемого файла

$mediaPlayer.open([uri]"$($file.fullname)") #Открываем объект Media Player

$mediaPlayer.Play() #Проиграваем файл

Start-Sleep -Seconds 30 #Время проигрывания - 30 секунд

$mediaPlayer.Stop() #Останавливаем проигрывание

3.2.2 Скрипт 2

#Диспетчер Задач "на минималках"

#Отрисовываем консольное меню

Write-Host #отступ строки (пустой)

Write-Host "Меню" -BackgroundColor Black -ForegroundColor Red #Надпись Меню в консоли с покраской заднего фона в черный и текст в красный

Write-Host #отступ строки (пустой)

#Вторая часть меню с пунктами выбора (варианты вводимых значений с зеленым цветом текста)

Write-Host "1. Просмотреть процессы" -ForegroundColor Green

Write-Host "0. Выход" -ForegroundColor Green

Write-Host

function readSubMenu {

$choice2 = Read-Host "Введите значение для работы"#Выводим текст о с просьбой ввода и считываем вводимое значение

switch ($choice2) {#Провиряем с помощью Switch введенный результат (если введено числовое значение из пункта меню - вызываем соответствующий метод.

#Иначе - просим ввести корректное значение)

1 { #Если выбран первый пункт, то

$idSelect = Read-Host "Введите id процесса для просмотра" #Выводим текст о с просьбой ввода и считываем вводимое значение

Write-Host "-----================Загружаю информацию================-----" -ForegroundColor Green

get-process -id $idSelect #Вывод информации о процессе по его id

}

2 { #Если выбран второй пункт, то

$idSelect = Read-Host "Введите id процесса для завершения" #Выводим текст о с просьбой ввода и считываем вводимое значение

Write-Host "-----================Останавливаю процесс================-----" -ForegroundColor Red

stop-process -id $idSelect -Confirm #Остановка процесса по его id

Write-Host "-----================Вывожу процессы================-----" -ForegroundColor Green

get-process

}

0{Write-Host "До скорой встречи!"; exit}

default {

Write-Host "Ошибка выбора! Попробуйте еще раз!" -ForegroundColor Red

readSubMenu

}

}

}

function readCommand{

$choice = Read-Host "Введите значение для работы"#Выводим текст о с просьбой ввода и считываем вводимое значение

Switch($choice){#Провиряем с помощью Switch введенный результат (если введено числовое значение из пункта меню - вызываем соответствующий метод.

#Иначе - просим ввести корректное значение)

1{

get-process

Write-Host

Write-Host "Работа с процессом" -BackgroundColor Black -ForegroundColor Red

Write-Host

Write-Host "1. Выдать информацию о процессе" -ForegroundColor Green

Write-Host "2. Завершить процесс" -ForegroundColor Green

Write-Host "0. Выход" -ForegroundColor Green

Write-Host

readSubMenu

}

0{Write-Host "До скорой встречи!"; exit}

default {

Write-Host "Ошибка выбора! Попробуйте еще раз!" -ForegroundColor Red

readCommand

}

}

}

readCommand

3.2.3 Скрипт 3

#Ищем в указанной папке и подпапках файлы с размером больше 100 байт

#Указываем путь к папке

#Передаем путь в считыватель файлов с параметром рекурсии для просмотра файлов в подпапках, передаем эти файлы дальше

#Если размер файла больше ста байт, то передаем информацию

#Сортирем полученные файлы по размеру с помощью ключа свойств Размера и передаем информацию

#Полученную информацию сохраняем в файл length.txt

$path = Read-Host "Укажите путь к папке"

Get-ChildItem -Path ${$path} -recurse |

Where-Object {$\_.length -gt 100} |

Sort-Object -Property length |

Out-File -FilePath length.txt

4 Результаты работы скрипта

4.1 Работа скрипта 1

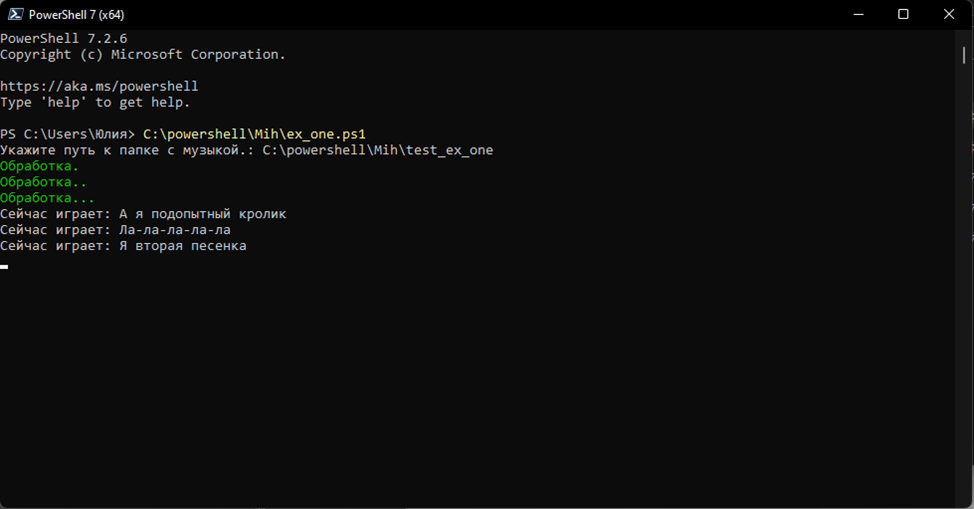


Рисунок 1 – Успешное проигрывание аудио-файлов

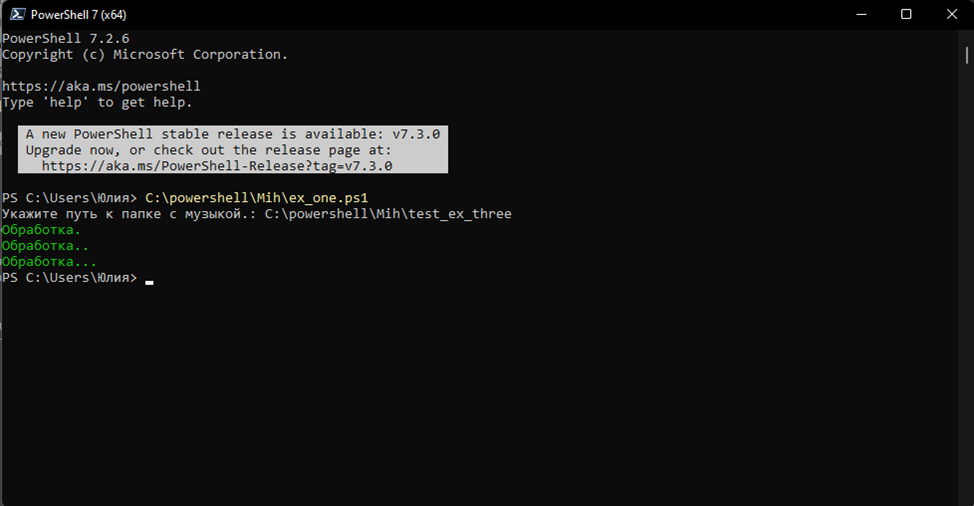


Рисунок 2 – Загрузка папки без аудио-файлов

4.2 Работа скрипта 2

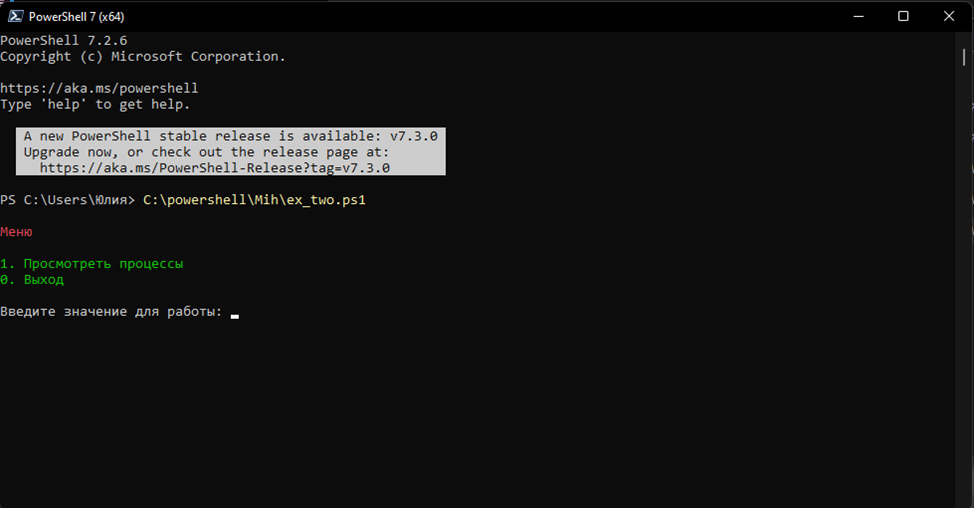


Рисунок 3 – Вывод основного меню

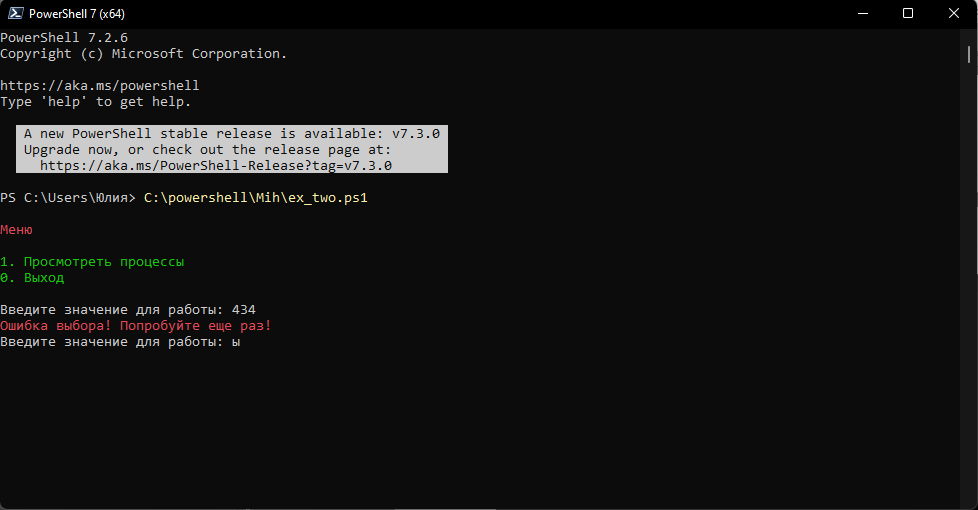


Рисунок 4 – Ввод некоректного значения в меню

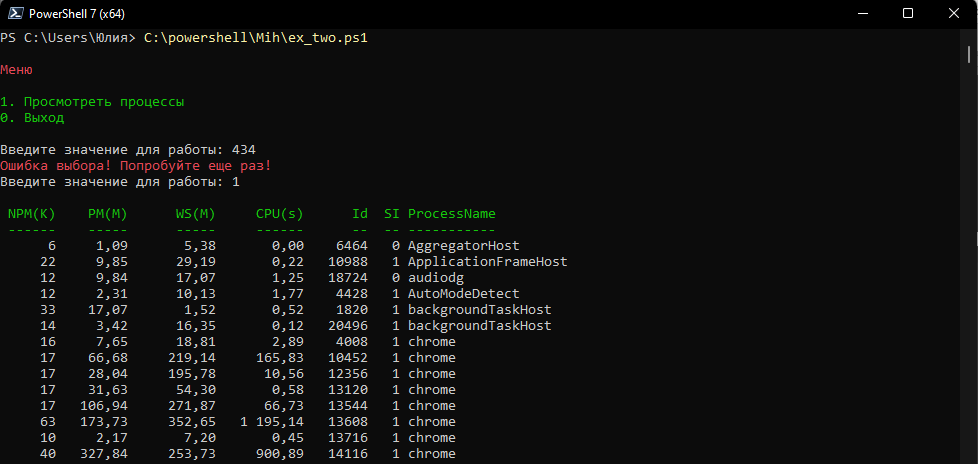


Рисунок 5 – Отображение процессов

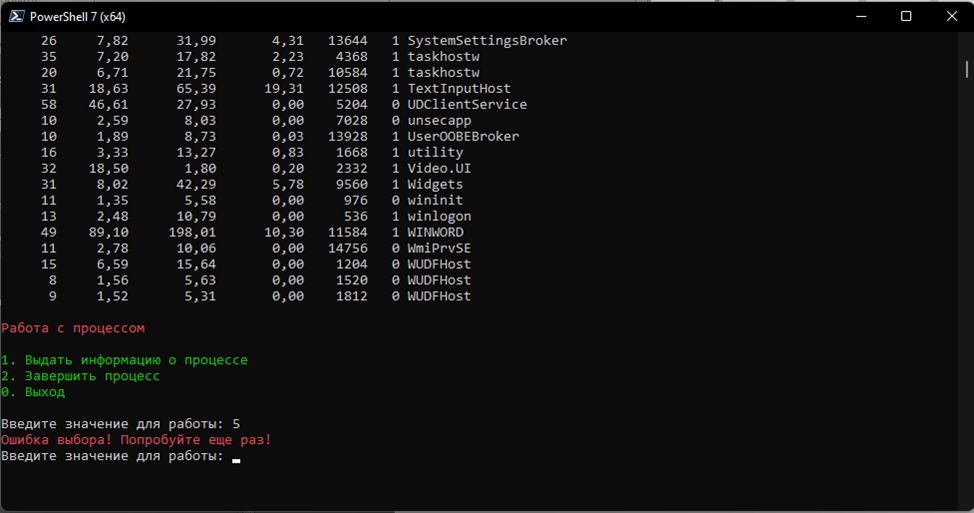


Рисунок 6 – Ввод некоректного значения в подменю

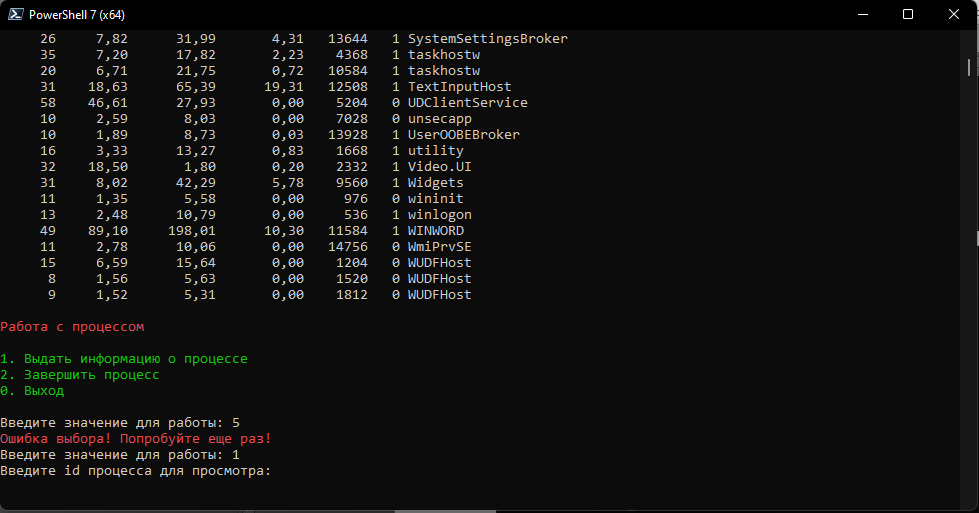


Рисунок 7 – Работа в подменю подменю

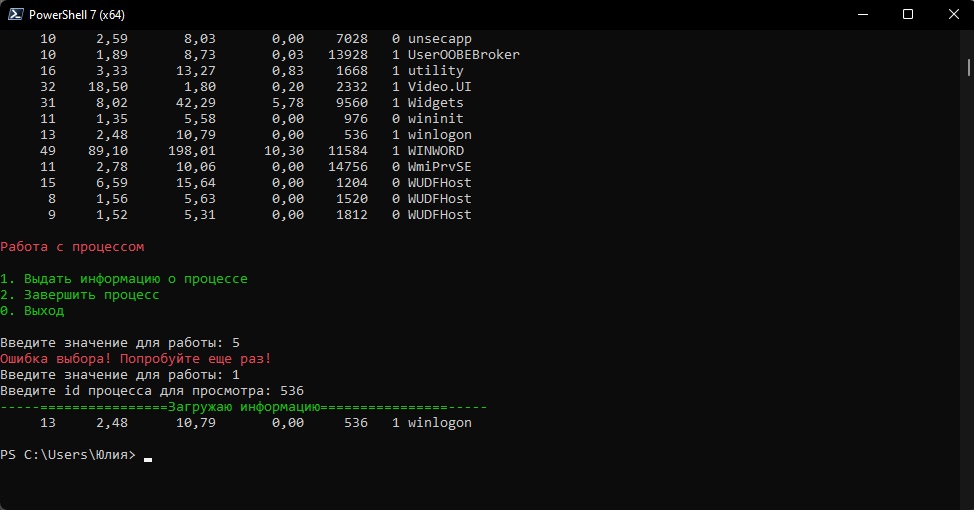


Рисунок 8 – Вывод информации о процессе

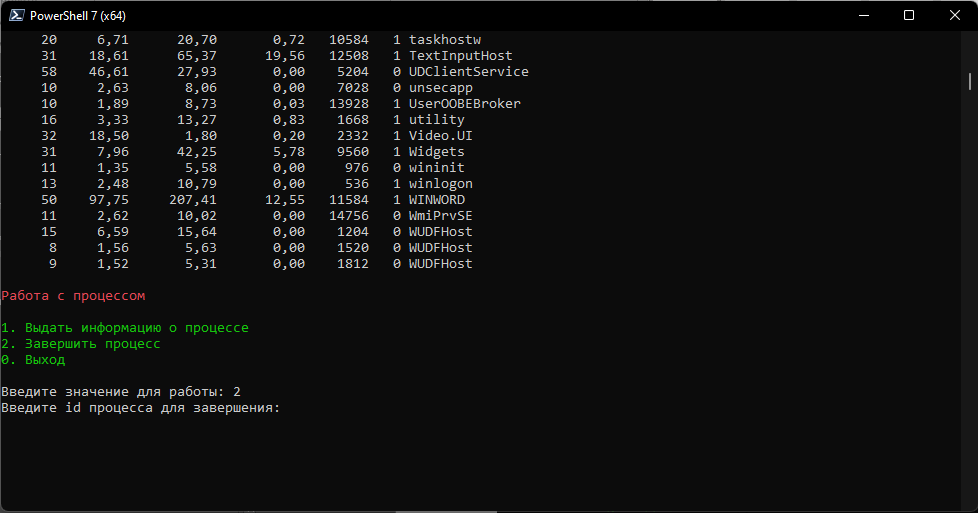


Рисунок 9 – Вызов меню остановки процессов

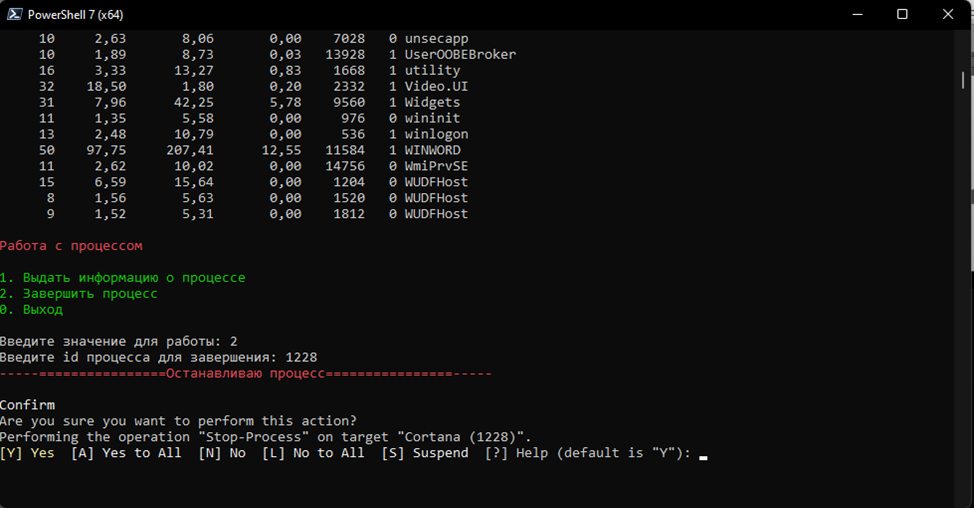


Рисунок 10 – Завершение процесса

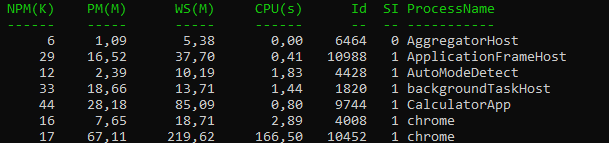


Рисунок 11 – Остановка процесса CalculatorApp

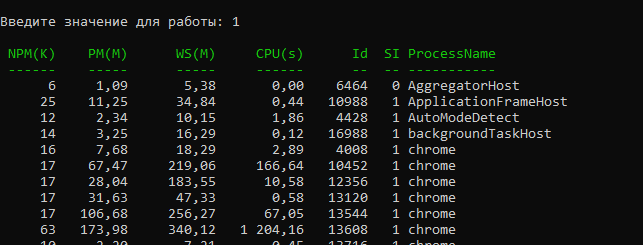


Рисунок 12 – Результат остановки процесса CalculatorApp

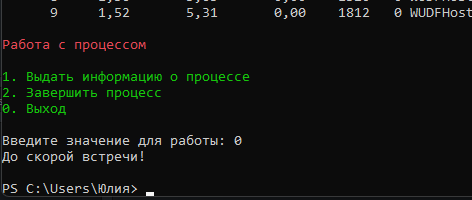


Рисунок 13 – Завершение работы скрипта

4.3 Работа скрипта 3

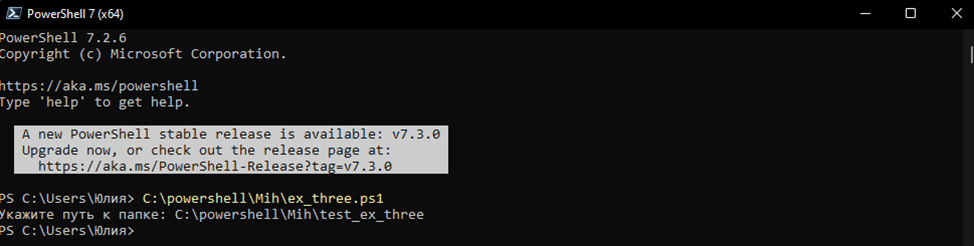


Рисунок 14 – Ввод пути к папке с файлами



Рисунок 15 – Файл с результатами работы скрипта



Рисунок 16 – Файл с результатами работы скрипта

5 Заключение

Несмотря на возможность выполнения большинства функций командной строки в Windows PowerShell, функционал приложений не совпадает.

В то время как Cmd использует команды DOS, PowerShell использует команды, известные как командлеты, которые представляют собой независимые программные объекты, позволяющие получить доступ к основным параметрам администрирования в Windows. Эти параметры недоступны из Cmd, что означает, что пользователи должны перемещаться по графическому интерфейсу, чтобы получить к ним доступ.

Cmd – это просто система оболочки, которая ограничивает пользователей выполнять только простые и базовые сценарии для выполнения командного файла. Однако PowerShell – это и оболочка, и среда сценариев, которая поддерживает создание больших и сложных файлов для управления системой Windows.

Командная строка не предназначена для системного администрирования, и пользователям требуются дополнительные утилиты для расширения ее функциональности. Напротив, в Windows PowerShell есть множество командлетов, которые упрощают жизнь системным администраторам. Он позволяет выполнять широкий спектр задач и автоматизацию, включая управление пользователями и разрешениями, конфигурации безопасности, извлечение данных и администрирование Active Directory.

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