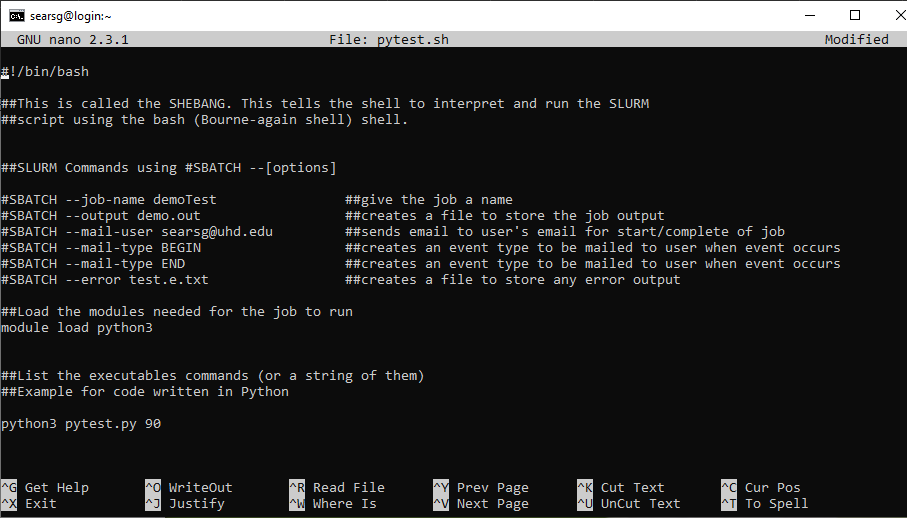
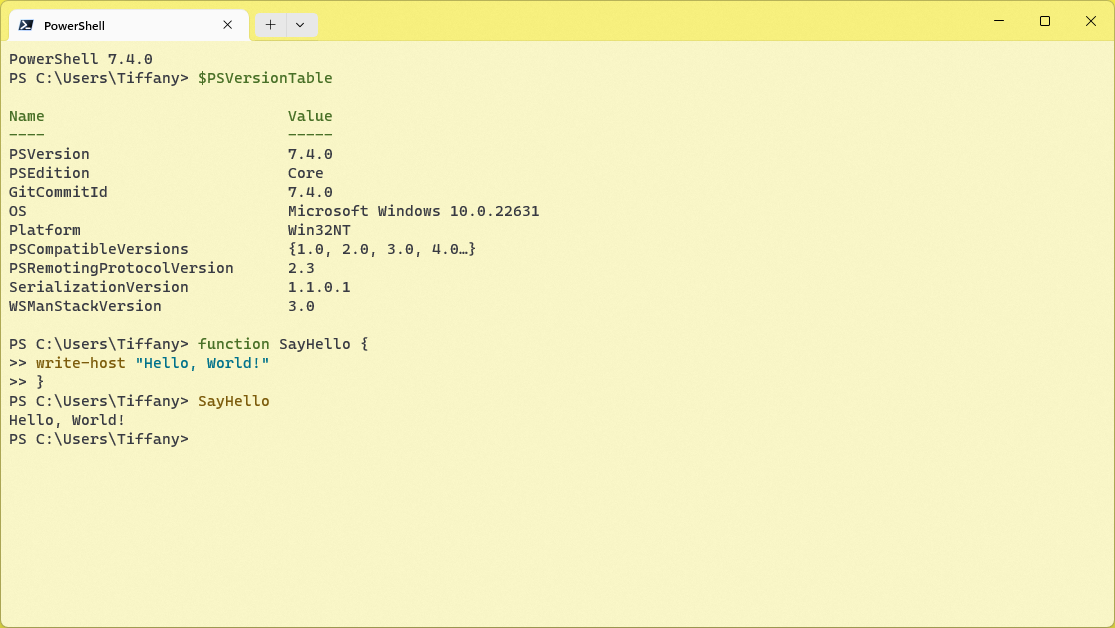
|  |  |
| --- | --- |
| **CSOPESY Major Output: Process Scheduler and CLI** | Created By: Neil Patrick Del Gallego, PhD |

By group

**[100 pts] General Instructions:** The first part of your emulator is the process multiplexer and your command-line interpreter (CLI).





**Shell Reference**

Please refer to a general Linux/Windows powershell/Windows command line. This serves as a strong reference for the design of your command-line interface.

For the process multiplexer, refer to the Linux “screen” command on its behavior: <https://www.geeksforgeeks.org/screen-command-in-linux-with-examples/>

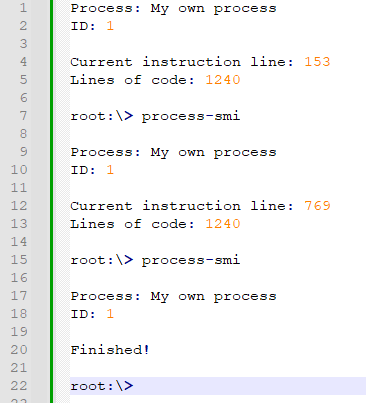
**Checklist of Requirements**

Your system must have ALL the following features implemented properly.

|  |  |
| --- | --- |
| **Requirement** | Main menu console |
| **Description** | A main menu console for recognizing the following commands:  “initialize” – initialize the processor configuration of the application. This must be called before any other command could be recognized, aside from “exit”.  “exit” – terminates the console.  “screen” – see additional details.  “scheduler-test” – continuously generates a batch of dummy processes for the CPU scheduler. Each process is accessible via the “screen” command.  “scheduler-stop” – stops generating dummy processes.  “report-util” – for generating CPU utilization report. See additional details. |
| **Requirement** | “screen” command support |
| **Description** | From the main menu, the user can perform the following:   * Create a new process via “screen -s <process name>” command. * Lists all running processes via “screen -ls” command. |
| **Requirement** | Generation of CPU utilization report |
| **Description** | The console should be able to generate a utilization report whenever the “report-util” command is entered. |
| **Requirement** | Configuration setting |
| **Description** | The “initialize” commands should read from a “config.txt” file, the parameters for your CPU scheduler and process attributes. |

**The “screen” command specifications**

The "screen" command emulates the screen multiplexer of Linux OS. Below is a CLI mockup of the screen command:



When the user types “**screen -s <process name>**” from the main menu console, the console will clear its contents and “move” to the process screen (lines 1 – 7). From there, the user can type the following:

* “process-smi” – Prints a simple information of the process (lines 9 – 13). The process contains dummy instructions that the CPU executes in the background. Every time the user types “process-smi”, it provides the updated details (e.g. lines 15 – 20). If the process has finished, simply print “Finished!” after the process name and ID has been printed (e.g. lines 17 – 20).
* “exit” – Returns the user to the main menu.

The range of instruction length per process, can be set through the “config.txt”

At any given time, any process can finish its execution. If this happens, the user can no longer access the screen after exiting.

The user can access the screen anytime by typing “**screen -r <process name>”** in the main menu. If the process name is not found/finished execution, the console simply prints “Process <process name> not found.”

Note that for the purpose of debugging/validating the correctness of your program, all finished and currently running processes must be reported in the “report-util” command.

**The “scheduler-test” and “scheduler-stop” commands**

To facilitate and stress-test the capabilities of your console, we should provide support for generating a batch of dummy processes.

“scheduler-test” – The behavior is as follows: Every X CPU cycles, a new process is generated and put into the ready queue for your CPU scheduler. This frequency can be set in the “config.txt.” As long as CPU cores are available, each process can be executed and be accessible via the “screen” command.

“scheduler-stop” – Stops generating dummy processes.

These commands are only accessible in the main menu console.

You must generate human-readable process names for the processes generated by the “scheduler-test” command to conveniently access them using the “screen -s <process name>” command described earlier. E.g.: p01, p02, …, p1240.

**The “screen -ls” and “report-util” commands**

These commands should be similar. The only difference is that “report-util” saves this into a text file – “csopesy-log.txt.” See sample mockup:



The “screen-ls” commands should list the CPU utilization, cores used, and cores available, as well as print a summary of the running and finished processes (lines 38 – 54). The “report-util” command saves the same info in the csopesy-log.txt file.

**The scheduler**

Your CPU scheduler is real-time and will continuously schedule processes as long as your console is alive. The scheduler algorithm will be set through the “initialize” command and through the “config.txt” file.

**The CPU cycle**

For simplicity, assume that the CPU cycle is an integer counter that tallies the number of CPU ticks. See pseudocode below:

A computer code with text

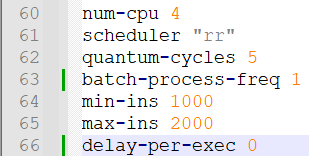
Description automatically generateds

**The config.txt file** **and “initialize” command**

The user must first run the “initialize” command. No other commands should be recognized if the user hasn’t typed this first. Once entered, it will read the “config.txt” file, which is space-separated in format, containing the following parameters.

|  |  |
| --- | --- |
| Parameter | Description |
| num-cpu | Number of CPUs available. The range is [1, 128]. |
| scheduler | The scheduler algorithm: “fcfs” or “rr”. |
| quantum-cycles | The time slice is given for each processor if a round-robin scheduler is used. Has no effect on other schedulers. The range is [1, . |
| batch-process-freq | The frequency of generating processes in the “scheduler-test” command in CPU cycles. The range is [1, . If one, a new process is generated at the end of each CPU cycle. |
| min-ins | The minimum instructions/command per process. The range is [1, . |
| max-ins | The maximum instructions/command per process. The range is [1, . |
| delays-per-exec | Delay before executing the next instruction in CPU cycles. The delay is a “busy-waiting” scheme wherein the process remains in the CPU. The range is [0, . If zero, each instruction is executed per CPU cycle. |

The default parameters and sample “config.txt” can be seen below:



**ASSESSMENT METHOD**

Your CLI emulator will be assessed through a black box quiz system in a time-pressure format. This is to minimize drastic changes or “hacking” your CLI to ensure the test cases are met. You should only modify the parameters and no longer recompile the CLI when taking the quiz.

Test cases, parameters, and instructions are provided per question, wherein you must submit a video file (.MP4), demonstrating your CLI. Some questions will require submitting PowerPoint presentations, such as cases explaining the details of your implementation.

**IMPORTANT DATES**

See AnimoSpace for specific dates.

|  |  |
| --- | --- |
| **Week 7** | Mockup test case and quiz |
| **Week 8** | Actual test case and quiz |

**Submission Details**

Aside from video files for the quiz, you need to prepare some of the requirements in advance, such as:

* + SOURCE - Contains your source code. Add a README.txt with your name and instructions on running your program. Also, indicate the entry class file where the main function is located. An alternative can be a GitHub link.
  + PPT – A technical report of your system containing:
  + Command recognition
  + Console UI implementation
  + Command interpreter implementation
  + Process representation
  + Scheduler implementation

**Grading Scheme**

* You are to provide evidence for each test case, recorded through video. Each test case will have some points allocated. The test cases will be graded as follows:

|  |  |  |
| --- | --- | --- |
| **Robustness** | | |
| No points | Partial points | Full points |
| The CLI did not pass the test case. **NO WORKAROUND** is available to produce the expected output. | The CLI did not pass the test case. **A workaround** is available to produce the expected output. | The CLI passed the test case using varying inputs and produced the expected output. |