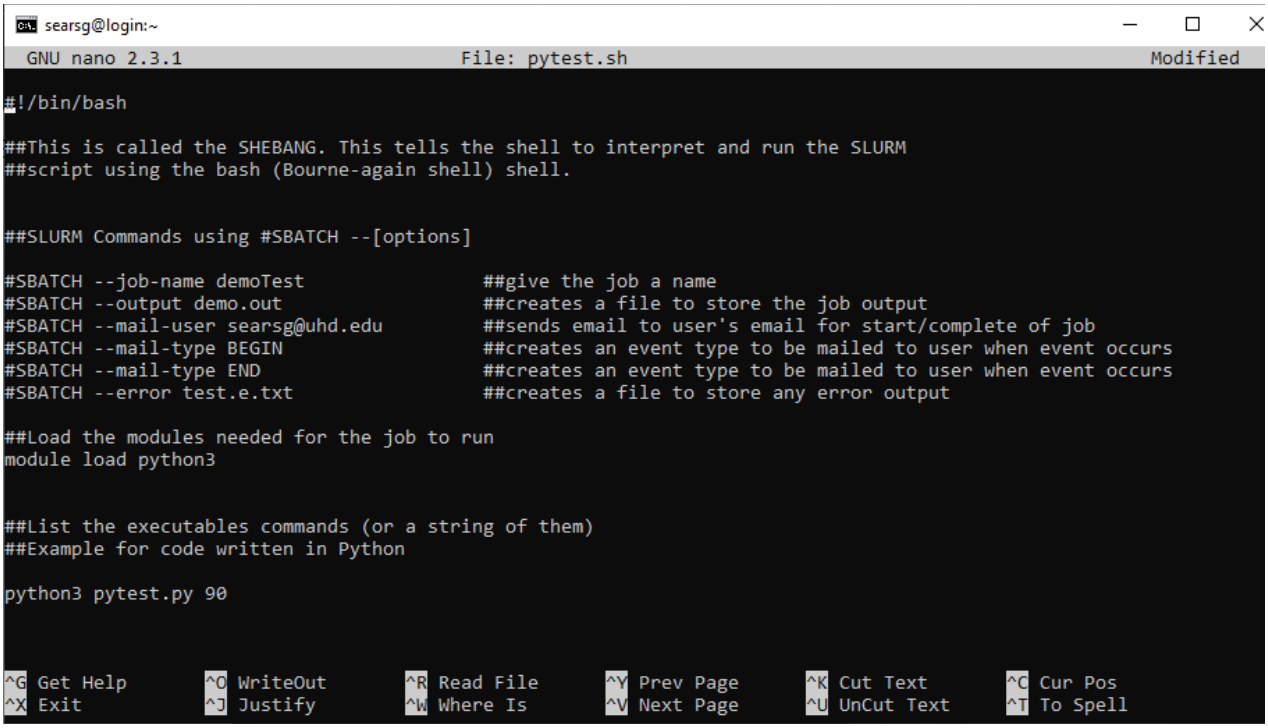
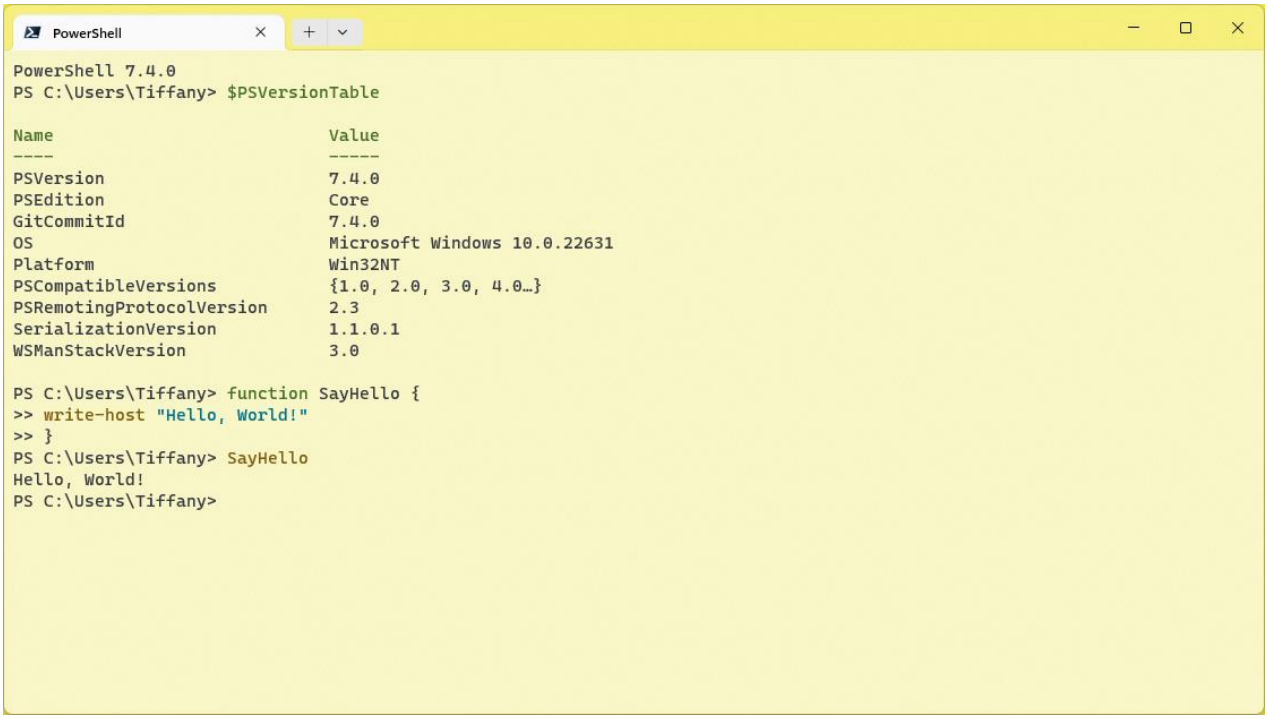
CSOPESY Major Output: Process Scheduler and CLI

Prepared by: Gregory Cu
Created By: Neil Patrick Del Gallego, PhD

Updated as of: Oct. 06, 2025

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





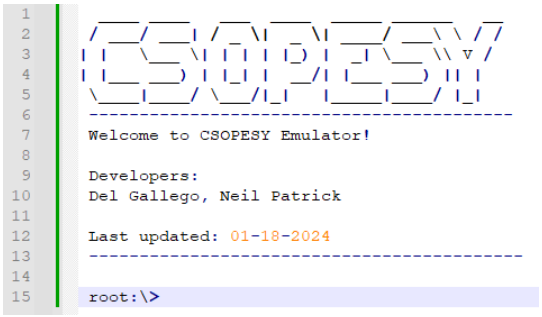
Project Grouping

This is a group project and should observe the same group grouping for the next major output.

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-start" (formerly 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	Barebones process instructions
Description	Support basic process instructions, akin to programming language instructions: • PRINT (msg) – display an output "msg" to the console. The output can only be seen when the user is inside its attached screen. The "msg" can print 1 variable, "var." E.g. PRINT ("Value from: " +x) • DECLARE (var, value) – declares a uint16 with variable name "var", and a default "value". • ADD (var1, var2/value, var3/value) – performs an addition operation: var1 = var2/value + var3/value var1, var2, var3 are variables. Variables are automatically declared with a value of 0 if they have not yet been declared beforehand. Can also add a uint16 value. • SUBTRACT (var1, var2/value, var3/value) – performs a subtraction operation: var1 = var2/value - var3/value • SLEEP (X) – sleeps the current process for X (uint8) CPU ticks and relinquishes the CPU. • FOR([instructions], repeats) – performs a for-loop, given a set/array of instructions. Can be nested. NOTES • Process instructions are pre-determined and not typed by the user. E.g., randomized via scheduler-start command. • Variables are stored in memory and will not be released until the process finishes.

	• uint16 variables are clamped between (0, max(uint16)). • Unless specified in the test case, the "msg" in the PRINT function should always be "Hello world from <process_name>!" • For loops can be nested up to 3 times.
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:

```
162  Process name: screen_01
163  ID: 1
164  Logs:
165  (08/06/2024 09:15:22AM) Core:0 "Hello world from screen_01!"
166  (08/06/2024 09:32:09AM) Core:1 "Hello world from screen_01!"
167
168  Current instruction line: 153
169  Lines of code: 1240
170
171  root:\> process-smi
172
173  Process name: screen_01
174  ID: 1
175  Logs:
176  (08/06/2024 09:15:22AM) Core:0 "Hello world from screen_01!"
177  (08/06/2024 09:15:28AM) Core:0 "Hello world from screen_01!"
178  (08/06/2024 09:32:09AM) Core:1 "Hello world from screen_01!"
179  (08/06/2024 09:33:12AM) Core:1 "Hello world from screen_01!"
180
181  Current instruction line: 769
182  Lines of code: 1240
183
184  root:\> process-smi
185
186  Process name: screen_01
187  ID: 1
188  Logs:
189  (08/06/2024 09:15:22AM) Core:0 "Hello world from screen_01!"
190  (08/06/2024 09:15:28AM) Core:0 "Hello world from screen_01!"
191  (08/06/2024 09:15:50AM) Core:0 "Hello world from screen_01!"
192  (08/06/2024 09:32:09AM) Core:1 "Hello world from screen_01!"
193  (08/06/2024 09:33:10AM) Core:1 "Hello world from screen_01!"
194  (08/06/2024 09:33:12AM) Core:1 "Hello world from screen_01!"
195
196  Finished!
197
198  root:\>
```

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 162 onwards). From there, the user can type the following:

- "process-smi" – Prints a simple information about the process (lines 9 – 13). The process contains dummy instructions that the CPU executes in the background. Whenever the user types "process-smi", it provides the updated details and accompanying logs from the print instructions. (e.g., lines 162 – 170). If the process has finished, simply print "Finished!" after the process name, ID, and logs have 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." Instruction types are randomized.

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 prints "Process <process name> not found."

Note that to debug/validating the correctness of your program, all finished and currently running processes must be reported in the "report-util" command.

The "scheduler-start" 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-start" – The behavior is as follows: Every X CPU ticks, 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:

```
25   ____ ____   ___  ____  _____ ______   __
26  / ___/ ___| / _ \|  _ \| ____/ ___\ \ / /
27 | |   \___ \| | | | |_) |  _| \___ \\ V /
28 | |___ ___) | |_| |  __/| |___ ___) || |
29  \____|____/ \___/|_|   |_____|____/ |_|
30  ----------------------------------------
31  Welcome to CSOPESY Emulator!
32
33  Developers:
34  Del Gallego, Neil Patrick
35
36  Last updated: 01-18-2024
37
38  root:\> screen -ls
39  CPU utilization: 100%
40  Cores used: 4
41  Cores available: 0
42
43  ---------------------------------------
44  Running processes:
45  process05   (01/18/2024 09:15:22AM)    Core: 0     1235 / 5876
46  process06   (01/18/2024 09:17:22AM)    Core: 1     3 / 5876
47  process07   (01/18/2024 09:17:45AM)    Core: 2     9 / 1000
48  process08   (01/18/2024 09:18:58AM)    Core: 3     12 / 80
49
50  Finished processes:
51  process01   (01/18/2024 09:00:21AM)    Finished    5876 / 5876
52  process02   (01/18/2024 09:00:22AM)    Finished    5876 / 5876
53  process03   (01/18/2024 09:00:42AM)    Finished    1000 / 1000
54  process04   (01/18/2024 09:00:53AM)    Finished    80 / 80
55  ---------------------------------------
56
57  root:\> report-util
    root:\> Report generated at C:/csopesy-log.txt!
```

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

```
int main() {
    int cpuCycles = 0;

    while(<OS is running>) {
        cpuCycles++;
    }
}
```

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, 2^{32}].
batch-process-freq	The frequency of generating processes in the "scheduler-start" command in CPU cycles. The range is [1, 2^{32}]. 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, 2^{32}].
max-ins	The maximum instructions/command per process. The range is [1, 2^{32}].
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, 2^{32}]. If zero, each instruction is executed per CPU cycle.

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

```
num-cpu 4
scheduler "rr"
quantum-cycles 5
batch-process-freq 1
min-ins 1000
max-ins 2000
delay-per-exec 0
```

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 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.