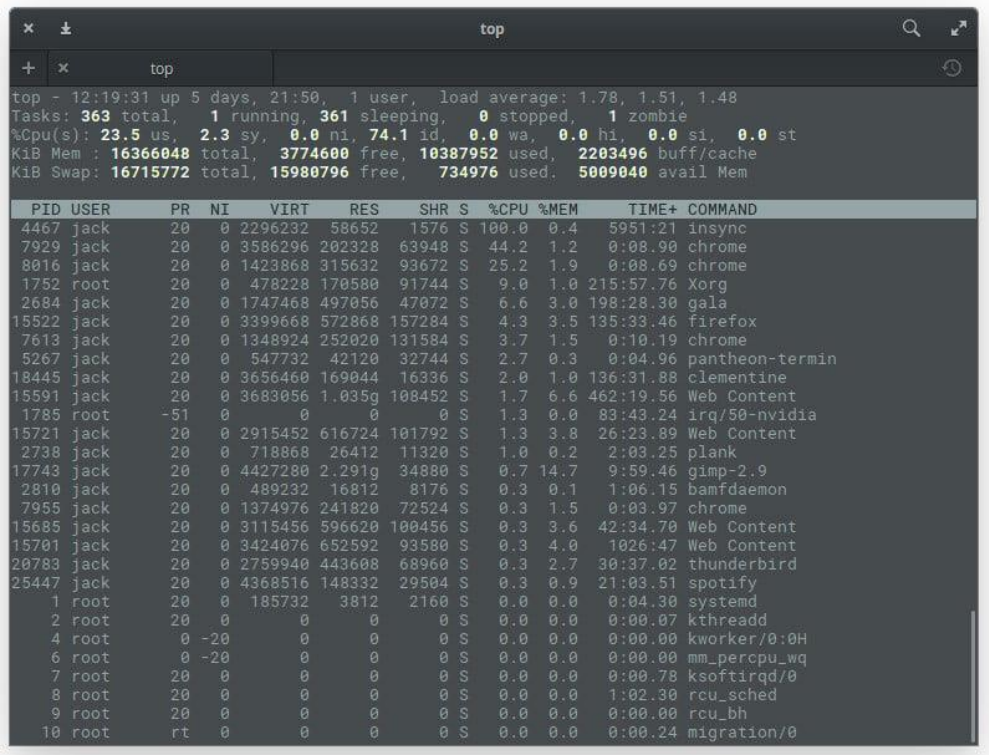
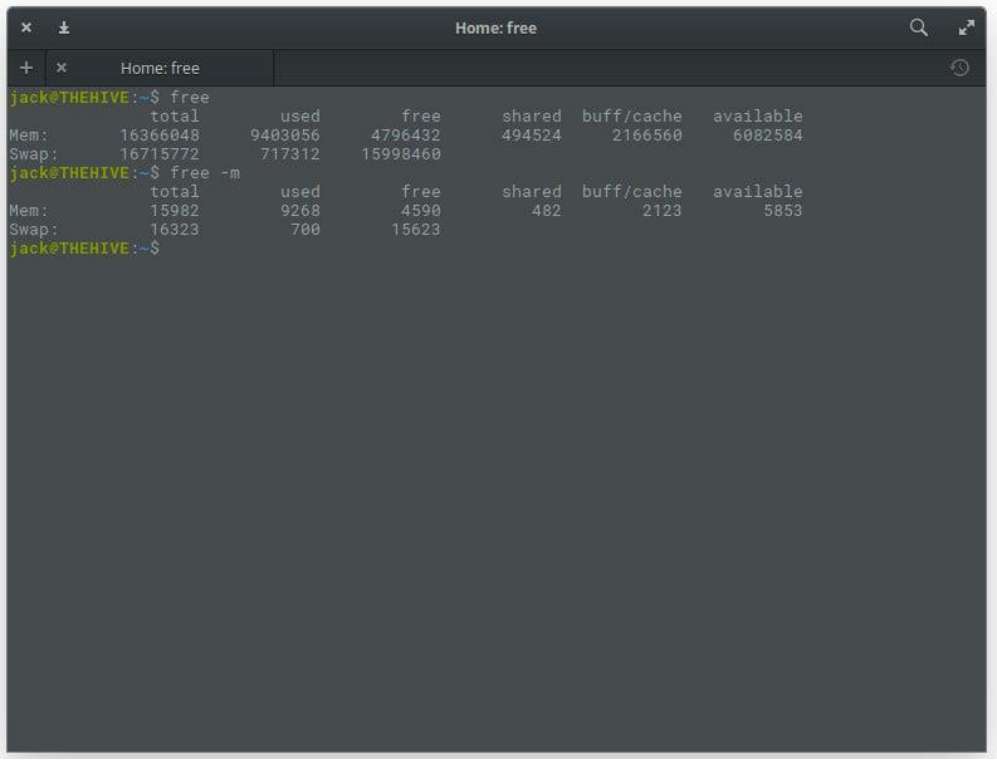
CSOPESY Major Output: Multitasking OS

Prepared by: Gregory Cu
Created By: Neil Patrick Del Gallego, Ph.D.

Updated as of: Oct 6, 2025

[100 pts] General Instructions: The final part is your multi-tasking OS with memory management.





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. Aside from this, you should check the memory debugging tools in Linux CLI to give you an idea of what to do in this final output.
https://www.linuxfoundation.org/blog/blog/classic-sysadmin-linux-101-5-commands-for-checking-memory-usage-in-linux

Checklist of Requirements
Your system must have ALL the following features implemented properly.

Requirement	Main menu console
Description	 Additional commands must be recognized in the main menu: • "process-smi" – provides a summarized view of the available/used memory, as well as the list of processes and memory occupied. This is similar to the "nvidia-smi" command. • "vmstat" – provides a detailed view of the active/inactive processes, available/used memory, and pages.
Requirement	Memory manager
Description	It must support a demand paging allocator. For the demand paging allocator, pages are loaded into physical memory frames on demand. When a process references a virtual memory page that is not currently in a frame, a page fault occurs, and the required page is brought from the backing store into a free frame. If no frames are free, a page replacement algorithm selects a page to be evicted to the backing store.
Requirement	Memory visualization and backing store access
Description	The application has some way to debug the memory, such as "vmstat" and "process-smi." The backing store is represented as a text file that can be accessed at any given time. It is saved in a text file "csopesy-backing-store.txt."
Requirement	Required memory per process
Description	When creating processes via "screen -s" command, a memory size is required. The new "screen -s" command is: `screen -s <process_name> <process_memory_size>:` This part of the screen command creates a new process with a given name and memory allocation. NOTES: • All memory ranges are $[2^6, 2^{16}]$ bytes and the power of 2 format. The console will throw an "invalid memory allocation" message to the user if it's outside of range. • Sample usage: `screen -s process1 256` (allocates 256 bytes to the process) • Processes must require memory of at least 64 bytes to store variables.

Requirement	Simulating memory access via process instruction
Description	In addition to previous process instructions (e.g. PRINT, DECLARE, etc.), there must be a mechanism to simulate memory access: • READ (var, memory_address) – performs a retrieval of a uint16 value from memory and stores it to a variable, var. If the memory block isn't initialized, the uint16 value is 0. • WRITE (memory_address, value) – writes uint16 value to the specified memory address. NOTES: • Variables are tied to a process, stored in memory, and will not be released until the process finishes. • uint16 variables are clamped between (0, max(uint16)) and consume 2 bytes of memory. • uint16 variables are stored in the symbol table segment of the process. • The symbol table segment has a fixed size of 64 bytes. Your program can store a maximum of 32 variables. If the limit is reached, succeeding instructions involving variable declarations will be ignored. • memory_address is a hexadecimal value. Example usage: 1. READ my_var 0x1000 - Read the uint16 value at address 0x1000 and store it in my_var. If 0x1000 is unitialized, returns a 0. 2. WRITE 0x2000 42 – Writes the uint16 value 42 to address 0x2000. 3. READ my_var_2 0x2000 – Reads the uint16 value at address 0x2000. Since this is initialized already, it returns a value of 42. • Attempting to read/write to an invalid memory address (e.g., outside the dedicated memory space) will throw an access violation error and shut down the process, akin to a memory access violation error in user programs. • Memory addresses and representation of memory are emulated. It is not a 1:1 mapping of the physical memory/RAM when running the program. • Read/write memory operations are now included in generating process instructions via "scheduler-start" command.
Requirement	User-defined instructions during process creation
Description	Ability to add a set of user-defined instructions when creating a process. We will introduce the "screen -c" command as follows: `screen -c <process_name> <process_memory_size> "<instructions>":` This part sends a string of 1 – 50 instructions to be executed by the specified process. Instructions are semicolon-separated. Throws "invalid command" if the instruction size is not met. Sample usage: `screen -c process2 "DECLARE varA 10; DECLARE varB 5; ADD varA varA varB; WRITE 0x500 varA; READ varC 0x500; PRINT(\"Result: \" + varC)"` 1. `DECLARE varA 10:` Declares a uint16 variable "varA" and sets it to 10. 2. `DECLARE varB 5:` Declares a uint16 variable "varB" and sets it to 5. 3. `ADD varA varA varB:` Adds varA and varB, storing the result (15) in varA. 4. `WRITE 0x500 varA:` Writes the value of varA (15) to memory address 0x500. 5. `READ varC 0x500:` Reads the uint16 value from memory address 0x500 and stores it in varC. 6. `PRINT("Result: " + varC):` Prints the string "Result: " followed by the value of varC (which should be 15).
Requirement	Previous features from MO1
Description	All implemented features from the MO1, but with additional features, focused on memory management and file system interface.

	To indicate memory access violation errors, the "screen -r" command must be updated: From MO1: 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." Addition for MO2: If the process name has prematurely shut down due to a memory access violation error, the console should print "Process <process name> shut down due to memory access violation error that occurred at <HH:MM:SS>. <Hex memory address> invalid."

The memory manager

Your system is simulating memory in the background. Thus, it would be limited by the maximum amount of main memory allocated by your original OS. Memory spaces are bound within your running program's memory address. Memory spaces are pre-allocated and free to use by any processes upon startup.

The memory space will typically be limited to N bytes, and each process will utilize a fraction of the memory.

Your memory manager must support backing store operations when in low memory, context-switching processes in and out of the backing store (writing/reading in a file).

Memory visualization

There must be a mechanism to visualize and debug memory. The user can use either "process-smi," which provides a high-level overview of available/used memory, or "vmstat," which provides fine-grained memory details.

The "process-smi" is similar to the nvidia-smi command that prints a summarized view of the memory allocation and utilization of the processor (CPU for your program / GPU for nvidia-smi). A sample mockup is provided below:

```
 ____ ____   ___  ____  _____ ______   __
 / ___/ ___| / _ \|  _ \| ____/ ___\ \ / /
| |   \___ \| | | | |_) |  _| \___ \\ V /
| |___ ___) | |_| |  __/| |___ ___) || |
 \____|____/ \___/|_|   |_____|____/ |_|
----------------------------------------
Welcome to CSOPESY Emulator!

Developers:
Del Gallego, Neil Patrick

Last updated: 01-18-2024
root:\> process-smi

------------------------------------------------
| PROCESS-SMI V01.00 Driver Version: 01.00 |
------------------------------------------------
CPU-Util: 100%
Memory Usage: 1245MiB / 4799MiB
Memory Util: 26%

================================================
Running processes and memory usage:
------------------------------------------------
process05 134MiB
process06 134MiB
process07 977MiB
------------------------------------------------
root:\>
```

The "vmstat" command provides a more detailed view. The following information are:

Total memory	Total main memory in bytes.

Used memory	Total active memory used by processes.
Free memory	Total free memory that can still be used by other processes.
Idle cpu ticks	Number of ticks wherein CPU cores remained idle.
Active cpu ticks	Number of ticks wherein CPU cores are actually executing instructions.
Total cpu ticks	Number of ticks that passed for all CPU cores.
Num paged in	Accumulated number of pages paged in.
Num paged out	Accumulated number of pages paged out.

You can follow a similar layout from vmstat:

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
From your MCO1 – OS Scheduler	
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-test" 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.
New parameters for MCO2 – Multitasking OS All memory ranges are [$2^6, 2^{16}$] and the power of 2 format.	
max-overall-mem	Maximum memory available in bytes.
mem-per-frame	The size of memory in bytes per frame. This is also the memory size per page. The total number of frames is equal to max-overall-mem / mem-per-frame.
min-mem-per-proc	Memory required for each process created via the "scheduler_start" command.
max-mem-per-proc	Let P be the number of pages required by a process and M is the rolled value between min-mem-per-proc and max-mem-proc. P can be computed as M/ mem-per-frame.

Scheduler and memory interaction
Consistent with real-world OS, instructions can only be performed when a valid page has been found. Page fault handling continuously occurs until a valid page has been returned, before an instruction is performed.

Example scenario:
```
screen -c faulty_process "DECLARE varA 10; DECLARE varB 5; ADD varA varA varB; WRITE 0x500 varA; READ varC 0x500; PRINT(\"Result: \" + varC)"
```

1. As there are only 3 variables required, this only occupies 2x3 = 6 bytes of memory, well within the 64-byte symbol table segment size.
2. Assume that the physical memory is full and occupied by other running processes.
3. Assume that 0x500 is not in physical memory, then a page fault occurs.
4. The demand pager finds a victim frame to be removed.
5. 0x500 page is brought to a valid frame.
6. Restart the WRITE instruction.
7. Steps 3 – 5 repeat indefinitely until a valid frame is found.

Similarly, variable declaration commands cannot execute if the symbol table segment is not in physical memory. Thus, a page fault also occurs.

Memory allocation and page fault handling only occur when the process is assigned a CPU worker.

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 13	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
  - Process representation with an emphasis on memory representation, such as memory addressing.
  - Scheduler implementation
  - Memory management – demand paging and backing store operation

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.