

## CSOPESY Major Output: Multitasking OS

Created By: Neil Patrick Del Gallego, Ph.D.

By group

Updated as of: June 24, 2025

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

```
x  top
+ x top
top - 12:19:31 up 5 days, 21:50, 1 user, load average: 1.78, 1.51, 1.48
Tasks: 363 total, 1 running, 361 sleeping, 0 stopped, 1 zombie
%Cpu(s): 23.5 us, 2.3 sy, 0.0 ni, 74.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 16366048 total, 3774600 free, 10387952 used, 2203496 buff/cache
KiB Swap: 16715772 total, 15980796 free, 734976 used, 5009040 avail Mem

 PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
4467 jack      20   0 2296232  58652  1576 S 100.0  0.4  5951:21 insync
7929 jack      20   0 3586296 202328 63948 S 44.2  1.2  0:08.90 chrome
8016 jack      20   0 1423868 315632 93672 S 25.2  1.9  0:08.69 chrome
1752 root      20   0 478228 170580 91744 S 9.0  1.0 215:57.76 Xorg
2684 jack      20   0 1747468 497056 47872 S 6.6  3.0 198:28.30 gala
15522 jack     20   0 3399668 572868 157284 S 4.3  3.5 135:33.46 firefox
7613 jack      20   0 1348924 252020 131584 S 3.7  1.5  0:10.19 chrome
5267 jack      20   0 547732 42120 32744 S 2.7  0.3  0:04.96 pantheon-termin
18445 jack     20   0 3656460 169044 16336 S 2.0  1.0 136:31.88 clementine
15591 jack     20   0 3683056 1.035g 108452 S 1.7  6.6 462:19.56 Web Content
1785 root     -51   0     0     0 S 1.3  0.0 83:43.24 irq/50-nvidia
15721 jack     20   0 2915452 616724 101792 S 1.3  3.8 26:23.89 Web Content
2738 jack      20   0 718868 26412 11320 S 1.0  0.2  2:03.25 plank
17743 jack     20   0 4427280 2.291g 34880 S 0.7 14.7  9:59.46 gimp-2.9
2810 jack      20   0 489232 16812 8176 S 0.3  0.1  1:06.15 bamfdaemon
7955 jack      20   0 1374976 241820 72524 S 0.3  1.5  0:03.97 chrome
15685 jack     20   0 3115456 596620 100456 S 0.3  3.6 42:34.70 Web Content
15701 jack     20   0 3424076 652592 93580 S 0.3  4.0 1026:47 Web Content
28783 jack     20   0 2759940 443608 68960 S 0.3  2.7 30:37.02 thunderbird
25447 jack     20   0 4368516 148332 29504 S 0.3  0.9 21:03.51 spotify
 1 root       20   0 185732 3812 2160 S 0.0  0.0  0:04.30 systemd
 2 root       20   0     0     0 S 0.0  0.0  0:00.07 kthreadd
 4 root      -20   0     0     0 S 0.0  0.0  0:00.00 kworker/0:0H
 6 root      -20   0     0     0 S 0.0  0.0  0:00.00 mm_percpu_wq
 7 root       20   0     0     0 S 0.0  0.0  0:00.78 ksoftirqd/0
 8 root       20   0     0     0 S 0.0  0.0  1:02.30 rCU_sched
 9 root       20   0     0     0 S 0.0  0.0  0:00.00 rCU_bh
10 root      rt   0     0     0 S 0.0  0.0  0:00.24 migration/0
```

```
x  Home: free
+ x Home: free
jack@THEHIVE:~$ free
              total        used        free      shared  buff/cache available
Mem:       16366048     9403056   4796432     494524    2166560      6082584
Swap:      16715772      717312   15998460
jack@THEHIVE:~$ free -m
              total        used        free      shared  buff/cache available
Mem:          15982      9268        4590        482      2123        5853
Swap:         16323       700      15623
jack@THEHIVE:~$
```

## 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.

<b>Requirement</b>	Main menu console
<b>Description</b>	<pre> 1   _____\_____ 2   / \ / \ / \ / \ 3                   4   \ \ \ \ \ \ \ \  5                   6   \ \ \ \ \ \ \ \  7   _____\_____ 8                   9   \ \ \ \ \ \ \ \  10                  11  \ \ \ \ \ \ \ \  12                  13  \ \ \ \ \ \ \ \  14                  15  \ \ \ \ \ \ \ \  </pre> <p>Welcome to CSOPESY Emulator!</p> <p>Developers: Del Gallego, Neil Patrick</p> <p>Last updated: 01-18-2024</p> <p>root:\&gt;</p>
	<p>Additional commands must be recognized in the main menu:</p> <ul style="list-style-type: none"> <li>“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.</li> <li>“vmstat” – provides a detailed view of the active/inactive processes, available/used memory, and pages.</li> </ul>
<b>Requirement</b>	Memory manager
<b>Description</b>	<p>It must support a demand paging allocator.</p> <p>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.</p>
<b>Requirement</b>	Memory visualization and backing store access
<b>Description</b>	<p>The application has some way to debug the memory, such as “vmstat” and “process-smi.”</p> <p>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.”</p>
<b>Requirement</b>	Required memory per process
<b>Description</b>	<p>When creating processes via “screen -s” command, a memory size is required. The new “screen -s” command is:</p> <p>screen -s &lt;process_name&gt; &lt;process_memory_size&gt;: This part of the screen command creates a new process with a given name and memory allocation.</p> <p>NOTES:</p> <ul style="list-style-type: none"> <li>All memory ranges are <math>[2^6, 2^{16}]</math> 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.</li> <li>Sample usage: screen -s process1 256 (allocates 256 bytes to the process)</li> <li>Processes must require memory of at least 64 bytes to store variables.</li> </ul>
<b>Requirement</b>	Simulating memory access via process instruction

<b>Description</b>	<p>In addition to previous process instructions (e.g. PRINT, DECLARE, etc.), there must be a mechanism to simulate memory access:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• WRITE(memory_address, value) – writes uint16 value to the specified memory address.</li> </ul> <p><b>NOTES:</b></p> <ul style="list-style-type: none"> <li>• Variables are tied to a process, stored in memory, and will not be released until the process finishes.</li> <li>• uint16 variables are clamped between (0, max(uint16)) and consume 2 bytes of memory.</li> <li>• uint16 variables are stored in the symbol table segment of the process.</li> <li>• 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.</li> <li>• memory_address is a hexadecimal value. Example usage:             <ol style="list-style-type: none"> <li>1. READ my_var 0x1000 - Read the uint16 value at address 0x1000 and store it in my_var. If 0x1000 is uninitialized, returns a 0.</li> <li>2. WRITE 0x2000 42 - Writes the uint16 value 42 to address 0x2000.</li> <li>3. READ my_var_2 0x2000 - Reads the uint16 value at address 0x2000. Since this is initialized already, it returns a value of 42.</li> </ol> </li> <li>• 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.</li> <li>• Memory addresses and representation of memory are emulated. It is not a 1:1 mapping of the physical memory/RAM when running the program.</li> <li>• Read/write memory operations are now included in generating process instructions via “scheduler-start” command.</li> </ul>
<b>Requirement</b>	<p>User-defined instructions during process creation</p> <p><b>Description</b></p> <p>Ability to add a set of user-defined instructions when creating a process. We will introduce the “screen -c” command as follows:</p> <p><code>screen -c &lt;process_name&gt; &lt;process_memory_size&gt; "&lt;instructions&gt;":</code> 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.</p> <p>Sample usage:</p> <pre>screen -c process2 "DECLARE varA 10; DECLARE varB 5; ADD varA varA varB; WRITE 0x500 varA; READ varC 0x500; PRINT(\"Result: \" + varC)"</pre> <ol style="list-style-type: none"> <li>1. DECLARE varA 10: Declares a uint16 variable "varA" and sets it to 10.</li> <li>2. DECLARE varB 5: Declares a uint16 variable "varB" and sets it to 5.</li> <li>3. ADD varA varA varB: Adds varA and varB, storing the result (15) in varA.</li> <li>4. WRITE 0x500 varA: Writes the value of varA (15) to memory address 0x500.</li> <li>5. READ varC 0x500: Reads the uint16 value from memory address 0x500 and stores it in varC.</li> <li>6. PRINT("Result: " + varC): Prints the string "Result: " followed by the value of varC (which should be 15).</li> </ol>
<b>Requirement</b>	<p>Previous features from MO1</p> <p><b>Description</b></p> <p>All implemented features from the MO1, but with additional features, focused on memory management and file system interface.</p> <p>To indicate memory access violation errors, the “screen -r” command must be updated:</p>

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:

```
72
73
74
75
76
77      / \ / \ / \ / \ / \ / \ / \
78      | | | | | | | | | | | |
79      | | | | | | | | | | | |
80      | | | | | | | | | | | |
81      \ \ / \ / \ / \ / \ / \ / \
82
83
84
85
86      Welcome to CSOPESY Emulator!
87
88
89
90
91
92
93      Developers:
94      Del Gallego, Neil Patrick
95
96
97      Last updated: 01-18-2024
98      root:\> process-smi
99
100
101      | PROCESS-SMI V01.00 Driver Version: 01.00 |
102
103      CPU-Util: 100%
104      Memory Usage: 1245MiB / 4799MiB
105      Memory Util: 26%
106
107
108      =====
109      Running processes and memory usage:
110
111
112      process05 134MiB
113      process06 134MiB
114      process07 977MiB
115
116
117      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:

```
jacket@THEHIVE:~$ vmstat -s
 16366040 K total memory
 5522924 K used memory
 6847600 K active memory
 5176984 K inactive memory
 3595752 K free memory
 370116 K buffer memory
 6877248 K swap cache
 16715772 K total swap
   0 K used swap
 16715772 K free swap
 4346370 non-nice user cpu tick
 10222 nice user cpu ticks
 602720 system cpu ticks
 76488300 idle cpu ticks
 74043 IO-wait cpu ticks
   0 IRQ cpu ticks
 7043 softirq cpu ticks
   0 stolen cpu ticks
 5643394 pages paged in
19691626 pages paged out
   0 pages swapped in
   0 pages swapped out
136447937 interrupts
518085297 CPU context switches
1528741508 boot time
 145536 forks
```

### 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
<i>From your MCO1 – OS Scheduler</i>	
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 <code>max-overall-mem / mem-per-frame</code> .
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 $M / \text{mem-per-frame}$ .

### 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  $2 \times 3 = 6$  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 12	Mockup test case and quiz
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:

- o 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.
- o PPT – A technical report of your system containing:
  - o Command recognition
  - o Process representation with an emphasis on memory representation, such as memory addressing.
  - o Scheduler implementation
  - o 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:

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