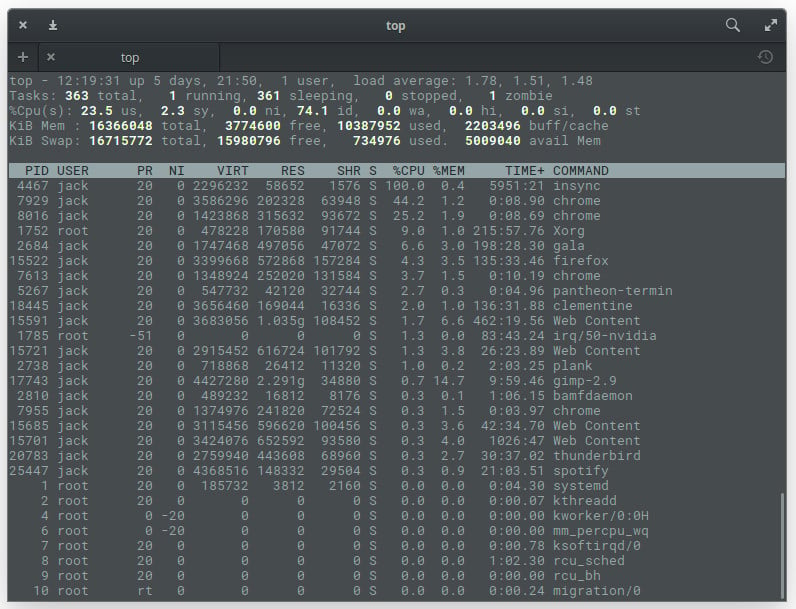
|  |  |
| --- | --- |
| **CSOPESY Major Output: Multitasking OS** | Created By: Neil Patrick Del Gallego, PhD |

By group

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



A screenshot of a computer

Description automatically generated

**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 both a flat memory allocator and a paging allocator.  For a flat memory allocator, when no free memory is available (or fragmented), the oldest process is removed from the main memory and put into the backing store.  For the paging allocator, pages are loaded into memory when free frames are available. All pages must be loaded for the process to execute. The backing store also triggers when there’s a need to remove pages. The oldest process is first removed. |
| **Requirement** | Memory visualization |
| **Description** | The application has some way to debug the memory, such as “vmstat” and “process-smi.” |
| **Requirement** | Previous features from MO1 |
| **Description** | All implemented features from the MO1, but with additional features, focused on memory management and file system interface. |

**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.

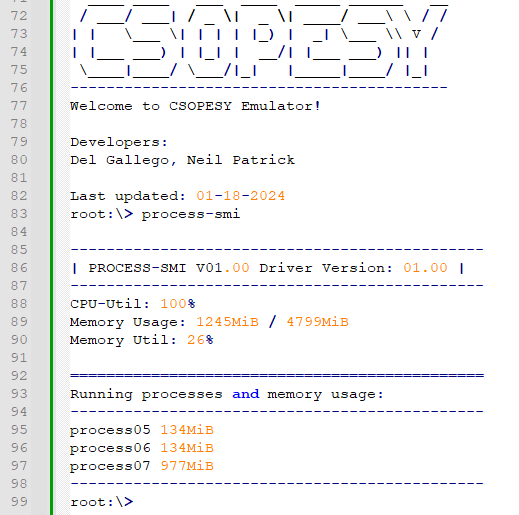
The memory space will typically be limited to N kilobytes, and each process will utilize a fraction of the main memory.

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:



The “vmstat” command provides a more detailed view. This is more useful for the paging allocator. The following information are:

|  |  |
| --- | --- |
| Total memory | Total main memory in KB. |
| 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:

A computer screen shot of a program

Description automatically generated

**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, . |
| 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. |
| **New parameters for MCO2 – Multitasking OS**  *All memory ranges are* and power of 2 format. | |
| max-overall-mem | Maximum memory available in KB. |
| mem-per-frame | The size of memory in KB per frame. This is also the memory size per page.  The total number of frames is equal to max-overall-mem / mem-per-frame.  If max-overall-mem = mem-per-frame, then the emulator will use a flat memory allocator. |
| min-mem-per-proc | Memory required for each process  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. |
| max-mem-per-proc |

**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:

* + 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
  + Scheduler implementation
  + Memory management – flat memory and paging

**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. |