CMSC 312

Operating System Simulator Project

Paul Hudgins, Emily Klein, Quark Wei

V70270484, V00374656, V00687866

November 29, 2016

1 Usage

The command line interface will auto-suggest while you type. Suggestions can be completed by pressing the Tab key. The previous command can be repeated by pressing the Enter key. All previous commands can be accessed with the Up arrow.

The *Executable* directory contains the simulator as an executable jar, and the *Program Files* subdirectory contains program and job files for testing. All job and program files must be placed in the *Program Files* directory to be used, and must have the file extensions *.prgrm* and *.job*, respectively.

The *LOAD* command is used to load *jobs* and *programs*, allowing for multiple inputs/parameters at a time, but is also used to list all loadable files if no parameters are passed in.

The command-line interface implements most features of cat, which allows for redirection into an output file. This means you can create and view the contents of program and job files on the fly!

1.1 Program Files

- First item in a list
- Second item in a list

1.2 Job Files

2 Simulation Architecture

The simulator consists of three main packages. The *simulator* package simulates hardware operation, including CPU execution and IO. The *kernel* package contains the operating system which controls the simulated hardware. The *user_interface* package contains the shell and GUI.

2.1 Execution

Programs are read as a text file and then assembled as an array of operations. Because the CALCULATE operation consumes a variable number of cycles, the CPU uses an Operation Counter as well as a Program Counter. The Program Counter, as usual, indicates which operation is currently being executed, and the Operation Counter indicates how many cycles are remaining for that operation. Because kernel methods are executed on the JVM and not on the simulated processor, kernel methods do not consume CPU cycles. All operations on the simulated CPU can be considered to execute in "user mode" and all methods from the *kernel* package can be considered to execute in "kernel mode". The only way to go from user mode to kernel mode is to signal an interrupt.

2.2 Interrupts

The hardware will blindly continue execution of the current process in user mode until an interrupt flag is set in the interrupt processor. The interrupt processor then routes the interrupt to the appropriate handler. Most interrupt handlers will make use of a common context-switch handler which copies CPU registers to the PCB for the current process and copies saved register states from the next PCB to the CPU. All interrupts are preemptive. The interrupt processor supports two system-driven interrupts and four traps:

- YIELD: Triggered by expiration of the burst timer set by the short term scheduler
- IO_COMPLETE: Signals that an IO event needs to be handled
- TERMINATE: Terminates the current process
- ACQUIRE: Requests access to a resource, blocking if the resource is not available, so that busy waiting is avoided
- RELEASE: Releases a resource
- WAIT_FOR_IO: Blocks until an IO_COMPLETE signal is received

3 Scheduling

The system uses a short-term scheduler and a long-term scheduler. The short-term scheduler executes at the end of every CPU burst. The long-term scheduler executes after 20 calls to the short-term scheduler, or if the short-term scheduler is unable to select a process for execution.

3.1 Short-term Scheduler

The short-term scheduler schedules CPU bursts and IO for processes that are in memory. It utilizes the following queues:

- Ready Queue: There is one priority queue of processes waiting for CPU time.
- Device Queues: There is one queue of processes waiting for accesses for each IO device. Currently, the simulation only uses one IO device, but it can accommodate more. When one process releases a device, the short-term scheduler immediately attempts to execute another process waiting for that device in order to maximize IO utilization.

- Waiting Queue: There is one queue of processes waiting for a signal. Because IO response is the only signal in the simulator, there is no Event Queue. The running process will be preempted and the waiting process will be executed as soon as a signal is received.

3.2 Long-term Scheduler

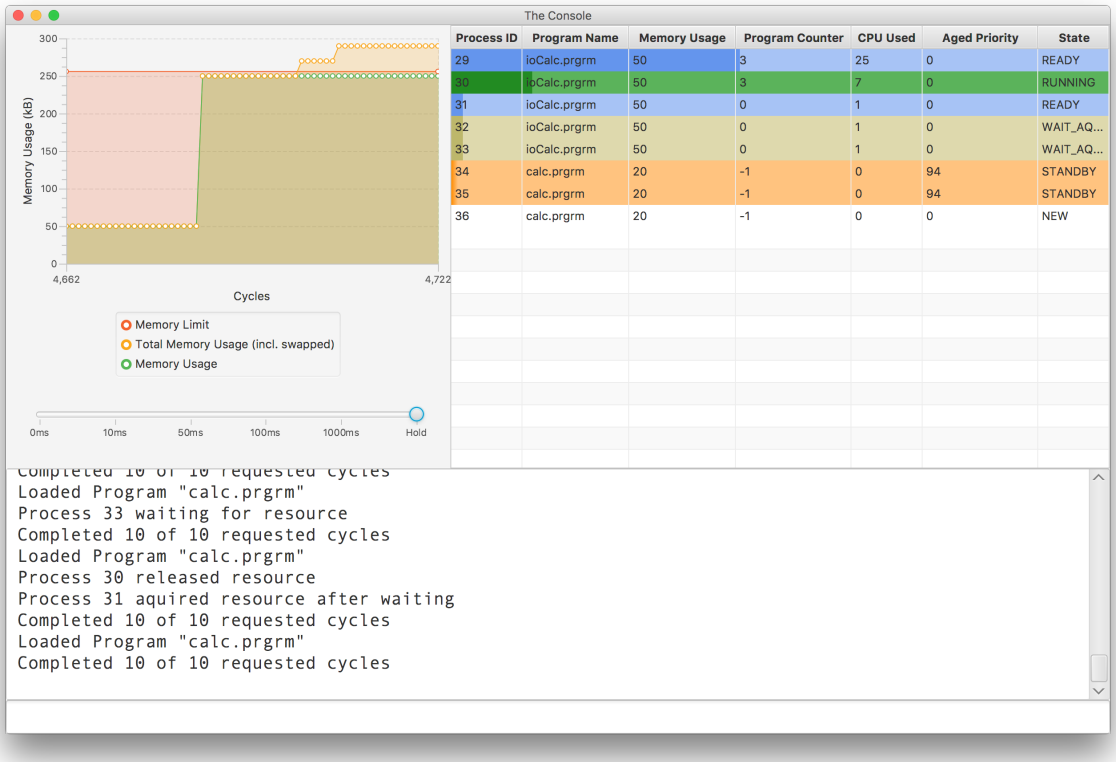
The long-term scheduler swaps processes in and out of memory, ensuring that the memory limit is not exceeded. Each time the long-term scheduler executes, it does the following:

- Pulls all new processes off of the New Process Queue
- If space is available in memory, processes from the Standby Queue are swapped into memory until there is no more space available.
- If processes remain in Standby Queue, some processes are swapped out of memory to make room for the processes that are standing by.

3.3 Priority and Aging

Each program can be assigned a priority in the second line of the program file, with the command PRIORITY and an integer argument. Scheduling queues are implemented as priority heaps. The effective priority of a process is its base priority plus a constant fraction of its age. Age is defined as the time since the end of the last CPU burst.

4 GUI



4.1 Overview

The GUI features a live memory usage graph, process viewer, simulation speed control slider, and a terminal emulator for the simulated operating system.

4.2 Memory Graph

The memory graph shows the system memory limit, the amount of memory currently being used, and the amount of total memory (including swapped out memory that isn't being used, but is ready to be swapped in by processes).

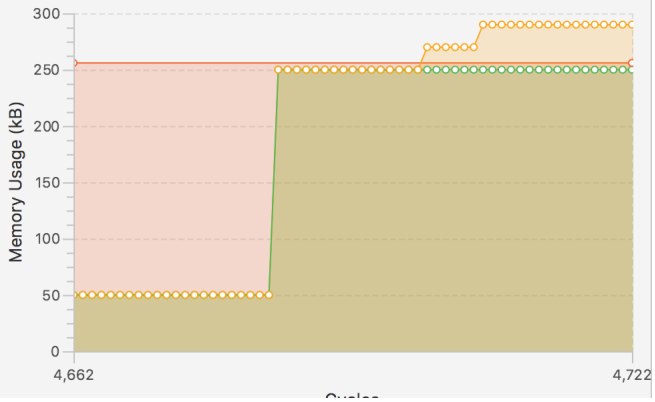


Figure 4.1: The memory graph showing that some processes are swapped out

4.3 Delay Slider

The slider changes the amount of delay between each CPU cycle. Moving this all the way to the right will temporarily pause execution. If the simulator is in the middle of an execution block, the block must be completely run before any other commands must be input. Thus, the slider may not be used to inject commands.

The slider can be used as a "step" button by quickly sliding from *hold*, to *1000ms*, and back to *hold*.

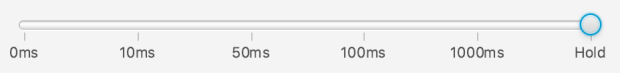


Figure 4.2: The delay slider in the 'Hold' position

4.4 Process Table

The process table shows a complete list of processes, and is sortable by any column. Each row represents a process and is colored based on process state. The background color for each row also serves as a progress bar to hint at when the process will be finished executing.

4.5 Terminal Emulator

The terminal emulator supports the following commands: "PROC", "MEM", "LOAD", "EXE", "CAT", "RESET", and "EXIT".

Auto-suggest supports all commands, including parameters (file names) for LOAD and CAT. Auto-suggest even supports multiple parameters, and will not suggest file names that have already been typed.

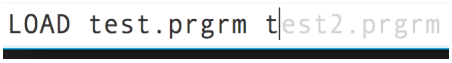


Figure 4.3: *test.prgrm* comes before *test2.prgrm* in lexicographic order, but is not suggested because it is already used in a previous parameter

The terminal emulator supports the Enter key to repeat the last command, the Tab key to auto-complete suggestions, and the up and down arrow keys to browse command history.

If an typo is entered, the terminal emulator will try and figure out your desired command, and display it for you to type correctly.

```
>XE
ERROR: Command "XE" not found
Did you mean: "EXE" or "EXIT"
>OALDE
ERROR: Command "OALDE" not found
Did you mean: "LOAD"
```

Figure 4.4: *EX* is similar to commands *EXE* and *EXIT*, so both are suggested

- **PROC** is used to display info on all processes in plain-text. This is useful if you want to copy and paste process info, but is otherwise redundant (because of the process table)
- **MEM** is used to display the amount of memory currently being used, as well as the amount of memory that is swapped out.
- **LOAD**
  - If no parameters are passed in, a list of loadable programs and jobs will be shown.
  - One or more file names can be passed into be loaded by the simulator, in order of appearance. Job files are run on read, and are functionally equivalent to typing the contents of the job file into the terminal emulator.
- **EXE** will run any program files that have been loaded. An integer can be passed in as a parameter to limit execution to a certain number of cycles. Otherwise, all loaded programs are run until termination.
- **CAT** behaves very similarly to its bash counterpart. If run without any parameters, some help text will be output to the terminal.
- **RESET** resets the simulator.
- **EXIT** exits the program.