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-complete arguments to the LOAD command. 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. The all job and program files must be placed in the *Program Files* directory to be used.

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.