CMSC 312

Operating System Simulator Project

Paul Hudgins, Emily Klein, Quark Wei

V70270484, V00374656, V00687866

November 29, 2016

1 Usage

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 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 *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 shortTermSchedulerTest.job

The following features can be observed when running shortTermSchedulerTest.job.

- The most CPU time is given to a High_Priority_CPU_Bound_Process.

- IO utilization is indicated by a process being in the pink WAIT_FOR_IO state. Utilization is higher than would be expected based on priorities and aging alone. This is because a waiting process is given preference for the burst after the IO device is released. Frequently, a process can be observed in the WAIT_FOR_IO state with an aged priority of 1 or 0, which is lower than the high-priority process' base priority of 2.
- The Low_Priority_Background process receives the least CPU time, but does not undergo starvation.

1.2 LONGTERMSCHEDULERTEST.JOB

The following features can be observed when running longTermSchedulerTest.job.

- Memory usage never exceeds 256kb. Processes that are in the Orange STANDBY or White NEW states consume memory on the backing store, but not in main memory.
- The long-term scheduler is executed at least once for every 20 cycles of the short term scheduler. Execution of the long-term-scheduler can be observed when there is a change in which processes are in the Orange STANDBY state.
- Processes that have not yet been executed are given an age based on the start-time of the system. Ensures that new processes have the highest priority, and are swapped into memory first.

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 COMPILER AND CRITICAL SECTIONS

The Compiler in the *utilities* package reads program files and compiles them into an array of Operation objects, which are machine code instructions for our simulator. In addition to creating the appropriate parameters for CALCULATE operations, the Compiler will also insert an ACQUIRE operation before and a RELEASE operation after each IO operation, creating a critical section in the code. When a process is executing in its critical section, no other process will be able to acquire the IO device in use.

2.2 Execution

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.

2.3 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 processes that are standing by.

3.3 Scheduling Algorithm

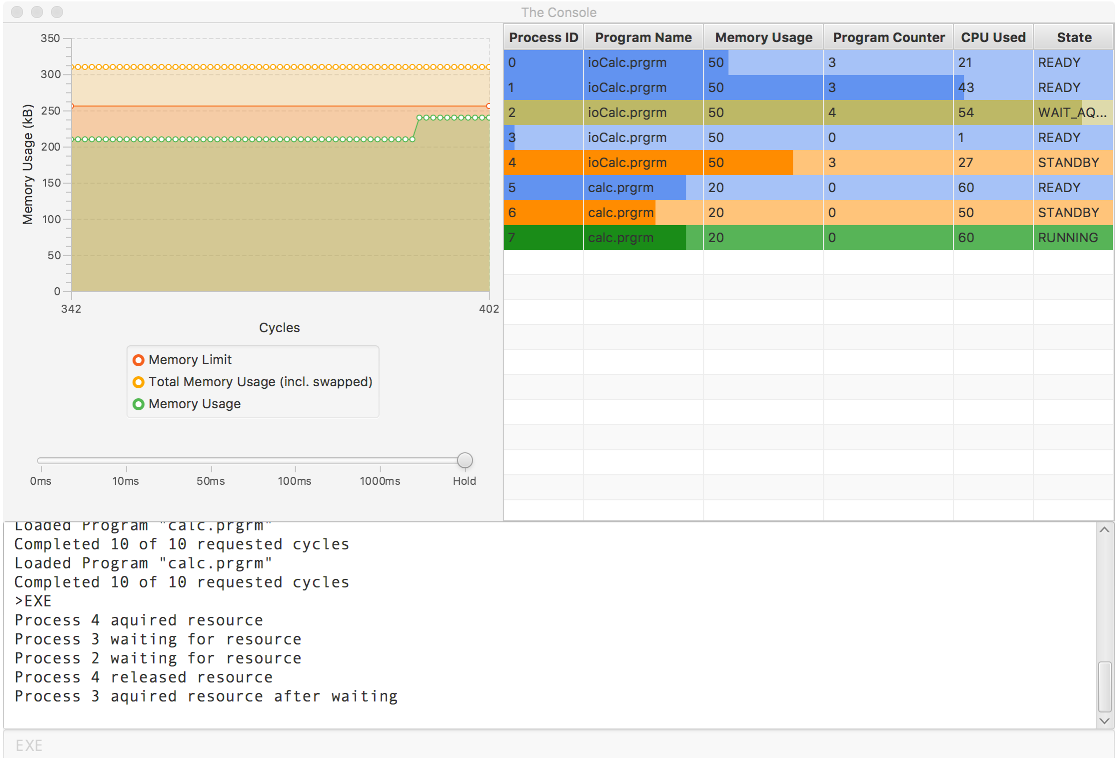
Both the long-term scheduler and short-term scheduler use highest-priority first, with aging. Each program can be assigned a priority in the second line of the program file, with the command PRIORITY and an integer argument. Scheduling queues are implemented as priority heaps. The effective priority of a process is its base priority plus one fiftieth of its age. Age is defined as the time since the end of the last CPU burst.

To maximize IO utilization, the RELEASE handler will immediately schedule the next process waiting for the device, even if there are higher priority processes in the waiting queue. This gives the IO-bound process one “free” CPU burst, so that it can attempt to enter a state where it is waiting for a response signal from the IO device.

4 GUI

4.1 Overview

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


The Console
Memory Usage (kB)
350
300
250
200
150
100
50
0
342
402
Cycles
Memory Limit
Total Memory Usage (incl. swapped)
Memory Usage
0ms
10ms
50ms
100ms
1000ms
Hold
Process ID
Program Name
Memory Usage
Program Counter
CPU Used
State
0 ioCalc.prgrm 50 3 21 READY
1 ioCalc.prgrm 50 3 43 READY
2 ioCalc.prgrm 50 4 54 WAIT_AQ...
3 ioCalc.prgrm 50 0 1 READY
4 ioCalc.prgrm 50 3 27 STANDBY
5 calc.prgrm 20 0 60 READY
6 calc.prgrm 20 0 50 STANDBY
7 calc.prgrm 20 0 60 RUNNING
Completed 10 of 10 requested cycles
Loaded Program "calc.prgrm"
Completed 10 of 10 requested cycles
>EXE
Process 4 aquired resource
Process 3 waiting for resource
Process 2 waiting for resource
Process 4 released resource
Process 3 aquired resource after waiting
EXE