Operating Systems Simulation Applet

Dr. Ronald Marsh
Computer Science Department
University of North Dakota
Grand Forks, ND 58201
rmarsh@cs.und.edu

Joseph Kerian
Computer Science Department
University of North Dakota
Grand Forks, ND 58201

Cory Hanson
Computer Science Department
University of North Dakota
Grand Forks, ND 58201

Abstract

This paper discusses a Java-based operating system simulator developed by students in the Computer Science Department at the University of North Dakota. The goal was to develop a Java applet that provides a graphical simulation of select operating system operations including: CPU scheduling of multiple processes, memory management in the form of page-fault simulation, and process life-cycle explication. The applet is platform independent and compatible with any Java enabled browser.

Introduction

An operating system is a program that manages the computer hardware, provides a basis for application programs, and acts as an intermediary between a user of a computer and the computer hardware. An operating system can be viewed in either of two ways: user and system. The user view is what the user can see and do with the computer. The system view considers the operating system as a resource allocator. A computer system has many resources (hardware and software) that maybe required to solve a problem: CPU time, memory space, file-storage space, I/O devices and so on. The operating system acts as the manager of these resources. Facing many conflicting requests for resources, the operating system must decide how to allocate them to specific programs.

The goal of this project was to develop a Java applet that provided a graphical simulation/demonstration of the operation of a hypothetical operating system (system view). Since the simulation was designed to be used as a teaching tool for future Csci-451 (Operating System I) classes, it designed to be run in most common browsers and was designed with a simple user interface. Figure 1 depicts the architectural design.

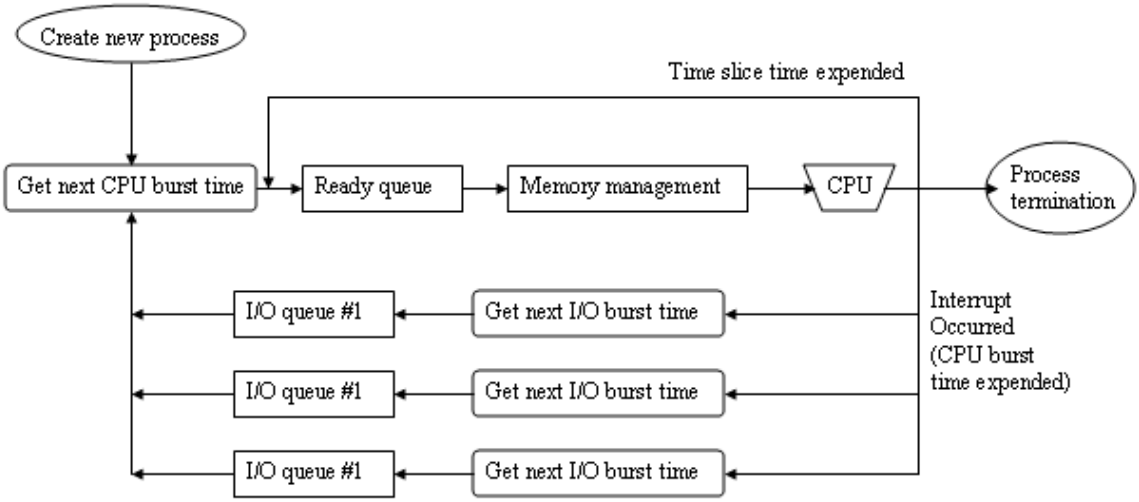

Figure 1. Simulation architecture.

Simulation Details

A process is a program in execution. However, a program by itself is not a process; a program is a passive entity, while a process is an active entity. Processes need resources in order to function, including CPU time, memory, data, and access to I/O devices. The operating system has responsibilities of creating and deleting both system and user processes. It also needs to suspend and resume process. Finally, a process can be in one of five states: new, running, waiting, ready, and terminating. As shown in Figure 1, when a process is newly created it is assigned a randomly generated CPU burst time and a randomly generated amount of memory (10 pages maximum), goes into the ready queue, and waits to be executed by the CPU.

The time a process must sit in the ready queue is dependant on the CPU scheduling algorithms and while there are many different CPU scheduling algorithms, this simulation uses the round robin scheduler. A round-robin scheduling algorithm attempts to share the CPU fairly among all process of the same priority currently waiting in the ready queue. Without the proper scheduling algorithm, a single task can usurp the processor by never blocking, thus never giving other equal priority tasks a chance to run. Round-robin scheduling achieves fair allocation of the CPU to tasks of the same priority by an approach known as time slicing. Each task executes for a defined interval or time slice; then another task executes for an equal interval, in rotation. The allocation is fair in that no task of a priority group gets a second slice of time before the other tasks of a group are given a slice. Finally, the scheduling algorithm must be preemptive in the sense that if an interrupt (hardware or software) occurs the process must be suspended while the interrupt handling routine executes. After which the process may be resumed. Since this is a simulation, we simulated the occurrence of interrupts by the CPU burst time that is randomly generated each time a process moves into the ready queue. A process will get the CPU for either the length of the time slice or the length of the CPU burst time, whichever is less. If the CPU burst time is less, the process is assumed to require I/O and moves into one of the I/O queues. If the time slice is less, the CPU burst time is reduced by the time slice time and the process returns to the ready queue.

The operating system is also responsible for keeping track of which parts of memory are currently being used and by whom. It also decides which processes are to be loaded, allocates, and deallocates memory space. To improve both the utilization of the CPU and the speed of the computers response to its users, we must keep several processes in memory; that is we must share memory.

In a modern computers memory management is done using a method called paged memory. A page, in virtual memory, is a block of memory with its own virtual address. Paging increases the memory available to a program by temporarily transferring (swapping) less-needed parts of the program's working memory from RAM into secondary storage (hard drive). When a page is needed, it is read back into RAM, possibly replacing an existing page, which is moved into the secondary storage. This “juggling” of memory is accomplished by a memory management unit (MMU) and a page replacement algorithm. By using virtual memory, a computer can run tasks that exceed the limits of its physical memory.

While there are a several page replacement algorithms, the algorithm used in the simulation is the Least Recently Used (LRU) algorithm. As a process in the simulation moves from the ready queue into the CPU it first passes through the memory management unit. The memory management routine randomly generates a set of pages that will be required by the process during its current execution period and executes the LRU page replacement algorithm. With LRU, if there are no empty pages, the page that not has been used for the longest time is selected for replacement. As implemented by the simulation, the LRU algorithm:

- Inspects the simulated physical memory – are the required pages already in physical RAM?

- If they are, allow the process to execute.
- If they are not, generate a “page fault” and then:
  - Read the required page from the simulated swap space (hard drive).
  - If there is a free frame in physical memory, use it for the page read.
  - If there is no free frame in physical memory, select the page not used in the longest time to be the victim frame.
  - Write the contents of the victim page to the simulated disk and change the page and frame tables to reflect the memory changes.
  - Insert the page read in to the new frame.
  - Start the user process.

To add some form of realism to the simulation there are 2 types of queues that are modeled: the ready queue and 3 different I/O queues. Each I/O queue is meant to represent a different I/O device. By different, we mean devices that have vastly different “service” times. For example, a computer mouse has a very short service time, while a disk drive has a much longer service time. Thus, when a process is interrupted (CPU burst time expended) the simulation randomly decides what type of I/O is required (Main HD, Printer, or NIC). The process is then sent to the appropriate I/O queue. As Figure 1 depicts, each I/O queue has an associated I/O burst time determination. The I/O burst time determine routine randomly generates (using different distributions) an I/O burst time (I/O service time) which each instance of I/O requires. Once the process has waited the appropriate amount of time it returns to the ready queue and assigned a new CPU burst time.

Simulation GUI

The applet window has been set at 640 x 450 allowing an applet that is viewable even on small (640x480) screen sizes. The interface allows the user to stop, start, step through the simulation, to control the execution speed, and to inspect various other factors, such as the memory allotment/assignment, and process parameters. Finally, the decision was made to use swing components for this applet, as opposed to the now outdated awt components. As a result the main applet extends Japplet, rather than the basic class Applet.

The applet window is split into three panels, in clockwise order starting in the upper left: the display panel, the control panel, and the message panel. The display panel is divided into several subpanels that provide a graphical view of the simulation during execution. The display panel will be described in more detail later. The control panel contains buttons for the basic applet controls (start, stop, step, new process creation, simulation execution speed, and number of steps through the simulation - simulation time). The message panel contains a scrollable text area and is used to provide simulation status messages to the user during applet execution.

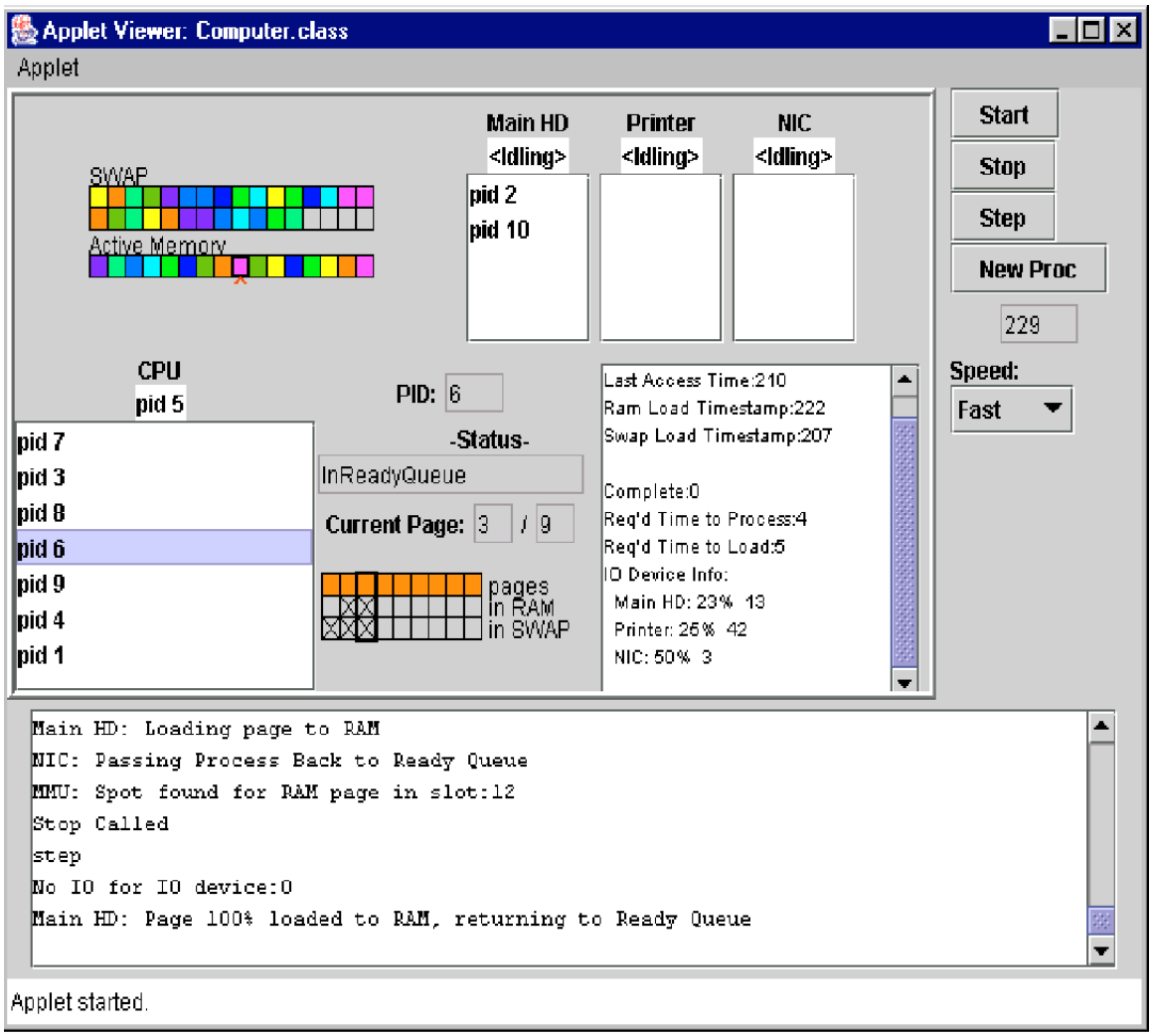

Figure 2. Simulation interface.

While the control and the message panels provide control and information to the user, the display panel provides the majority of the information. In the upper left the display panel provides a display of the memory (swap space and physical RAM) usage. Note that the simulation assumes there are 16 pages of physical RAM and 32 pages of swap space, which is the typical configuration for a Linux system. As processes are created they are assigned a unique color and as processes consume memory the pages assigned are given the appropriate color. Thus, the user can easily monitor the memory consumption.

In the upper right are 3 scrollable text boxes that represent the 3 I/O queues. As processes require I/O they are moved into the appropriate I/O queue. This is represented by the applicable text box displaying the process ID number (PID) of the process. Note in Figure 1 that under each queue title is the text "<Idling>." This text indicates that no process is currently retrieving data from the applicable I/O device. When a process is actively retrieving data from the applicable I/O device, the process's ID (PID) is displayed instead of the text "<Idling>."

In the lower left is a scrollable text box that represents the ready queue. Again the text "<Idling>" indicates that no process is currently executing. When a process is in execution (as is the case in Figure 1), the process's ID (PID) is displayed instead of the text "<Idling>."

The remainder of this panel is used to provide detailed information regarding a process to the user:

- The PID box displays the process ID of the selected process.
- The status box displays the current status of the selected process.
- The current page boxes display the number of pages currently in RAM verses the total number of pages allotted to/required by the process.
- The 30 cells below the current page box depicts the current state of the process's memory pages – whether they are in RAM, in swap, or if they have even been assigned yet.
- The scrollable text box in the lower right provides detailed information regarding a process's history, including:
    1. Ram and Swap Load Timestamps for future use
    2. Amount of this page that the CPU has executed.
    3. Total time to Load this Page from Disk and time to execute by CPU
    4. Simulated IO information
    5. Percent Probability of IO for this page
    6. Delay (in CPU cycles) until IO is complete

All the user needs to do to display the above information is click on any process in any queue. The user can also click on any memory page to display the applicable process's information.

Conclusion

The Java-applet based operating system simulation that was developed by students in the Computer Science Department at the University of North Dakota provides insight into the inner workings of the basic components of an operating system. The simulator allows a student to watch the operations of a typical operating system and investigate certain parameters in detail. While the current simulator meets all of the original goals, we expect to expand the functionality of simulator further.

The current simulator can be executed from the following URL:
http://people.aero.und.edu/~rmarsh/CLASS/CS451/SIMULATION/Computer.htm

References

1. Java, http://java.sun.com/getjava/download.html
2. SIMOS, The Complete Machine Simulator, http://simos.stanford.edu/
3. The Flux Oskit, http://www.cs.utah.edu/flux/oskit/
4. Topsy - A Teachable Operating System, http://www.tik.ee.ethz.ch/~topsy/
5. Nachos, http://www.cs.washington.edu/homes/tom/nachos/