Empirical Exploration In Undergraduate Operating Systems

Steven Robbins
Division of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249-0667
210-458-5544
srobbins@utsa.edu

Kay A. Robbins
Division of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249-0667
210-458-5543
krobbins@utsa.edu

1 ABSTRACT

The undergraduate operating systems course can provide students with a valuable introduction to empirical testing and experimentation. We have developed a process scheduling simulator designed to develop student empirical skills while they are learning part of the standard operating systems curriculum. The simulator is written in Java and available for direct experimentation via the World Wide Web. By accessing the remote URL through an appletviewer, students can permanently save input test data and simulator results generated in HTML format. In one type of assignment, students are given a hypothesis about process scheduling and are asked to develop experiments to support or disprove the hypothesis. In a second type of assignment students are asked to develop their own hypotheses. Not only did these assignments enhance student understanding of process scheduling, but the techniques exposed students to empirical approaches to validation and testing.

1.1 Keywords

operating systems, process scheduling, education, undergraduate curriculum

2 INTRODUCTION

Experimentation is the centerpiece of the traditional scientific method. Experimental exploration can provide new insights, eliminate unproductive approaches and validate theories and methods. Walter Tichy [4] cites several examples in the systems software area where commonly-held assumptions were shown to be false by careful empirical studies. Computer science, as a discipline, has a notoriously poor record in the area of empirical validation. Several studies of computer science publications [5, 6] have shown that the percentage of papers providing no substantiation for claims that needed experimental verification was much higher than in other disciplines in either the hard or soft sciences.

In some respects the weak computer science tradition in experimentation is not surprising given the discipline's rapid emergence and constant pressure for change. There is also little evidence for movement towards a more empirically grounded approach. While undergraduate computer science majors may take a laboratory science as part of their general education requirement, few computer science programs provide students with empirical experience in their discipline.

This paper describes empirical techniques and support tools that we have developed to introduce students to empirical exploration in the undergraduate operating systems course. The particular example presented here is the unit on process scheduling, but the work is part of a larger curriculum development effort supported by the National Science Foundation [7].

The typical presentation of process scheduling includes a description of idealized algorithms such as shortest job first (SJF), first-come, first-served (FCFS) and preemptive priority scheduling. Gantt charts are used to visualize differences in algorithms for short examples. The unit, which typically takes about a week, ends with a discussion of multi-level feedback queues as the method used in practice by most current commercial operating systems. All of the standard textbooks [1, 2, 3] provide exercises on process scheduling, but those that compare the various algorithms consider one cpu burst of at most 5 processes. The student is left with the impression that this is sufficient for evaluation of these algorithms.

The process scheduling simulator allows students to explore process scheduling in an empirical setting. A primary goal is to help the students develop a better understanding of process scheduling including the working of the algorithms and the system parameters that influence their performance. A second goal is expose students to empirical methods in a realistic computer science setting.

Students are introduced to the simulator and then given two types of assignments. In one type of assignment the students are presented with a specific hypothesis about process scheduling and are asked to devise and perform experiments to support or disprove the hypothesis. In the second type of assignment, students are asked to develop and test their own hypotheses about process scheduling. The simulator is designed to make the specification of a series of experiments convenient. An automatic logging facility outputs tables, graphs and comments in HTML format so that the students can easily keep track of their experiments and produce web-based reports of results.

The next section of the paper provides an overview of the simulator. Section 4 presents a sample assignment, and Section 5 describes our experience with using the simulator and hypothesis-based assignments in an undergraduate operating systems class. Section 6 talks about the larger project and invites participation in this project by others.

3 SIMULATOR OVERVIEW

The process scheduling simulator provides a web-based testbed for experimentation with process scheduling algorithms. The simulator interface shown in Figure 1 makes it easy to run experiments on collections of processes with different scheduling parameters and to compare such statistics as throughput and waiting time. Information about the experiment including the specification of the processes and the statistics and graphs resulting from the experiment is stored in a log file in HTML format suitable for viewing from a browser.

The main simulator window shown in Figure 1 has several distinct display areas. The subwindow

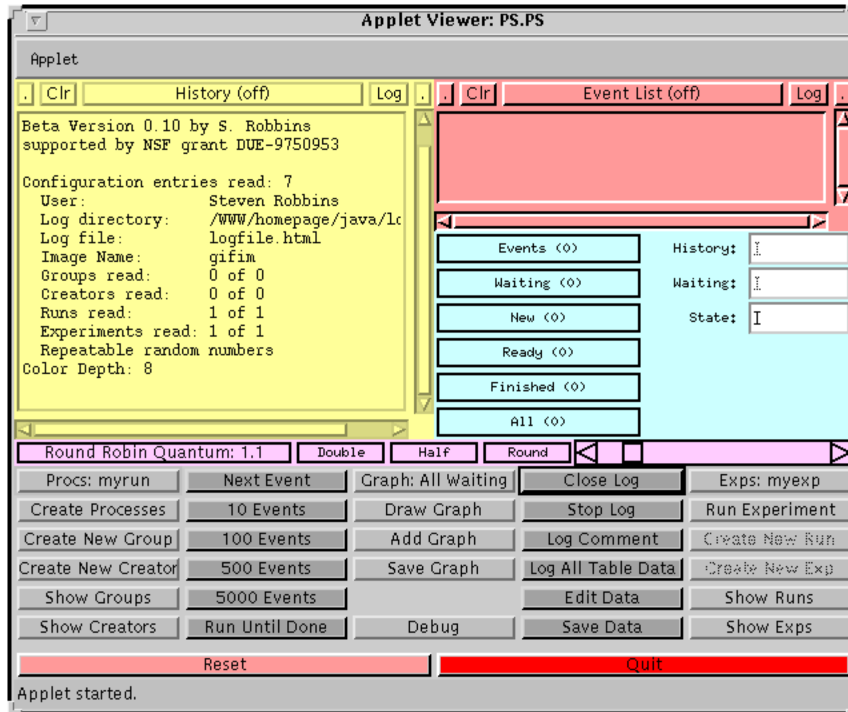

Figure 1: A view of the main simulator window.

in the upper left corner labeled **History** shows the initial configuration read in from a configuration file when the simulator starts up. The configuration file specifies the user's name (which will appear in the log file) as well as information about where to store the log file and which experiments are to be loaded into the simulator. The upper left subwindow can optionally display a complete log of the simulator. The subwindow in the upper right can be used to log all simulation events. The various buttons in the upper right portion of the window allow detailed information about a run to be displayed or logged. For example, you can display the entire history of any process including each time it entered or left a queue. This type of tracing can be useful in determining why an experiment turned out as it did. The contents of either window can be downloaded into the log file in HTML format.

There are five columns of buttons at the bottom of the main window. The rightmost (fifth) column of buttons is for selecting and running experiments. Pushing the top button in this column advances through the available experiments. Pushing the second button, **Run Experiment**, runs the chosen experiment until completion.

The fourth column of buttons controls the log file. The top button opens the log file. When the log file is opened, the button is changed to a **Close Log** button as shown in the diagram above. As runs are made they are logged in the log file and statistics about the run are saved. Pushing the **Log All Table Data** button puts two tables of statistics in the log file. The tables contain entries for all runs

and include statistics on CPU utilization, throughput, turnaround time and waiting time.

The third column of buttons controls graphs. Graphs of the data produced by the simulator can be displayed or inserted in the log file. Some of the available graphs include number of processes completed and average waiting time. Figure 2 shows a graph of average waiting time for three runs. The waiting time is measured by the ratio of total CPU time divided by the total of the CPU time and the time spent in the ready queue. Smaller numbers correspond to longer waits. This measure of waiting time was chosen because it can be used to compare processes with a large variance in duration.

The second column of buttons allows finer control of the running of the simulation, while the first column controls a lower level interface that allows for a more complicated mix of processes.

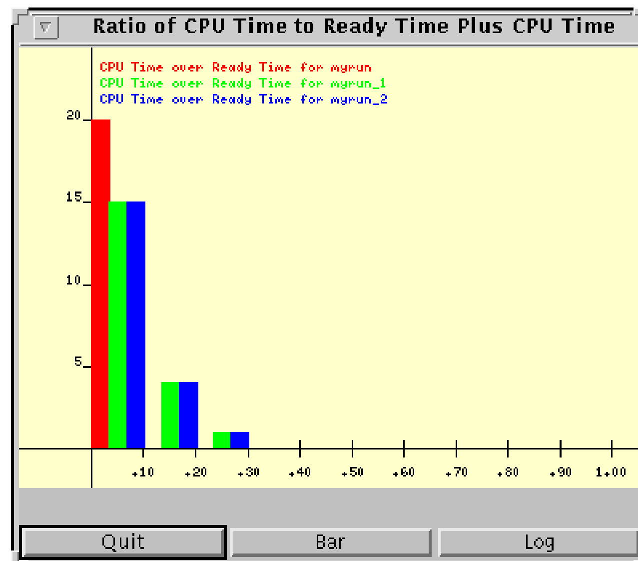

Figure 2: A graph generated by the simulator.

3.1 Specifying an Experiment

An *experimental run* consists of a scheduling algorithm and a collection of processes to be run under that algorithm. An *experiment* consists of a number of experimental runs that are to be compared and analyzed. The simulator organizes the input information in order to make it simple to do experiments in which one or more parameters vary. After the experiment has completed, the simulator can produce tables or graphs of various statistics such as average waiting time and throughput.

To perform an experiment, students must create two files. One file (the run file) contains information about the parameters for one of the runs. The other file (the experiment file) contains a list of runs to be made and the parameters that vary between runs.

Data for the simulator can be described in several ways. Particular care has been taken to allow students to generate simple experiments with a minimum of effort. The simplest interface will be described here. A more detailed interface also exists that allows a more complicated mix of process to be specified.

An experiment consists of a number of experimental runs. Typically one experimental run is made and additional runs keep almost all of the parameters the same, except for one or two of them. In the simplest case an experiment is specified by a file with extension `.exp` that lists experimental runs and the parameters to be modified. A typical experimental run file is given below.

```
name myexp
comment An experiment with 3 runs
run myrun
run myrun cpuburst uniform 10 90
run myrun cpuburst exponential 50
```

The name of the file is `myexp.exp`. The second line is a comment which is ignored by the simulator but appears in the log file. The subsequent lines specify experimental runs with an optional list a parameters that vary. In the example given above, three runs are made. The first uses the experimental run called `myrun`. The second and third are identical to the first, except for the distributions of CPU burst times.

An experimental run such as `myrun` above must specify the process scheduling algorithm to be used, the number of processes, the arrival time of the first process, and probability distributions for the inter-arrival times, the durations, the CPU bursts, and I/O bursts of the processes. It also specifies the base priority under which the processes should run. A sample file `myrun.run` appears below. Each line begins with a key word that specifies a parameter followed by the value of the parameter.

```
name myrun
comment A sample experimental run file
algorithm SJF
numprocs 20
firstarrival 0.0
interarrival constant 0.0
duration uniform 500.0 1000.0
cpuburst constant 50.0
ioburst constant 1.0
basepriority 1.0
```

The simulator allows for the setting of a seed for the portable random number generator used for calculation of the probability distributions. This facility allows experiments to be exactly repeated so that it is possible to later look at an experiment in detail.

4 A SAMPLE ASSIGNMENT

The simulator is designed to help students see relationships among various parameters and to be able to organize their findings. Two types of experiments are illustrated in this section. Experiment I presents a specific hypothesis that students are asked to support or disprove. In Experiment II students are also asked to develop their own hypotheses.

Experiment I:

Hypothesis: *Round robin (RR) scheduling with n processes makes each user think the machine is running at 1/n-th the speed as long as the I/O times are small.*

1. Devise tests to support or disprove the hypothesis.
2. Conduct a series of experiments to determine the effect of perceived performance as a function of I/O burst time.

Experiment II:

Hypothesis: *Shortest-Job-First (SJF) and First-Come, First-Served (FCFS) are the same when all of the processes have exactly the same CPU burst and I/O burst. When the CPU burst times vary, SJF reduces the average waiting time when compared with FCFS.*

1. Devise tests to support or disprove the hypothesis. Explain your results.
2. Hypothesize on the influence of some other parameter (besides CPU burst variability) on the results produced by these two scheduling algorithms. Devise and run experiments to show the influence and explain the results.

Additional Instructions:

Use the logging features of the simulator to enter your hypotheses, describe the parameters that you are varying and log the results. Before running any experiments, you should also enter into the log a paragraph explaining what results you expect to see. As part of your analysis you should explain how the results confirmed or disproved your expectations. Edit and print out the HTML log files as your report for this assignment.

5 EXPERIENCE WITH THE SIMULATOR

The process scheduling simulator was introduced in an undergraduate operating systems course in the Spring of 1998 after two 50-minute lectures on process scheduling. The simulator was demonstrated during the third class period using a laptop and video projection unit. Most of the approximately 30 students had previously taken two semesters of calculus-based probability and statistics. There were a few students in the class with graduate training in a technical field other than computer

science, and these students showed considerably more sophistication in their designs than the best computer science undergraduates.

The reports were graded and returned to the students. A class critique, summarized below, was posted on the web. The results were also discussed in class. Students indicated that the discussion and critique were useful to them. They also felt that it would be helpful to get feedback on one design before they had to submit the full report. The consensus of the class was that the experience was useful. In addition to gaining experimental experience, students had a much clearer understanding of the mechanics of time sharing and the implications of the CPU burst and I/O burst times.

Some students had difficulty with the precise syntax needed for the input files that specify the experimental parameters. This issue is being addressed by developing a GUI tool for creating these files.

Assignment critique:

The objective of Experiment I was to show that when I/O was small, turnaround = duration * number of processes. A reasonable strategy for Experiment I was to first vary the number of processes keeping the CPU burst and duration fixed. Verify that the relationship holds when the I/O burst is small. After the initial set of experiments, do a second experiment where you increase the I/O burst and see at what point the relationship no longer holds.

Many students did not make good choices for parameters for Experiment I. Some people didn't vary the number of processes at all in this. Memorably bad choices of parameters included:

1. Running the simulation for 1, 2 and 3 processes. (You probably need at least 20 processes to get statistically significant results.)

2. Selecting unrealistic values for the I/O burst time. For example, one person did an extensive study of I/O burst times between 1 and 2 with different distributions when the CPU burst time is 50. Another person studied burst times between 5 and 20 when CPU burst is 50. (These burst times look tiny to any algorithm. More reasonable choices might be 1, 10, 100, 1000, 10000, etc.)

3. Setting a duration of 15 and a CPU burst time of 50. The scheduling algorithm has no influence at all in this case. You should pick a reasonable duration so that the processes have at least 10 bursts (duration / average cpu burst > 10). Varying the duration just tells you when the statistics are significant. You should try to keep the number of burst times approximately constant for this.

The objective of Experiment II was to determine how variability of CPU burst time affects performance of SJF. A reasonable strategy for Experiment II would be to run a few SJF and FCFS experiments with constant CPU and I/O bursts and verify that they give the same results. Then introduce variability into the CPU burst times and see how that influences average waiting time. Variability can be measured by fixing the average for the uniform distribution and increasing the interval around the average. Surprising, any variability has the same effect as a lot of variability on SJF.

Only one person in the class correctly identified how CPU burst variability affects turnaround time for SJF. Many people didn't understand what variability meant. Memorably bad choices of parameters for Experiment II included:

1. Small durations or number of processes as in Experiment I.
2. Thinking variability meant running experiments with different constant CPU burst times.
3. Introducing distributions that had different averages so that you weren't looking just at variability (e.g. constant 50, uniform 50 100, uniform 100 150, uniform 150 200).

Assignment follow-up:

As a follow-up to the project, students were also given the following problem on the final examination in the course:

Propose an experiment (process simulator settings and what you are going to vary) to explore how the quantum could be set based on system load and the characteristics of that load. Give a specific hypothesis that you would test.

The students did reasonably well in selecting appropriate parameters. They managed to avoid most of the pitfalls mentioned in the assignment critique.

6 DISCUSSION

Our preliminary use of the process scheduling simulator in undergraduate operating systems has been very successful. We are currently seeking feedback and participation of other faculty who are interested in incorporating empirical methods at the undergraduate level. If you have any comments or are willing to test the instructional materials, please contact Steve Robbins at `srobbins@utsa.edu`. The process scheduling applet described here is part of a larger project supported by NSF to incorporate an experimental approach into undergraduate operating systems and networks courses. The web site for the project is: `http://vip.cs.utsa.edu/nsf/`. This applet and others are available at this site for general use. Supporting instructional materials are also available at this site.

7 ACKNOWLEDGEMENTS

This work has been supported by NSF grants: *An Electronic Laboratory for Operating Systems and Computer Networks*, DUE-9750953 and *A Web-Based Electronic Laboratory for Operating Systems and Computer Networks*, DUE-9752165.

8 REFERENCES

[1] Silberschatz, A. and Galvin, P. B., *Operating System Concepts*, 5th edition, Addison-Wesley, 1998.

[2] Stallings, W., *Operating Systems*, 2nd edition, Prentice Hall, 1995.

[3] Tanenbaum, A. S., *Modern Operating Systems*, Prentice Hall, 1982.

[4] Tichy, W. F., “Should computer scientists experiment more?” *IEEE Computer*, May 1998, pp. 32–40.

[5] Tichy, W. F., et al., “Experimental evaluation in computer science: A quantitative study,” *J. Systems and Software,* Jan 1995, pp. 1–18.

[6] Zelkowitz, M. V. and Wallace, D. R., “Experimental models for validating technology,” *IEEE Computer*, May 1998, pp. 23–31.

[7] http://vip.cs.utsa.edu/nsf/

[8] http://vip.cs.utsa.edu/nsf/process_scheduling.html