COLBY COLLEGE
Department of Mathematics
and Computer Science

CS357 **Allen B. Downey**
Spring 1998

Assignment 5: Processor Scheduling

Due: Thursday 19 March

The purpose of this assignment is to investigate the processor schedulers on the irises and, as usual, try to figure out what's happening in the kernel by running user-level experiments.

Reading

Read Silberschatz and Galvin, Chapter 5, CPU Scheduling. Pay special attention to Sections 5.3.5 and 5.3.6. Pay less attention to Section 5.5. Pay even less attention to Section 5.6.

Also, please pick up and read the Lottery Scheduling paper from the web page. I think you will find it interesting.

Exercises

The exercises in this chapter are quite good, so I recommend doing them. Exercises 5.3 and 5.4 are the sort of thing I might put on an exam. On the other hand, they tend to be a little monotonous, so I am not going to assign them or collect them. Just do enough of them to convince yourself that you know what's going on. Or, if you don't, then you will know what to ask about.

Build a time bomb

1. Write a program called alarm that executes a mathematically intensive loop. Make the inner loop do something interesting like floating-point division or square roots.

2. Make your program take an argument. The following code fragment shows how to get arguments from the command line:

```
int main (int argc, char *argv[])
{
  int num_children;

  if (argc == 2) {
    num_children = atoi (argv[1]);
  } else {
    printf ("Usage: fork number_of_children\n");
    exit (-1);
  }
}
```

This is from the fork program I used two assignments ago. To execute this program, you type `fork 16` on the command line and it forks 16 children. If you have the wrong number of arguments, it prints a "Usage" message telling you what the arguments are supposed to be. This is a fairly standard interface for UNIX programs.

3. For alarm, the argument should be the number of seconds the loop should continue. You can use the alarm system call to run a program for a given length of time and then quit. Check out the man page for alarm.

4. Run your program with various command-line arguments and confirm that it quits after the appropriate amount of time. You can use the time command as in the last assignment, or you can use the

   `\time`

   command instead, which prints slightly different information about your process (thanks to Peter and Jen who discovered this apparently undocumented feature by accident).

Build a looper

1. Write a second program called cpuloop that runs a mathematically intensive loop for a fixed number of iterations. Make it take the number of iterations as an argument from the command line. Calibrate the program so that you know how many iterations take about 3 seconds of CPU time. It should not print anything inside the loop.

   This program will be CPU-bound; that is, its run time will be primarily determined by how much CPU time it gets. Other programs might be I/O-bound, meaning that their performance is determined by the performance of one of the I/O systems, and relatively insensitive to the amount of CPU that's available.

Pit CPU-bound vs. CPU-bound

1. If you start two processes at the same time, you expect each of them to get about half the CPU cycles. Test this hypothesis by running the following command: `alarm 20 &; \time loop 3`. That will start alarm running in the background and then immediately start loop running (without waiting for alarm complete). Make sure that the alarm time is long enough that loop completes before alarm. What fraction of the CPU time did loop get? Run your programs several times to get an idea of how consistent your results are. If you can, run your programs while the machine is relatively idle.

   NOTE: You should definitely use `\time` for this assignment; `time` does not seem to work correctly when it is on the command line with other commands.

2. Now instead of starting the programs at the same time, introduce a delay between when you start alarm and when you start loop: `alarm 10 &; sleep 1; \time loop 3`. Again, make sure loop has time to complete before alarm. As the delay increases, what happens to the percentage of the CPU that loop gets? What does this relationship tell you about the processor scheduling policy.

   ANOTHER NOTE: Unfortunately, sleep only allows you to introduce delays that are at least a second long. To get intermediate values, you will have to be creative. One possibility is to put something else in place of the sleep statement, for example `alarm 30 &; \time` `ls; \time loop 3`. You can use the first \time info to measure the delay and the second \time info to measure the fraction of CPU time.

CPU-bound vs. I/O bound

1. Modify alarm so that instead of doing only CPU work, it alternates between bursts of CPU and I/O, where the I/O in this case will be printing things on the screen. In other words, it should loop for a little while, print something, loop for a little while, print something, etc.

2. Run the experiment again and see if the behavior of the background process (alarm) affects the amount of CPU time allocated to the foreground process (loop). Modify alarm so that it does different amounts of CPU work between each print statement. How does the frequency of the background process's I/O affect the amount of CPU time the foreground process gets?

3. Modify alarm so that instead of doing I/O to the screen it does I/O to the disk. You can do this internally in alarm, by using fprintf, or you can do it at the shell level by redirecting the output from alarm to a file. Does it change your results when the background process is doing disk I/O instead of screen I/O?

Make the foreground process I/O-bound

1. Modify loop so that instead of being CPU-bound it is I/O-bound. You can check by running the program on an idle machine and checking the percentage of CPU it uses. If it is significantly below 100%, it is probably because it is spending lots of time waiting for I/O.

2. Run your experiments again (or a subset of them—you don't have to be exhaustive), and see what effect a background job has on the performance of I/O-bound jobs.

iris12?

If you are feeling gung-ho, you might want to run some experiments on iris12 to find out what kind of scheduling it is doing. Of course, things are complicated by the fact that there are two processors, so I am not requiring this as part of this assignment. On the other hand, it might make a good term project.

Find the quantum?

1. I would really like for us to find out what the quantum is on these machines, keeping in mind that it may be different for processes with different ages. I can't really think of an experiment that will reveal it, though. Can you?

What to turn in

As usual, I would like a single report from each unit, including an explanation of each experiment you ran, and what you discovered. As in the previous reports, try to integrate your tables and code samples with the text rather than making them appendices. That way you have more incentive to be concise and to explain the relevance of everything you present.

Also, if you observe something you can't explain, don't be shy about describing it as it happened. Resist the temptation to sweep unexpected behavior under the rug. Also, don't feel like you have to offer a definitive explanation for everything—it is reasonable to list two or three possible explanations.

As usual: Label your axes. Mark the data points. Explain every number. Along with every number try to give some indication of its accuracy. Don't report more significant digits than you really have. Use more words than numbers. Use more pages for words than for graphs or code.

Have fun! Learn something!