Introduction

Proportional share scheduling aims to guarantee each job a proportional share of the CPU. Two such algorithms are the Lottery algorithm and the Stride algorithm. The goal of this assignment is better understand scheduling by empirically comparing these two against each other as well as against round robin scheduling.

Analysis (What is the problem?)

The analysis has two aspects: “What is scheduling?” and “What empirical data should be compared?”

To address the first of these aspects, re-read Chapters 7-9. To address the second you will implement three scheduling algorithms: round robin, lottery, and stride.

The second aspect raises two further questions “What is the program’s input?” and “What data should be collected?”

The input is a list of processes, one per line. Each process includes a (space free) name, a number of tickets, and a total CPU need. For example “A 100 10.” The CPU need is given in “ticks,” which is the same unit of time that is used in the simulation’s TIME_SLICE.

In short the answer to the data collection question is the total time taken and the throughput. In greater detail each time your program is run it should output the following for each scheduling algorithm:

1. Each time a process completes execution, print out its name and the current time.
2. When all processes have completed, print the value of TIME_SLICE, the current time, and the throughput (using "%5.2f").
3. For debugging include an option to print the time remaining for each job at the top of each scheduling algorithm’s main loop. Make this code optional by commenting it out (the pedestrian way), using preprocessor directives (the C way), or using a command line switch (the hacker way).

Design (How is the problem solved?)

I will leave to you which data structures to use in your implementation. I will note that unlike your shell, which has to deal with unbounded input, for this assignment you can assume that there are no more than MAX_JOBS processes and that each input line is no longer than BUF_LEN. For example, my code includes

```
#define MAX_JOBS 10
#define BUF_LEN 100
#define TIME_SLICE ...
```

As far as how the data will be analyzed the goal is to investigate the impact of the value of the time slice on throughput. To do this, compute the throughput of each algorithm for several different time-slice values, graph the resulting data, and finally, summarize the graph(s) by considering trends in the data for each scheduling algorithm separately, and then by comparing the trends of the three algorithms.
In addition, run the program ten times, and for each run extract the times that the first job in the input is scheduled (selected to run). (I used `grep` for this.) For each run graph the CPU time remaining on a single graph using different colors for the three different scheduling algorithms. Finally, compute the average of all ten runs and graph all three averages. You can do this in a separate graph or using bold for the averages on a single graph.

While optional, consider also computing average response time (overall and for each process).

Assumptions

1. All process arrive at time zero.
2. Each process arrives with a name, a number of tickets, and a CPU need.
3. When scheduled (assigned the CPU) each process gets a fixed constant amount of CPU time before the CPU is scheduled again.
4. Time is measured in ticks. The current time is the number of ticks since the program started execution.
5. Use `arc4random()` (see its man page for why).

What to hand in

1. Hand in a 2up nicely formatted copy of your source code.
2. Commit and push your code to GitHub. Include a `makefile` such that the command “`make data`” will build the executable if needed and then compute the empirical data. (Consider include the target `make debug` that builds and runs the program to output the debugging information listed above.) Finally, avoid committing derived binary files.
3. Edit the `README.md` file in your repository to clearly contain a description of how your program works (i.e., your design) and a statement that you have not violated the honor code in doing this assignment.