CS4328: Project #1

Due on Mar, 29, 2019 at 11:59PM

Mina Guirguis

This project is going to take a good amount of work and time, so please start early. You would not be able to finish if you start few days before the due date. Late submissions would incur a penalty of 20% per day for up to 2 days, then they will not be accepted. Leave the last few days for documentation, further testing and formatting the results. Please read the description carefully and come see me (hopefully early) if you have any questions. You may discuss this project with other students. However, you must write your code and your report on your own.

1 Overview

In this project, we are going to build a discrete-time event simulator for a number of CPU schedulers on a single CPU system. The goal of this project is to compare and assess the impact of different schedulers on different performance metrics, and across multiple workloads.

1.1 CPU Scheduling Algorithms

We are going to implement the following scheduling algorithms that we discussed in class:

1. First-Come First-Served (FCFS)
2. Shortest Remaining Time First (SRTF)
3. Highest Response Ratio Next (HRRN)
4. Round Robin, with different quantum values (RR)

1.2 Performance Metrics

We are interested to compute the following metrics, for each experiment:

- The average turnaround time
- The total throughput (number of processes done per unit time)
- The CPU utilization
- The average number of processes in the ready queue

2 The Simulator

The simulator needs to generate a list of processes. For each process, we need to generate its *arrival time* and its requested *service time*. We can assume that processes arrive with an average rate λ that follows a Poisson process (hence exponential inter-arrival times). The service times are generated according to an exponential distribution. We will vary λ to simulate different loads while keeping the average service time fixed. The simulator should stop after processing 10,000 processes to completion (without stopping the arrival process), then it should output the statistics (i.e., the metrics above).

Events (e.g., process arrival, process completion, time-slice) that occur causes the simulator to update its current state (e.g., cpu busy/idle, number of processes in the ready queue, etc.) To keep track and process events in the right order, we keep events in a priority queue (called "Event Queue") that describes the future events and is kept sorted by the time of each event. The simulator keeps a clock the represents the current time which takes the time of the first event in the Event Queue. Notice that when an event is processed at its assigned time, one or more future events may be added to the Event Queue. For example, when a process gets serviced by the CPU, another process can start executing (if one is waiting in the ready queue)

and under FCFS, we know exactly when this process would finish (since FCFS is non-preeptive), so we can schedule a departure event in the future and place it in the event queue. Notice that time hops between events, so you would need to update your simulator clock accordingly.

The simulator should take few command-line arguments. The first is to indicate the scheduler, a 1 through 4 value based on the list above. Also, it should take other arguments such as the average arrival rate, the average service time and the quantum interval (for RR). Running the simulator with no arguments, should display the parameters usage.

Each scheduler would need to maintain a queue (the "Process Ready Queue") for the ready processes that are waiting for the CPU. A scheduler will select a process to run next based on the scheduling policy. Clearly, this queue should not be confused with the Event Queue that is used to hold events to be processed in the future.

3 The Runs

We will vary the average arrival rate, λ, of processes from 1 process per second to 30 processes per second (based on a Poisson process). The service time is chosen according to an exponential distribution with an average service time of 0.06 sec.

For each value of λ, we need to compare the performance of each scheduler, based on the metrics above. It is recommended that you write a simple batch file that would run those experiments and put the results in a file (that you can later import into a spread sheet and plot the values).

```
#!/bin/bash
rm sim.data
for ((i = 1; i < 31; i++)); do
   ./sim 1 $i 0.06 0.01
   cp sim.data /data/1-$i-006.data
done
```

This will run the simulator using FCFS (indicated by the value 1 in the first argument) for 30 different values of λ using 0.06 as the average service time and a quantum value of 0.01 (which is ignored in all algorithms, except round robin). Then, the script will move the sim.data file to a safe place.

With the Round Robin algorithm, an argument is supplied to indicate the quantum used. Use 2 different values of quantum; 0.01 and 0.2. Clearly, if a process finishes before its quantum expires, the CPU will schedule the next process right away.

4 Submission details

Submissions are done through TRACS. Submissions will include the code and how to compile and run the simulator on one of the CS servers, along with a report containing the results and their interpretation.

The report will include the results of the experiments along with a description. We will run 5 different algorithm (since the round-robin will have 2 different settings for quantum values), each for 30 different values of λ. A total of 150 runs! The report should include a single plot for each one of the above metrics. The plot on the x-axis will vary λ and represent the metric on the y-axis with different line color for each scheduler.

You can write your simulator in any of these languages (C, C++, Python or Java), however, it is your responsibility to ensure it runs under the CS Linux servers with a command line – nothing graphical. Please indicate clearly how to compile and run your simulator.

Grading breakdown: 30% of the grade is on developing the correct design and data structures (e.g., event queue, ready queue, etc.) for the simulator. 60% of the grade is on obtaining the correct results (i.e., the metrics above) for the schedulers. 10% of the grade is on proper documentation (i.e., explanation of the results, providing the compile and run command lines, etc.).