CS 3360 Programming Assignment 2

Due: 11:59pm Tuesday, March 28, 2023

Instructor: Kecheng Yang (yangk@txstate.edu)
Doctoral Instructional Assistant: Muhieddine Shebaro (m.shebaro@txstate.edu)

Instructions

You can write your program in any of these languages (C, C++, Python or Java), however, it is your responsibility to ensure it compiles and runs under the **CS Department Linux server (zeus.cs.txstate.edu)** with a command line (i.e. no GUI, no IDE). Please include a README to indicate clearly how to compile and run your program. **Submissions that do not compile as described above may not receive credits**, even if it compiles and runs in other environment such as various IDEs.

Your submission should include:

- **SOURCE CODE**;
- **README** text file, which indicates how to compare and run your program, including all command lines to compile, run, and test your code on zeus.cs.txstate.edu (i.e., starting from the first command right after log in on zeus.cs.txstate.edu);
- A **REPORT** in MS Word or PDF format that briefly describes your submitted files and programs relating to the problems, report the results by your program, and answer the questions if there are any asked in the problems.

Overview

In this project, we are going to build a discrete-time event simulator for a First-Come First-Served (FCFS) CPU scheduling algorithm. The goal of this project is to assess the impact of different workloads on the following (observed) performance metrics:

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

The simulator needs to generate a sequence 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 and the arrivals follow a Poisson Distribution (hence Exponentially Distributed interarrival 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 10,000 processes complete, then it should output the metrics above.

Events (e.g., process arrival, process departure) that occur causes the simulator to update its current state (e.g., CPU busy/idle, number of processes in the Ready Queue). Events are to be kept in a priority queue (which can be implemented by a sorted linked list) called Event Queue that describes the future events and is kept sorted by the time of each event. The simulator keeps a clock variable the represents the current time which is initially set as 0. The initialization also creates the first event --- the arrival of the first process (its time can be obtained by 0 plus a generated interarrival time) and adds it into the Event Queue. As the simulator runs and events are handled, additional future events may be added to the Event Queue. For example, when the simulation clock reaches an arrival event, we can schedule the next arrival event and determine its event time by generating an interarrival time. As another example, at the time when a process just begins its service (execution) on CPU, we can schedule a departure event in the future and determine its event time (i.e., the finish time of this process just begins being serviced) by generating a service time, because FCFS is non-preemptive. Notice that time hops between events, so you would need to update your simulator clock accordingly.

Note that the Ready Queue is not to be confused with the Event Queue. For the purpose of this assignment, it might not be necessary to actually implement a Ready Queue; rather, keep number of processes in the Ready Queue as part of the system state should be sufficient. On the other hand, Event Queue would need to be implemented (as a sorted linked list) to progress the simulation.

The slides about "A Simple Example" and "Metrics for PA2" in the "Discrete-Time Event Simulation" could be helpful for this assignment.

You might need auxiliary variables/codes (i.e., in addition to those necessary for progressing the simulation) to keep log records, in order to obtain the four itemized metrics above at the end of a simulation run.

The Runs

The simulator should take 2 command-line arguments (or write a "simulator-run" function that takes 2 arguments, if command-line stuffs are too difficult to you): The first is the average arrival rate and the second is the average service time. We will vary the average arrival rate, λ, of processes from 10 processes per second to 30 processes per second with increment step of 1 (i.e., we are doing 21 simulation runs for $\lambda = 10,\ \lambda = 11,\ \lambda = 12,\ \ldots\ \lambda = 30$, respectively). The service time is generated according to an Exponential Distribution with an average service time of 0.04 second. For each run, you would need to output the four itemized metrics above.

The Plots

The report would include a brief description of the results and show a single plot for each one of the above metrics (with λ as the x-axis). That is, each plot would have 21 data point, and there are four plots (one for each metrics). A brief description and/or discussion on the interpretation of the plots should also be included.