CS3360: Homework #5

Due on Nov, 21, 2025 @ 11:59 pm (firm)

Mina Guirguis

PLEASE READ: You may discuss this problem set with other students as long as you do not share/verify answers. However, you must also write the names of other students you discussed any problem with. You must *write up your answers on your own showing your work.* Each problem has a different weight. Please state any assumptions you are making in solving a given problem. Late assignments will not be accepted with prior arrangements. By submitting this assignment, you acknowledge that you have read the course syllabus and abiding to the copyright notice (e.g., no posting) and the Honor Code (e.g., not using note sharing sites such as Chegg, OneClass, CourseHero, etc. as well as using chatGPT) requirements.

Problem 1

Overview: In this project, we are going to build a discrete-time event simulator (similar to our previous project) to assess the speedups realized from using multiple CPUs in which the First-Come First-Served (FCFS) scheduling algorithms are used. The goal of this project is to assess the impact of different designs on the following performance metrics:

- The average turnaround time of processes
- The average throughput (number of processes done per unit time) for the system
- The average utilization of every CPU
- The average number of processes in the Ready queue(s)

Similar to our previous project, 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 10,000 processes complete, then it should output the metrics above.

We are interested to study the following 2 design scenarios:

1- Scenario 1: Every CPU has its own Ready Queue. When a process arrives, it selects one of the CPUs uniformly at random (e.g., if we have 4 CPUs, a process would select a particular CPU with probability 0.25).

1- Scenario 2: All CPUs share a global Ready Queue. When a process arrives and none of the CPUs are available, it joins the Ready Queue. Once a CPU becomes available, it would dispatch a process from the Ready Queue.

Events (e.g., process arrival, process departure) that occur causes the simulator to update its current state (e.g., status of every CPU and contents of the Ready Queues). Events are to be kept 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 variable the represents the current time which initially takes the time of the first event in the Event Queue and is updated when any event occurs. As the simulator runs and events are handled, additional future events may be added to the Event Queue.

The Runs: The simulator should take 4 command-line arguments: The first is the average arrival rate, the second is the average service time, the third is which scenario (i.e., 1 or 2), and the fourth is the number of CPUs.

We will vary the average arrival rate, λ, of processes from 50 processes per second to 150 processes per second (based on a Poisson process) in steps of 10. The service time of a process is chosen according to an exponential distribution with an average service time of 0.02 sec. For each scenario, we will study the case with 4 CPUs. For each run, you would need to output the four metrics above.

The Output and submission details: An output report would include a brief description of the results (based on the parameters above for both scenarios) and show plots of the above metrics with λ on the x-axis and the metric on the y-axis. Submissions are done through Canvas. Submissions will include the code, how to compile and run the simulator on one of the CS servers, along with a report containing the results and their interpretation.

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 queues, etc.) for the simulator. 60% of the grade is on obtaining the correct results (i.e., the metrics above). 10% of the grade is on proper documentation (i.e., explanation of the results, providing the compile and run command lines, etc.). This project is worth 100 points.