CS4328: Project #1

Due on Mar, 22, 2024 at 11:55PM (Firm)

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 10% 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 to simulate two interactive components of an operating system which are the CPU and the Disk. The goal of this project is to assess the performance of those two components under multiple workloads (by changing the rate of arrivals of processes – the higher the arrival rate, the higher is the load).

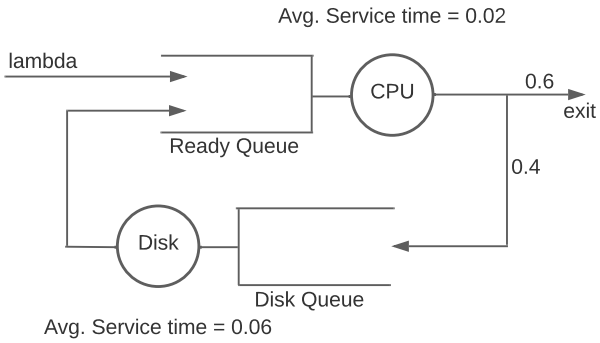

Figure 1: An Overview of the OS components to simulate

1.1 The Workload

We assume that all processes arrive to the CPU with an average arrival rate of λ (Lambda) that follows a Poisson distribution. Once a process arrives, if the CPU is idle, it goes directly for processing, otherwise, it joins the "Ready Queue" waiting for the CPU. The CPU allocates service times that are exponentially distributed with an average service time of T_s^{CPU}. Once a process is done with the CPU it exits with probability 0.6 (i.e., it is done) or goes to the Disk system for service with the remaining probability. The Disk allocates service times that are exponentially distributed with an average service time of T_s^{Disk}. After a process gets service from the Disk, it goes to the CPU system. We assume the CPU and Disk service process in FCFS. Figure 1 describes the system to simulate.

1.2 Performance Metrics

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

- The average turnaround time of the processes that finished
- The average throughput (number of processes done per unit time)
- The average CPU utilization
- The average Disk utilization

- The average number of processes in the CPU Ready queue
- The average number of processes in the Disk Queue.

2 The Simulator

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

Events (e.g., process arrival, process completion on the CPU/Disk) that occur causes the simulator to update its current state (e.g., cpu/disk busy/idle, number of processes in the ready/disk queue, etc.) To keep track and handle 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 variable the represents the current time which initially takes the time of the first event in the Event Queue and gets updated when any event happens. Notice that when an event is handled 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, it may join the Disk queue based on the probabilities and this would create a new event. Notice also that time hops between events, so you would need to update your simulator clock accordingly.

The simulator should take 3 command-line arguments. The first is the average arrival rate λ; the second is the average CPU service time; the third is the average Disk service time. Running the simulator with no arguments, should display the parameters usage. The CPU will maintain a queue (the "Process Ready Queue") for the ready processes that are waiting for the CPU. Similarly, the Disk will maintain a queue (the "Disk Queue") to hold processes that are waiting for the Disk. Clearly, these queues 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 times for the CPU are chosen according to an exponential distribution with an average service time (T_s^{CPU}) of 0.02 sec, while the service times for the Disk are chosen according to an exponential distribution with an average service time (T_s^{Disk})of 0.06 sec. For each value of λ, we need to plot the above metrics, each on a separate plot. It is recommended (but not required) 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).

4 Submission details

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.

The report will include the results of the experiments along with a description. 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 of interest on the y-axis.

You can write your simulator in any of these languages (C, C++, Python or Java), however, **it is your responsibility to ensure that it runs under the CS Linux servers with a command line – nothing graphical**. Do not wait to the last minute to test your program. 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, disk queue , 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.).