CIS–4020 Lab Scheduling Simulation

Last Revised: October 4, 2016

Introduction

In this lab you will write a program that simulates a simple operating system scheduler. We will consider only the non-preemptive case. Each job, or *CPU burst*, is allowed to run for as long as it wants, a time we will call its service time, T_s. The jobs will arrive to be serviced at a rate of λ jobs per unit time. Each job will, perhaps, wait in a queue of w jobs for a time T_w. The total time required to process a job will be the turnaround time T_r given by $T_w + T_s$. The normalized turnaround time is T_r/T_s. The goal of this lab is to simulate the random arrival of some number of jobs and compute average values for the normalized turnaround time, queue length, and processor utilization. You will examine two different scheduling policies: "First Come First Served" (FCFS) and "Shortest Job Next" (SJN). An extension of this lab explores SJN in more detail by having you experiment with different methods the operating system might use to estimate a job's service time.

Although many systems use preemptive scheduling, the non-preemptive case is still important. In a system where a process is allowed to run until it blocks, each burst of activity from the process could be considered, for our purposes here, as a separate job. For this reason I use the term "job" in this handout rather than "process."

1 Simulation Parameters

Some of the simulation parameters are to be provided by the user while others may be built into the software. The primary input parameters are:

1. The total amount of simulated time in time steps. You can imagine each time step as being one millisecond but keep in mind that the results can be scaled to any other time frame.

2. The average rate at which jobs arrive, λ (in jobs per time step). In our simulations λ will be significantly less than one. For example, if one job is to arrive every 10 time steps on average then $\lambda = 0.1$.

3. The average service time for each job, β (in time steps). Keep in mind that when $\lambda \geq 1/\beta$ the system is in saturation and jobs are arriving faster, on average, than they can be processed. You should avoid this situation in your simulations (although you can and should approach it). You will want to obey the following relationship

   $$1 > 1/\beta > \lambda$$

   I suggest choosing a beta value of 10 and keeping it fixed during the simulation.

You should assume that jobs arrive "randomly" and independently. In that case, the probability that n jobs will arrive during a particular time step is given by a Poisson Distribution.

$$P(n) = \frac{e^{-\lambda}\lambda^n}{n!}$$

Where n is the number of jobs arriving in a time step and λ is as defined above. When λ is significantly less

than one, $n = 0$ is the most likely possibility. You can assume that the only other possibility is $n = 1$. *Note:* for large λ, the possibility of multiple jobs arriving during one time step can not be ignored.

To determine if a job arrives during a particular time step, compute $P(0)$ for the given λ. Then draw a uniform random value from the half open interval [0, 1) and see if it is less than $1 - P(0)$. If it is, then you can assume that a job has arrived.

Although in real life operating systems rarely have the luxury of knowing how long a job will take before executing it. We will assume that the length of each job is known beforehand. This information might come from the user or it might be estimated from past performance of the same job (for some suitable definition of "same"). Our desire here it to choose a distribution of service times that is realistic and yet also easy to handle. Assuming that all jobs have exactly the same service time is not realistic. Assuming that the service times are distributed uniformly over some range is not very realistic either. Instead you should make the common assumption that service times are distributed according to an exponential distribution. In particular

$$f(x) = \frac{1}{\beta} e^{-x/\beta}$$

where β is the average service time length. To generate service times from this distribution, first draw a uniform random value from the half open interval [0, 1). Call that value r. Using the inverse of the above function, a service time can be computed from

$$T_s = -\beta \ln(r)$$

You will need to convert T_s into a suitable integer number of time steps.

2 Implementation Details

Your program should be a loop that executes once for each simulated time step. If a job arrives at a particular time step you should assume that it arrives at the beginning of the time step. An instantaneous dispatching choice (from the queue) is made, if appropriate, and the processor either works or is idle for the full duration of the time step. Your program should track the following parameters:

1. The number of items in the queue during each time step.
2. The total number of time steps during which the processor is busy.
3. The number of time steps each job spends waiting in the queue.
4. The normalized turnaround time for each completed job.

Your program should compute average queue length (over time), average processor utilization (over time), and average normalized turnaround time (over completed jobs).

You will probably want to create job objects to represent the waiting jobs and maybe also the running job. At each time step decide if a new job arrives using the Poisson distribution. If so, create a job object and add it to the queue. If the processor is available, dispatch a job from the queue to the processor using whichever scheduling policy you are testing. Adjust counters and other statistics as necessary.

Enclose the program in an outer loop that steps λ from some small value up to just below saturation. Use at least 20 steps. Output tables of the averaged parameters for each step in λ and then import the table into a spreadsheet program to graph the results. Repeat the experiment for the two scheduling policies mentioned earlier: FCFS and SJN. Note that to get good results you will probably need to run the simulation for millions of time steps.

3 Estimating T_s (Optional)

The simulation of SJN described above assumes the operating system knows how long each job will take

before the job runs. Of course, it is not possible to know this and so real systems must estimate the expected value of T_s in some manner. In this optional extension to the lab you will experiment with two different methods of creating such estimates.

To implement this additional feature you will need to distinguish between actual service times (which you compute as before) and estimated service times which you will compute using an appropriate estimation algorithm. The scheduling decision is made based on the estimated times. However, you will need to continue tracking the actual service times because once a job reaches the CPU the time it consumes will be the actual time and not the estimate. Also computations of turn around time must us the actual value of T_s.

The two algorithms you should explore are as follows.

1. Use the last actual time (on the CPU) as an estimate of T_s.
2. Use an exponential average of the actual time as an estimate of T_s. This is given by the formula

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

   Where τ_n is the current estimated time, t_n is the actual time of the last job, and τ_{n+1} is the next estimated time. You can experiment with different values of α, but try for example $\alpha = 0.5$. Note that the first algorithm mentioned above is really the same as the second algorithm with $\alpha = 1.0$.

Does exponential averaging produce better results than the simpler method of just using $\alpha = 1.0$?