**CS4328: Project #1**

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

*Mina Guirguis*

1

*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.*

# Miscellanous

# Stuff He said

# How do you track list of proceeses create class

# Id: Service Time: how much does it need, you need to gather all that for every process.

# Like a process control block. So its a data structure that contains these, when did it arrive, how much times does it need, how many times does it go through round robin...

# Example of State Shit

# You can make a state, so at time 7 7:10 someone comes in, so theres one person in the room, at 7:12 did this number change. 7:22

# Can track when events happen it helps with that.

# Event Vs Ready Queue

# Ready Carries Events

# Event Carries Processes

# Event Queue

# Event Queue is IMPORTANT :

# Imagine want to simulate simple example, arrival 7:10, 7:14, and

# Departure 7: 25

# So it's better to have que set in time. Going to have a struct called event,

# Linked list between these events A (arrive)-> b(arrive)-> C(Departure)

# clock = 7:

# state # of people in class = 0

# See photos.

# clock = 7:10 arrival event

# State = 1

# Remove that event.

# Clock 7:18

# State = 2

# Remove Event

# // Every time you process it doesn't matter if you have the event queue the state holds it.

# Clock = 7:22

# State =3

# Remove Event.

# // So if I set simulation end time, it wouldn't process if it ends earlier than process 7:30 if it did .

# Event Queue holds just events.

# Time and event, and a pointer to a process. not differentiating between them. Want to compute an average turn around time.

# EVENTS, STATE ,CLOCK

# EVENTS -- these change the state.

# Process finishes so we can remove from the system that's what we are gonna talk about.

# STATE -- Data Structures (Example, Ready Queue)

# As processes change the state.

# CLOCK - A GLOBAL variable Double Clock is going to update in there, anything inbetween these times, nothing changes so theres no need to cahnge it, if there's an hour where nothing happened then we can skip right away to that.

# 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.

## CPU Scheduling Algorithms

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

* + 1. First-Come First-Served (FCFS)
       - FCFS Example
         * See photo.
         * P1 = 3 cpu units A
         * time = 2
         * p2 = 5 cpu units B
         * time = 4
         * P3 = 2 cpu ints C
         * time = 7
         * Need to have a ready queue, based on whats going ot happen. The state is going to be in the ready queue.
         * What else would help?
         * Adding to the state:
         * Is the CPU idle or not, is the CPU Idle or busy
         * Ex.
         * Clock = 0
         * Cpu = idle ==1
         * R.Q = empty.
         * EVent Queue Time = 0 // not part of the state.
         * A -> B -> C
         * // Then look at event queue see next event is going to happen at time 2.
         * Clock = 2
         * // Cpu is idle when is it going to leave.
         * //Leave at time 5.
         * // Can sceudle a departure event at time 5, and actually process A. Create that event at time 2 because I knew this was going to happen in the future. Need to update state at 4 then at 5 and then at 7.
         * // WE plug it in where it needs to go in the event queue
         * CPU idle == 0 // false
         * R.Q = Empty
         * Clock = 4
         * Cpu Idle == 0
         * R.Q = p2. // Notice we don't schedule when p2 leaves.
         * Clock moves = 5 // Departure event Go into R.Q and see whats next. When does P2 going to be done, we are at 5 . Schedule am event at time 2 thats a departure, placed in event queue then that comes after C .
         * R.Q = empty // because we moved it to the event queue? // we can now remove 5
         * Clcok 7 // sees p3 in a process and repeats.
         * Simplier because first come first serve and non preembtipve.
       - Side Notes
         * When we make these decisions what kind of event accuses you to call the scheduler? the departures, when someone leaves you look at it to decide what happens to it.
    2. Shortest Remaining Time First (SRTF)
    3. Highest Response Ratio Next (HRRN)
    4. Round Robin, with different quantum values (RR)
       - Time slice take process that was in ready queue, then remember how each process goes through this, every time it cuts service and every time it goes through when time hits zero schedule a departure

## Performance Metrics

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

* The average turnaround time
  + Simulator generates the times with an average of .66 For Arrival times.
  + How do we do it? every process needs service time and arrival time.
  + HW2 will have a function, that gives you... See photo.
  + Rand() and lambda you get a random # that is exponentiallly distributed 1 - e^ (-Lamnda\*x) That gives you arrival times.
  + Only differnce in service time is to switch lambda to mu?
* The total throughput (number of processes done per unit time)
* The CPU utilization
  + Ready Que Size is y Axis two process comes at timex axis
  + Time 0 one comes in and then 2 another arrival came, size of the queue is 2, if you want to compute the average queue size, Spend a long time in one over one time in 2, The queue size was one for how long and then compute it like that.
  + Another easy way is to sample the queue at another point of time maybe here, then I'm gonna average over all these readings. First method is more effort to figure out how long queue size was for how many for how long.
  + **Second one is easier create an event sample ready queue sample them apart, every time you get one of the events find the size of ready queue then you get an example of that**.
  + As Long as every Run each of the processs.
* The average number of processes in the ready queue

# The Simulator Notes

# Lambda Photo.

# 1,2,3,4,5.....30

# Ts = 0.06

# Mu = service rate = 1/Ts = 16.666

# Rho = Lambda / mu = Lambda \* Ts

# at 16.66 Beyond this point in time, the system is overloaded, meaning arrival rate is higher than the service rate.

# You cannot go over 100 that's the Utilization of the server.

# What's going to grow as arrival rate, happens ready queue will start increasing and decreasing. The queuing stuff we did can't handle any of the equations where arrival rate is larger than departure rate. So Our simulator can tell us what we happen. The analysis will fall short. When the arrival is 30, is there any formula to know how many processes are in the R.Q? we don't have a formula but when we simulate we can.

# OVERLOAD happening between 16-17

# Verification

# Verification, FCFS should match almost perfectly with what we have in queuing time, Tq Cpu Utilization.

# 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.

# 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.

# 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.).