DATA STRUCTURES

ECE 368 - SPRING 2018

PROJECT 1: Event Driven Simulation for a Multiprocessor System Using a Priority Queue (It is a team project. Note: 2 students/team)

ASSIGNMENT DATE: January 23, 2018

NOTE; This project has two parts which have different due dates.

NOTE: DO NOT COPY (EVEN PARTIALLY) SOMEONE ELSE CODE. ANY SUCH BAHVIOR WILL AUOMATIACLLY GIVE YOU A ZERO GARDE AND POSSIBLY AN "F" GARDE IN THE COURSE.

Part 1

DUE DATE: February 8, 2018
Simulator Program Due by 3:00pm

You need to write an event-driven simulation in C to evaluate the performance of a single CPU computer system with a priority-based queuing discipline. The operating environment of the system that needs to be simulated is described below.

1. Assume two types of tasks arrive at the system, having two priority levels; 0 for high, 1 for low. Tasks with priority 0 arrive according to a Poisson process with rate λ_0. Similarly, tasks with priority 1 arrive according to a Poisson process with rate λ_1.
2. Tasks with priority 0 are served first. Tasks with priority 1 are served only if no task with priority 0 is waiting. Assume a non-preemptive system.
3. Once the CPU is allocated to a task, the task requires a random service time which is exponentially distributed with average of $\frac{1}{\mu}$ units of time.

IMPLEMENTATION DETAIL

Your simulator must accept values at run time for λ_0, λ_1, μ and the total number of tasks to be simulated in each group (assume both groups have the same number of tasks). You need to conduct various experiments and collect data for four performance parameters, namely: the average waiting time of Type 0 and Type 1 tasks, the average queue length, and the average utilization of CPU. These numbers should be written in the given order to the file "proj1-

a_output", one per line. The average utilization of CPU is the ratio of the time the CPU is performing some useful computation to the total time of simulation.

You need to use the uniform random number generator routine, rand(), to generate the required exponential time distributions corresponding to λ_0, λ_1, and μ. DO NOT USE srand() ROUTINE.

Modes of Operation

For testing purposes, your simulator must run in two distinct modes. Mode 1 works exactly as described above, using the variables λ_0, λ_1 and μ in conjunction with a random number generator to create tasks. Mode 2 will simulate a predetermined sequence of tasks described in a text file.

- You will write a single program to operate in both modes based on the command line arguments

- **The command line format for Mode 1 will be**

```
<executable name> <λ0> <λ1 > <μ> <total tasks in a priority group>
```

```
        example:  project1-A 0.5 0.7 1  10000
```

- **The command line format for Mode 2 will be**

```
        <executable name> <input file name>

        example:  project1-A input.txt
```

The input file used in Mode 2 will contain the information for all the tasks to be simulated, one task per line. Tasks are listed in the order of arrival. Your program should read to the end of the input file and should exit only after all the tasks are processed. The format of a line in this file is:

```
        <arrival time> <priority> <duration>
```

where `duration is the service time of the task.`

```
        example:  1 0 4
```

The above line would be read as "A task arrives at time t=1. This task has priority 0 and has a service time of 4 units."

Additional Considerations:

1. Assume two simultaneous arrivals at t=0, corresponding to both types of tasks. Subsequently, generate two streams of arrivals independently.

2. Note: two tasks with different priorities can arrive simultaneously.

3. The observation of the queue length, for computing its average size, is done at every arrival.

4. Inter-arrival time generated by the random number generator should be rounded up to the next integer value.

5. For stability, we will assume $\lambda_0 + \lambda_1 < \mu$.. For testing your simulation, select these parameters accordingly. For grading your program, we will choose these parameters that satisfy the above inequality.

How to submit your project:

When you are ready to submit your project file(s), you should have a directory PRJ1_A where your file(s) is stored. Go to the directory which contains the directory PRJ1_A and execute the following UNIX command:

```
turnin -c ee368ta -p proj1_A_s18 PRJ1_A
```

Your whole directory will be submitted for grading. You can check the submitted file(s) using the following UNIX command:

```
turnin -c ee368ta -p proj1_A_s18 -v
```

Math Library

In addition to including the math.h header file, you need to use the compile flag **-lm** to access the math library functions.

Note on Operating System

Your project will be graded on a Linux machine.

Part 2

DUE DATE: Simulator Program Due on February 22, 2018, by 3:00pm,
Written Report Due: February 27, 2018 (in the class)

You need to write an event-driven simulation in C to evaluate the performance of a multiprocessor system with 64 processors and also write a comprehensive report on this project. Report must be typed. The operating environment of the system that needs to be simulated is described below.

1. Assume two types of tasks arriving at the system, having two priority levels; 0 for high, 1 for low. Tasks with priority 0 arrive according to a Poisson process with rate λ_0. Similarly, tasks with priority 1 arrive according to a Poisson process with rate λ_1.

2. Tasks with priority 0 are served first. Tasks with priority 1 are served only if no task with priority 0 is waiting.

3. At the time a task is selected for service, it is broken into multiple sub-tasks and, hence, requires a random number of processors. Assume, the number of sub-tasks follows a uniformly distributed random variable with discrete values between 1 and 32. At the time of service, if the desired number of processors are not available for the task present at the front of the queue, that task stays in the queue and the queue is scanned to choose another task that can be served with the currently available number of processors. In other words, the *first-fit strategy* is used for multiprocessor scheduling. You may need to scan the whole queue (starting from the head of the queue) to identify a task for scheduling. The selected task must have the number of sub-tasks which is less than or equal to the number of currently available processors. Assume, scanning of the queue requires 0 time.

    - Note, multiples tasks can be loaded in a given scan, provided the number of processors available are sufficient.
    - If no task can be scheduled from the queue in a given scan, then depending upon the following events, the system proceeds as follows:
        a. It must wait till more processors become available, which triggers a new scan.
        b. Some new task with sufficiently small number of sub-tasks arrives, which is then immediately loaded onto the available processors.
        c. If both events (a) and (b) happen simultaneously, give preference to event (a)

4. Once the processors are allocated to sub-tasks of a task at the start of service, each sub-task uses its allocated processor for a random amount of time which is exponentially distributed with average of $\frac{1}{\mu}$ units of time. When a sub-task of a task completes, it releases its allocated processor immediately. The freed processor becomes available for other tasks in the queue. At that time, the queue is scanned, as mentioned above. Note, all the subtasks of a task start at the same time, however, their individual completion time is random and upon completion they independently free their processors,

Note: Additional considerations of Part 1 remain valid.

Preemption: There is no preemption in the system. Once a task is scheduled to a group of processors, all its subtasks must be executed to completion.

<u>Implementation Detail and Reporting</u>

Your simulator must accept values for λ_0, λ_1, and μ at run time. You need to conduct various experiments and collect data for <u>five performance parameters</u>, namely: the average waiting time of type 0 and type 1 tasks, the average queue length, the average utilization of each processor, and an average load balancing factor (average over all the tasks). These numbers should be written, in the order given, to the file "proj1_output", one per line. The average utilization of a processor is the ratio of the time the processor is performing some useful computation to the total time of simulation. The load balancing factor for a task is $\frac{\frac{1}{\mu_{\min}} - \frac{1}{\mu_{\max}}}{\frac{1}{\mu}}$ where $\frac{1}{\mu_{\min}}$ is the CPU time taken by the longest sub-task and $\frac{1}{\mu_{\max}}$ is the CPU time taken by the shortest sub-task of the task.

Plots for the Report

Use the following parameters for your plots. The values (including the definitions of ρ_0 and ρ_1) have been chosen to allow stability in the simulation.

Run the simulation for 10,000 arrivals (in each group).
Let $\mu = 0.2$
$\rho_0 = \lambda_0 / (4* \mu)$
$\rho_1 = \lambda_1 / (4* \mu)$

Plot results for ρ_0 = 0.1, 0.3, 0.5, 0.7 with ρ_1 = 0.3
Plot results for ρ_1 = 0.1, 0.3, 0.5, 0.7 with ρ_0 = 0.3
Use μ to calculate appropriate values for λ_0 and λ_1

MODES OF OPERATION AND LENGTH OF SIMULATION

For testing purpose, your simulator must run in two distinct modes. Mode 1 works exactly as described in Part 1. You will simulate the arrival of the specified number of tasks and finish your simulation only when all tasks are completed.

Mode 2 will simulate a predetermined sequence of tasks described in a text file.

- You will write a single program to operate in both modes based on the command line arguments
- The command line format for Mode 1 will be

```
<executable name> <lambda0> <lambda1> <mu> <number of tasks>
```

    example: `project1-B 0.5 0.7 1 15000`

- The command line format for Mode 2 will be

```
<executable name> <input file name>
```

    example: `project1-B input.txt`

The input file used in Mode 2 will contain the information for all the tasks to be simulated, one task per line, in order of arrival. The format of a line of this file will be

```
<arrival time> <priority> <number> <duration list>
```

where `“number”` is the number of subtasks belonging to a task and `the duration list` contains the durations of these sub-tasks .

example: `1 0 4 6 7 8 9`

The above line would be read as “A task arrives at time 1. This task has priority 0 and breaks up into 4 subtasks, of duration 6, 7, 8 and 9”.
For Mode 2, your program should read to the end of the input file and exit only after all tasks are completed.

How to submit your project:

When you are ready to submit your project, you should have a directory PRJ1_B where your file(s) is stored. Go to the directory which contains the directory PRJ1_B and run the following UNIX command:

```
turnin -c ee368ta -p proj1_B_s18 PRJ1_B
```

Your whole directory will be submitted for grading. You can check the submitted file(s) using the following UNIX command:

```
turnin -c ee368ta -p proj1_B_s18 -v
```