CSCD 340
Lab 8

One of the classic problems with teaching operating systems, especially scheduling algorithms is that the trace of the algorithm is always done within a "bubble", with the processes always arriving at the same time. Then a lab or homework is given where the student has to create hand traces of the algorithms. The same classic problem still exists, everything is done in a "bubble". On simple solution is to have students code a scheduler based on some criteria. This is a viable solution but it is hard to code a solution for all the algorithms. In order to try to understand the classic algorithms it is best to use a simulator. This simulator is created and provided by Steve Robbins from the University of Texas San Antonio, which was supported by NSF DUE-9752165.

1. Attached to this lab is a zip file and a PDF. Download and extract the zip file. NOTE: this will make a directory named **psUnix**.

2. Down load and scan through the PDF. This PDF explains some of the setting of the simulator.

3. Become familiar with the simulator by completing the following

   - In the **psUnix** directory, execute **`runps`**. This will start the process scheduling simulator. You may have to chmod it.

   - Click on the green button labeled **Run Experiment** in the lower left-hand corner of the Process Scheduling Simulator window. This will start the simulation, which will complete within one second. This simulation involves creating 30 processes using the SJF (shortest-job-first) and FCFS (first-come-first-served) scheduling algorithms.

   - Terminate the simulator using the pink **Quit** button in the lower right-hand corner of the main window.

   - In the **ps** directory you will find the file **psconfig**. This is the configuration file for the process scheduling simulator. The fourth line in this configuration file begins with **user**. Change its default value of **Local User** to your name.

   - Execute the simulator again.

   - Click the **Open Log** button in the top row of the center column of the main window to open a log file. After you have pushed this button, its name will change to **Close Log**.

   - Click the **Run Experiment** button again.

   - Click the **Show All Table Data** button at the right side of the main window. The Table Data window showing a table of statistics appears.

- Click the **Log All Table Data** button in the middle column of the main window. The table of statistics is now sent to the log file.

- Click the **Draw Gantt Chart** button at the right side of the main window. A pop-up window appears to allow you to draw the Gantt Chart for either the SJF or the FCFS scheduling algorithms. Choose the **Gantt Chart for FCFS button** and a Gantt chart window appears illustrating the states of the 30 processes. Click the **Log** button in the lower right-hand corner of the Gantt Chart for FCFS window to send the output of the Gantt Chart to your log file. Close the Gantt Chart window.

- Return to the main window and click the Draw Gantt Chart button again, this time choose the Gantt Chart for SJF option. A Gantt chart illustrating the SJF algorithm of the states of the 30 processes is shown again. Log the Gantt Chart and close the Gantt Chart window.

- Return to the main window and click the Close Log button and then the Quit button. This creates a file called logfile.html in the main ps directory where you started the simulator from.

- Change the name of your logfile.html to your last name, first letter of your first name run1.html.

4. Answer the following questions based on what you just have completed.
   a. Examining the generated table, what is the CPU Utilization for myrun_2?
   b. Examining the generated table, which run has greater Throughput value?
   c. Which scheduling algorithm has the shorter average turnaround time?
   d. Which scheduling algorithm has the longer maximum turnaround time?
   e. Examining the Gantt chart for FCFS at approximately what time did process 15 finish?
   f. Examining the Gantt chart for SJF, which process finished first?
   g. Examining the Gantt chart for SJF, which process finished last?

   In the traditional UNIX scheduling algorithm the load average refers to the average number of processes in the ready queue, and can be calculated by using the generated table. The load average is the total waiting time divided by the total time for the experiment. To determine the total waiting time, multiply the average waiting time by the number of processes. The total time for the experiment is the time the last processes finishes and is available in the table.

   h. What is the load average for FCFS?
   i. What is the load average for SJF?

   The simulator does not take into account the context switch time required by the scheduling algorithm. The context switch time can be taken into account as follows: The number of context switches can be determined from the generated table, under the heading **Entries**, subheading **CPU**. Each time a context switch occurs, the context switch time must be added to the waiting time of each process in the ready queue. The previous two questions required calculating the load average for the FCFS and SJF scheduling algorithms. Assuming the context switch time would add 10% to the average waiting time, the context switch time can be determined as follows:

Let the context switch time be ***S***.
If there are ***N*** context switches and the load average is ***L***, the total waiting time due to context switches is ***N*** × ***S*** × ***L***.
Set ***N*** × ***S*** × ***L*** equal to 10% of the total waiting time and solve for ***S***.

j. What is the context switch time for FCFS?
k. What is the context switch time for SJF?

5. Now that you are familiar with the simulator let's add a few more items.

In the **ps** directory, there are two files: (1) **`myrun.run`** (the *run* file); and (2) **`myexp.exp`** (the *experiment* file.) The run file specifies various parameters such as the number of processes, interarrival times, process duration (how long it requires the CPU), and the length of CPU and I/O bursts. The experiment file specifies the number of experiments that are to be run using the run file. Each experiment in the experiment file runs the scheduler according to the parameters specified in the run file. For example, if the experiment file `myexp.exp` contains the following:

```
name myexp
comment This experiment contains 2 runs
run myrun algorithm FCFS key "FCFS"
run myrun algorithm SJF key "SJF"
```

This means the experiment specifies two runs will be performed, one using FCFS scheduling, the other using SJF scheduling.

If the run file **`myrun.run`** contains:

```
name myrun
comment This run specifies one type of process
algorithm FCFS
seed 5000
numprocs 20
firstarrival 0.0
interarrival constant 0.0
duration constant 100
cpuburst constant 10
ioburst constant 10
basepriority 1.0
```

This means each experiment will run the scheduler with 20 processes, all arriving at time 0. The duration of each process is 100, and the CPU and I/O bursts are both constant at 10.

The attached PDF of the user manual fully explains the parameters for the experiment and run files.

In Chapter 6 of the text the **convoy effect** of FCFS scheduling is described. The following values of myrun.run specify two types of processes:

```
name myrun
comment This run specifies two types of processes
```

```
algorithm FCFS
seed 5000
numprocs 5
firstarrival 0.0
interarrival constant 0.0
duration constant 50
cpuburst uniform 1 5
ioburst constant 10
basepriority 1.0

numprocs 1
firstarrival 0.0
interarrival constant 0.0
duration constant 50
cpuburst constant 25
ioburst uniform 1 5
basepriority 1.0
```

The first type of process specifies 5 processes with a uniform CPU burst uniformly distributed between 1 and 5 and a constant I/O burst of 10. The second type of process is a single process with a constant CPU burst of 25, and a uniform I/O burst distributed between 1 and 5. Thus, there are 5 I/O-bound processes, and 1 CPU-bound process.

Change the value of `myrun.run` as shown above. Run the simulator with this changed parameter, and examine the Gantt chart for the FCFS.

a. What process(es) are part of the convoy the second time process 6 gets the CPU?

Change the value of `myexp.exp` to the following:

```
name myexp
comment This experiment contains 5 runs
run myrun algorithm FCFS key "FCFS"
run myrun algorithm SJF key "SJF"
run myrun algorithm RR 1 key "RR 1"
run myrun algorithm RR 5 key "RR 5"
run myrun algorithm RR 10 key "RR 10"
```

This specifies 5 different runs of the scheduler, FCFS and SJF are the first two. The next three are round-robin (RR) with time quantums of 1, 5, and 10, respectively.

Change the value of `myrun.run` to the following:

```
name myrun
comment This contains one type of process
algorithm SJF
seed 5000
numprocs 20
firstarrival 0.0
interarrival constant 0.0
duration constant 50
```

```
cpuburst uniform 2 20
ioburst constant 10
basepriority 1.0
```

In this run file, there are 20 processes with a constant duration and I/O burst. The CPU burst will uniformly range between 2 and 20.

Run the simulator with this changed parameter, and click the **Show All Table Data** button.

b. Which scheduling algorithm has most context switches?

Draw the Gantt chart for both the FCFS and SJF algorithms. Notice they look quite different. Yet in the table, both FCFS and SJF have the same number of context switches (109)

c. Explain why.

Preemptive SJF (PSJF) scheduling is discussed in Chapter 6. PSJF is the situation whereby, if a newly arrived process has a shorter next CPU burst than the currently executing processes, the newly arriving process will preempt the process currently running. (SJF scheduling has no such preemption.) Add the following line to **`myexp.exp`**:

```
run myrun algorithm PSJF key "PSJF"
```

This adds an experimental run using PSJF scheduling. Run the simulator again, and examine the Table Data.

d. What can be said about the relationship between the number of context switches for the SJF and PSJF scheduling algorithms?
e. For the 3 RR scheduling runs, what can be said about average waiting time with an increasing time quantum?

Chapter 6 states the performance of round-robin scheduling is heavily dependent upon the length of the time quantum. If the time quantum is large, the RR scheduling policy is the same as FCFS policy.

Use the following `myrun.run` as a starting point

```
name myrun
comment This run specifies one type of process
algorithm FCFS
seed 5000
numprocs 20
firstarrival 0.0
interarrival constant 0.0
duration constant 100
cpuburst constant 10
ioburst constant 10
basepriority 1.0
```

Modify `myrun.run` and create/modify `myexp.exp` and add the following

- `run myrun algorithm RR 25 key "RR 25"`
- `run myrun algorithm RR 15 key "RR 15"`
- `run myrun algorithm RR 8 key "RR 8"`

f. Which Round Robin run, from above, was the closet to the FCFS policy?
g. How would `run myrun algorithm RR 100 key "RR 100"` affect the performance of the Round Robin Scheduler? Justify your answer.

Save your answers to the above questions as a PDF named, your last name, first letter of your first name, answers.pdf

TO TURN IN

A zip

- The log file (html file) from question 3
- The PDF of your answers

You will submit a zip file named your last name first letter of your first name lab8.zip (Example steinerslab8.zip)