Operating Systems 200 (Index No. 4542)

CURTIN UNIVERSITY
Department of Computing

CPU Scheduling Simulator

Due Date: **4PM, Monday May 19, 2014**

The objective of this programming assignment is to give you some experiences in using multiple threads and their inter-thread communications, in addition to implementing two known CPU scheduling algorithms. You will learn how to create threads and solve the critical section problems.

1) Write a program in C language that implements the **Round Robin** (**RR**) CPU scheduler. The program waits for an **input file** from the user that contains a list of processes, their **arrival times**, and their **burst times,** produces the **Gantt** chart of the processes, and computes the **average waiting time** and the **average turnaround time** from the chart. While waiting for the input, the program should show a user prompt "RR simulation:". Assume the input file name is no longer than 10 characters, e.g., input1, burst_list, etc.

   The following shows an example of an input file. The example shows that Process 1 arrives at time 1 and has a burst time of size 24 units, Process 2 arrives at time 1 and has 23, Process 3 at 3 and has 1, etc. Notice that the first line shows that the time quantum is 5.

```
5
1       24
1       23
3       1
4       20
4       100

.
.
.

12      30
20      20
```

   The program should print the average waiting time and the average turnaround time, which are calculated from the Gantt chart, and then wait for another user input; the program terminates if the user gives "QUIT" as input.

2) Write another program in C language that implements the **Shortest Job First** (**SJF**) CPU scheduler. The program should follow similar details as described for RR program, but ignore the time quantum.

3) Write a program (the parent thread) that does the following.

a) Create a **thread A** that runs the RR program and a **thread B** that runs the SJF program. Threads A and B block while waiting for input from the parent thread.

b) The parent thread waits for an input from the user. The input can be either a file name, e.g., input1, or "QUIT"

c) For each file name, e.g., input1, the program does the following:

- The parent thread stores the input, e.g., input1, in a buffer, called **buffer1**, and signals thread A and B to read the file. The parent thread then blocks while waiting for results from threads A and B, i.e., the average turnaround time, and the average waiting time. Notice that the parent thread acts as a producer and threads A and B as the consumers for a bounded buffer of size 1; buffer1 can store only one file name at a time.

- Each child thread that has computed the average turnaround time and the average waiting time writes the result in another buffer, called **buffer2**, and signals the parent thread to read the result. Then the child thread blocks waiting for another input from the parent thread. For this case, the parent thread acts as a consumer and threads A and B as the producers for a bounded buffer of size 1, i.e., buffer2 can store only one pair of the average times at a time.

- After the parent thread receives the results from both threads, it should print the results, e.g.,

  For input1:
  RR: the average turnaround time=5, the average waiting time=4
  SJF: the average turnaround time=4, the average waiting time=3.5

d) When the parent process receives "QUIT" as input, it tells threads A and B to terminate, and terminates itself.

e) The simulator must solve any possible synchronization/critical section issues among the threads. You have to describe the possible issues and how you solve the issues.

f) Use pthread_create(), pthread_mutex_lock(), pthread_mutex_unlock(), pthread_cond_wait(), and pthread_mutex_signal() in your program.

4) You MAY make your own assumptions/requirements other than those already given. However, YOU HAVE TO DOCUMENT ANY ADDITIONAL ASSUMPTIONS/LIMITATIONS FOR YOUR IMPLEMENTATIONS.

Instruction for submission

1. Put the completed program files i.e., simulator.c, makefile, and other files, in your home directory, under a directory named **OS200/assignment**.

2. Assignment submission is **compulsory**. NO LATE SUBMISSION WILL BE ACCEPTED. If your assignment is incomplete due to illness or other circumstances on or near the submission date you must submit the partial solution you have completed. Any justified exceptional circumstance (e.g., due to illness supported with a Medical Certificate) will be taken into consideration.

3. You must submit the software solution of your assignment (source code and readme file that, among others, explains how to compile your program and how to run the program), and an assignment report. The report can be submitted into the unit box, while the software and the soft copy of the report should be submitted to the unit Blackboard.

4. Your assignment report should include:

   - A cover page that includes the words "Operating Systems 200 Assignment", and your name in the form: family, other names. Your name should be as recorded in the student database.

   - You have to describe/discuss in detail how any mutual exclusion is achieved and what threads access the shared resources.

   - Description for any cases for which your program is not working correctly or how you test your program that make you believe it works perfectly.

   - Sample inputs and outputs from your running programs.

   - A printout of all source code for the programs with proper in-line and header documentation. Use proper indentation so that your code can be easily read.

5. Due dates and other arrangements may only be altered with the consent of the majority of the students enrolled in the unit and with the consent of the lecturer.

6. Demo requirements:

   - You may be required to demonstrate your program and/or sit a quiz during tutorial sessions (to be announced).

   - For demo, you MUST keep the source code of your programs in your home directory, and the source code MUST be that submitted. The programs should run on any machine in the department labs.

Failure to meet these requirements may result in the assignment not being marked.