COMP2006 - Operating Systems

CURTIN UNIVERSITY
Discipline of Computing

CPU Scheduling Simulator

Due Date: **4PM, Monday May 10, 2021**

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.

The following shows an example of an input file; numbers in each line is separated by a space. The example shows that Task 1 arrives at time 0, has a burst time of size 24 units and priority 1, Task 2 arrives at time 1, has a burst time of 23 and priority 2, etc. A task with a lower priority number has higher priority than a task with a higher priority number.

```
0 24 1
1 23 2
3 1 2
4 20 3
4 100 1

.
.
.

12 30 10
20 20 6
```

1) **(15 marks).** Write a program in C language that implements the **Priority Preemptive** (**PP**) CPU scheduler. The program waits for an **input file** from the user that contains a list of tasks, their **arrival times**, their **burst times,** and their **priority,** produces the **Gantt** chart of the tasks, 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 "PP simulation:". Assume the input file name is no longer than 10 characters, e.g., input1, burst_list, etc.

   The program should show a Gantt chart for the schedule, 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. You can present your Gantt in any way; however, it must be neat and readable / understandable.

2) **(15 marks).** Write another program in C language that implements the **Shortest Remaining Time First** (**SRTF**) CPU scheduler. The program should follow similar details as described for PP program.

3) **(Total: 50 marks).** Write a program (the parent thread) that does the following.

   a) Create a **thread A** that runs the PP program and a **thread B** that runs the SRTF 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 shows a Gantt chart for the schedule, 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.

        For example, for input1:
        PP: <Show the Gantt chart here>

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

        For example, for input1:
        PP: the average turnaround time = 4, the average waiting time = 3.5
        SRTF: 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. Each terminating thread print a message on the screen.
   For example:
   PP: terminate.

   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_cond_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. Assignment submission is **compulsory**. Students will be penalized by a deduction of ten percent per calendar day for a late submission. **An assessment more than seven calendar days overdue will not be marked and will receive a mark of 0**.

2. You must (i) submit the soft copy of the report to the unit Blackboard (**in one zip file**), and (ii) put your program files i.e., PP.c, SRTF.c, simulation.c, makefile, and other files, e.g., **sim_input** and **sim_out**, in your home directory named **OS/assignment**.

3. Your assignment report should include:

   - A signed cover page that explicitly states the submitted assignment is your own work. The cover includes the words "Operating Systems Assignment", and your name in the form: family, other names. Your name should be as recorded in the student database.

   - Software solution of your assignment that includes (i) all source code for the programs with proper in-line and header documentation. Use proper indentation so that your code can be easily read. Make sure that you use meaningful variable names, and delete all unnecessary comments that you created while debugging your program; and (ii) readme file that, among others, explains **how to compile your program and how to run the program**.

   - Detailed discussion on how any mutual exclusion is achieved and what processes / threads access the shared resources.

   - Description of 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.

Your report will be assessed (worth 20% of the overall assignment mark).

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