CS 3305A: Operating Systems
Department of Computer Science
Western University
Assignment 4
Fall 2020
Due Date: November 9th 2020

Purpose:

The goals of this assignment are the following:

- Gain more experience with the C programming language from an OS's *CPU scheduling* perspective.
- Get hands-on experience with the CPU *scheduling algorithms.*

Performance Evaluation of CPU Scheduling Algorithms

You will be applying *CPU Scheduling Algorithms* in the C programming language. A sample input file is provided with skeleton code, which must be used to develop the *CPU Scheduling Algorithm, **Round Robin*** **(RR)**.

Skeleton Code:

The provided skeleton code performs the following:

1. Creates the number of ready queues as stated in the input file and assigns a time quantum for each queue
2. Creates all the processes for each of the ready queues based on the input file specifications (such as CPU burst time, arrival order, etc.).
3. Once the simulation of all the ready queues is complete, the program will output results to a text file "cpu_scheduling_output_file.txt".

What you need to do:

The CPU scheduling algorithm RR must be applied on each ready queue. Ready queues should be simulated in the order of their queue number (q 1, q 2, etc.). You are required to implement the "rr" function in the file **scheduling.c**. If necessary, you may implement helper functions in the **scheduling.c** file to complete the assignment.

Function rr:

```
rr_result *rr (int *queue, int np, int tq);
```

Function rr **parameters**:

- queue: 1D int array containing the CPU burst time for every process in a ready queue
- np: number of processes (length of `queue` array)
- tq: time quantum for the RR algorithm

Function rr **return**:

rr_result struct pointer, where:

- rr_result -> order: 1D int array, containing the execution order of processes (i.e. `(rr_result -> order)[0]` is the process number of the first process)
  ** hint: The first process, p1, is always the first item in the order of selection for round-robin and FCFS
- rr_result -> order_n: int, representing the size of the `order` array, described above
- rr_result -> turnarounds: 1D int array, containing the turnaround time for each process (i.e. `(rr_result -> turnarounds)[0]` is the turnaround time for p1)

Compiling and Running

To compile, run, and test the program, a Makefile is provided with the skeleton code. If you are not familiar with Makefiles, consult this quick tutorial.

- To compile the program: from GAUL, run **make** (short for *make build*)
- To run the program after compiling: run ./a4
- To automatically test the program against expected output from the provided sample input file, run: **make test** after compiling
- To remove the output files generated, and the compiled executable, run: **make clean**

Important Notes

- You may **not** change the function signature for any existing functions in the provided skeleton code.
- You may **not** create new code files – any helper methods you may choose to write must be inside the **scheduling.c** file.
- Check for any memory leaks and make sure you free any dynamically allocated memory.
- Compiling your code should not give **any** warnings (i.e. running make or make test) or this may result in losing marks.
- Your code should be concisely written (if you're doing something unconventional, it might help to comment it). Should any test cases fail, you would lose marks if the code is not concise (i.e. meaningful variable names, clear comments, proper spacing and indentation).

Input File

Symbols used in the input file:

q: Ready queue
tq: time quantum

For example:

q 1 tq 4 p1 10 p2 5 p3 7 p4 20 p5 17 p6 9 p7 3 p8 11 p9 15 p10 1

In this example, ready queue 1 has a total of ten processes namely p1, p2, p3, p4, p5, p6, p7, p8, p9, and p10. The sequence of these processes represents their arrival order. For example, p1 arrives first and p10 arrives last in this list of processes. In the "px y" format, y refers to the CPU burst time for px. A time quantum of 4 is assigned to each process.

Computing Platform for Assignments

You are responsible for ensuring that your program compiles and runs without error on the computing platform mentioned on below. **Marks will be deducted** if your program fails to compile or your program runs into errors on the specified computing platform (see below).

- Students have virtual access to the MC 244 lab, which contains 30 Fedora 28 systems. Linux machines available to you are: **linux01.gaul.csd.uwo.ca** through **linux30.gaul.csd.uwo.ca**.
- It is your responsibility to ensure that your code compiles and runs on the above systems. You can SSH into MC 244 machines.
- If you are off campus, you have to SSH to **compute.gaul.csd.uwo.ca** first (this server is also known as sylvia.gaul.csd.uwo.ca, in honour of Dr. Sylvia Osborn), and then to one of the MC 244 systems **(linux01.gaul.csd.uwo.ca** through **linux30.gaul.csd.uwo.ca**).
- https://wiki.sci.uwo.ca/sts/computer-science/gaul

Assignment Submission

You must submit your Assignment through OWL. Be sure to test your code on one of MC 244 systems (see "Computing Platform for Assignments" section above). **Marks will be deducted** if your program fails to compile or your program runs into errors on the computing platform mentioned above.

Assignment 4 FAQ will be made available on OWL. Also, consult TAs, and the Instructor for any question you may have regarding this assignment.