CIS 3110: Operating Systems

Assignment 2: CPU Scheduling

Due Monday, Feb. 24, 2020 @ 11:59pm

1. Objective

The objective of this assignment is to implement a simplified Dispatcher for an OS as a C function (See **Implementation** at the end of this document for details). It will simulate dispatching processes using one of two dispatching policies, and change the states of processes admitted to the system accordingly, as described in the lecture notes and textbook.

2. How to decide what is your policy

If you are in **Class session 1** (including sections 0101, 0102 and 0103) then you will implement **Round Robin**. If you are in **Class session 2** (including sections 0204, 0205 and 0206) then you will implement **Shortest Remaining Time**.

3. Input

The function takes 3 parameters for the Round Robin Policy:

- a file descriptor of the file containing the simulation input
- a positive integer representing the CPU quantum
- a positive integer representing how long does it take to the hard drive to process requests

The function takes 2 parameters for the Shortest Remaining Time Policy:

- a file descriptor of the file containing the simulation input
- a positive integer representing how long does it take to the hard drive to process requests

Also, events will be simulated. The sequence of events will be given in the standard input (one event per line). Empty line will signify the end of the input. In other words, you can also save the events into a file and then use input redirection to read the simulation input, which consists of several lines (each representing a process that wants to run) formatted as follows:

<start_time> <process_id> <run_time> <exchange_time_1> ... <exchange_time_n>

where

- <start_time> is a non negative integer number denoting the system time in milliseconds that a process starts at.

- <process_id> is a unique non negative integer number denoting the process id. The System idle process has a process id = 0
- <run_time> is a non negative integer number denoting how long this process wants to run for in milliseconds
- <exchange_time_i> is a non negative integer number denoting the process local time at which it requests an exchange with the hard drive.

For example the input line

1000 12 5000 500 2500 3000 4500

means, that 1000 milliseconds after system start-up the process with id 12 is admitted to the system. This process requires 5000 milliseconds of CPU time, and will request 4 exchanges with the hard drive: after it used 500, 2500, 3000 and 4500 milliseconds of CPU.

The input line

2000 14 3000 1500

means, that 2000 milliseconds after system start-up the process with id 14 is admitted to the system. This process requires 3000 milliseconds of CPU time, and will request one exchange with the hard drive after it used 1500 milliseconds of CPU.

An empty line signifies the end of input. Anything following this line is ignored.

4. Output

When the simulation is finished your function will print out statistics for each process with the following format, starting with process 0 followed by each remaining process in the order that they finished running.

<process_id> <total running time> <total ready time> <total blocked time>

where output fields are separated by a single whitespace.

Please note that Process 0 only prints out total running time. All values are integers.

5. Assumptions

Your Dispatcher will have to keep track of events and changes in the state of the processes, taking into account the following assumptions:

- All input values are correct
- Start times increase as input lines increase
- Process 0 starts running at time 0
- Hard drive requests are served in FIFO (first in first out) order
- Each hard drive request takes a positive integer (> 0) number of milliseconds to complete

- Hard drive exchange #s increase from left to right on input lines
- If there are no ready user processes then process 0 process is running
- If process 0 is running and a new process is created, or a blocked process becomes unblocked, this process gets the CPU immediately
- If a process is unblocked at the same time another is preempted the unblocked process enters the ready queue first
- If a process is unblocked or preempted at the same time another one is admitted to the system the unblocked/preempted processes enters the ready queue first
- If a process requests an exchange with the hard drive at the same time its CPU quantum expires the process is blocked.

6. Implementation

You will be given the following files:

- makefile
- main.c
- dispatcher.h
- dispatcher.c

In addition, you will have the followings

- how_to_run.txt: It shows you how to compile and run the program
- rr_test.sh for Round Robin (or srt_test.sh for Shortest Remaining Time): an automation test script for you to test your program, which will be detailed later.
- public_test_inputs.zip: It contains the input for different public tests. **All the included test input files must be placed in a "test_inputs" sub-folder under the folder where your executable Dispatcher as well as the test script (rr_test.sh or srt_test.sh) are**

1) expected_results_rr.zip for Round Robin (or expected_results_srt.zip for Shortest Remaining Time): It contains the expected output for the corresponding public tests. **All the included expected test output files must be placed in a "rr_test_outputs" sub-folder for Round Robin (or "srt_test_outputs" for Shortest Remaining Time) under the folder where your executable Dispatcher as well as the test script (rr_test.sh or srt_test.sh) are**

Do not edit any of the files except for dispatcher.c where your code goes and makefile if needed. The marker should be able to use an entirely different main.c and get the same results as you. You are encouraged to add additional .c and .h files as you desire to implement and useful data structures as long as your code still compiles using the makefile. If you reuse code from previous years you must make this obvious in the header of these files. Additional recommendations are given in a2_DataStructures_and_other.pdf on Courselink.

Besides the functionalities required in this assignment, an emphasis should also be placed on writing concise easy to read code. Please refer to the coding style guidelines (e.g., C Programming Style Guide[1]) to provide direction regarding the format of the code you write. As the program grows, you may want to improve the structure of the code by moving certain

parts into separate functions.

7. How to Test Your Program

We provide you with several basic test cases, in file testX.in (as the input) and its expected output rr_resultsX.txt if using Round Robin Policy (or srt_resultsX.txt if using Shortest Remaining Time policy), where X is a test case number.

Assume that the resulted executable file after compiling your dispatcher C program is called Dispatcher. For example, you can use the test case 1, which is one test case provided

For Round Robin Policy:
Assuming your code compiled to test it, run
./Dispatcher quantum_# harddrive_request_time < test_input_file.txt
in command line window from inside the same folder. For example
./Dispatcher 500 800 < test_inputs/test1.txt

For Shortest Remaining Time Policy:
Assuming your code compiled to test it, run
./Dispatcher harddrive_request_time < test_input_file.txt
in command line window from inside the same folder. For example
./Dispatcher 800 < test_inputs/test1.txt

Afterwards, you can check the expected result, rr_results1.txt, against the actual result from your program. If you implemented everything correctly the output will match the expected result provided.

$ cat ./rr_test_outputs/rr_results1.txt //./srt_test_outputs/srt_results1.txt if using SRT policy
0 100
12 800 200 800
13 800 700 0
11 1000 300 1100

Sample Input & Output for the above test
100 12 800 500
300 11 1000 500
700 13 800

With CPU quantum 500ms and hard drive exchange time 800ms Round Robin Output
0 100
12 800 200 800
13 800 700 0
11 1000 300 1100

With hard drive exchange time 800ms Shortest Remaining Time Output
0 900
13 800 0 0
12 800 100 800
11 1000 1400 800

Also, **an automation testing tool is provided to facilitate a faster testing of all cases**. To facilitate

your testing process, a shell script for testing your code named rr_test.sh for Round Robin (or srt_test.sh for Shortest Remaining Time) has been provided. Make sure that you generate your executable Dispatcher in the same folder where the shell script is. For ease of presentation, we then present an example of Round Robin policy illustrating its use. Note that you also need to set up proper permissions on the above testing shell script before you execute the script, for example,

```
chmod +x rr_test.sh
```

Please make sure that two subfolders exist, the "test_inputs" sub-folder contains the input for different public tests and the "rr_test_outputs" sub-folder for RR policy (or the "srt_test_outputs" sub-folder for SRT policy) contains the expected output for the corresponding tests.

Then, enter

```
./rr_test.sh
```

The following shows example execution outputs when using the above testing script

```
oscreader@OSC:~$ ./rr_test.sh
Start public testing ...
Test 1 passed
Test 2 passed
Test 3 passed
Test 4 passed
Test 5 passed
Test 6 passed
Test 7 passed
Test 8 passed
Test 9 passed
Test 10 passed
Public testing is done!
```

Which means that you pass all the public test cases.

8. Marking

Your code will be tested against several test cases. Some of these will be provided on Courselink prior to the due date and some will not. The expectation is that if your code passes the provided tests cases it should pass the additional ones, so if you only pass the provided test cases you will pass the assignment (although not with flying colors). Expect the additional tests cases to cover less obvious edge cases.

Grading scheme

Program compiles, runs	2
Code quality (e.g., CODE Style & Documentation)	3
Public tests	50
Private tests	40

Stability and reliability testing (e.g., not have memory leaks or memory violations) 5

9. Submission

- If you have any problems in the development of your programs, contact the teaching assistant (TA) assigned to this course.
- You are encouraged to discuss this project with fellow students. However, you are **not** allowed to share code with any student.
- If your TA is unable to run/test your program, you should present a demo arranged by your TA's request.
- Please submit 1 (one) zip file to the Courselink dropbox.
- How to name your programming assignment projects: For any assignment, zip all the files required to a zip file with name: cis3110_a2_XXX.zip, where XXX is your University of Guelph's email ID (Central Login ID). This naming convention facilitates the tasks of marking for the instructor and course TAs. It also helps you in organizing your course work. Failure to follow the requirements will result in mark reduction.

  Note: to zip and unzip files in Unix:

  ```
  $zip -r filename.zip files
  $unzip filename.zip
  ```

References:
[1] C Programming Style Guide
http://faculty.cs.tamu.edu/welch/teaching/cstyle.html