Sabanci University

Faculty of engineering and Natural Sciences

CS 300 Data Structures Homework 1

Assigned: **Oct 22, 2021** Due: **Nov 2, 2021 @ 11:55 pm**

PLEASE NOTE

SOLUTIONS HAVE TO BE YOUR OWN. NO COLLABORATION OR COOPERATION AMONG STUDENTS IS PERMITTED.

10% PENALTY WILL BE INCURRED FOR EACH DAY OF OVERTIME. SUBMISSIONS THAT ARE LATE MORE THAN 3 DAYS WILL NOT GET ANY CREDITS.

SUBMISSIONS WILL BE MADE TO THE SUCOURSE SERVER. NO OTHER METHOD OF SUBMISSION WILL BE ACCEPTED.

Introduction

In this homework assignment, you are required to implement a **Multi-Level Feedback Queue (MLFQ)** (Figure 8.1, taken from ostep.org). MLFQ is used in operating systems (OS) as a part of the scheduler, which stores processes and determines the order in which the processes need to be executed. For this purpose, you need to implement the algorithm provided below, the details of which are extensively discussed in the attached document (also taken from ostep.org).

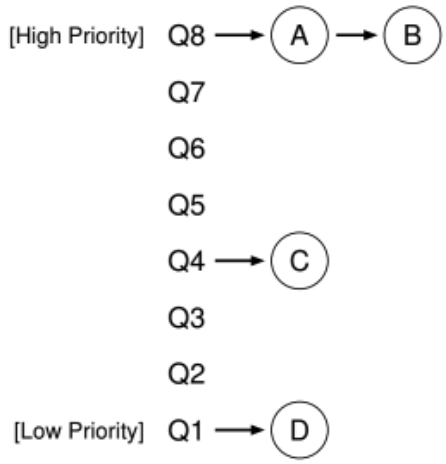

Figure 8.1: **MLFQ Example**

Before going into more detail you need first to get yourselves familiar with the following concepts:

What is an OS process?

An OS process (for short, process) is a **computer program** that is executed by one or more threads. So your browser, note-taking app, or the program you compile and run are all processes in the eyes of the operating system.

How do processes use the CPU?

There may be dozens of processes running **concurrently** in modern computers. Given a CPU core, the processes take turns to get the control of the CPU, so that they can execute. A short time frame, during which the control of the CPU is given to a process, is often referred to as a *time slice*. In this context, given a time slice, you may assume that only one process is executing on a CPU core.

What is a scheduling algorithm?

A scheduling algorithm determines which CPU will be assigned to which process at a given point of time. The order of execution as well as the amount of time reserved for each turn are instrumental in having a high-performing system. Thus, the development of “optimal” schedulers is of great practical importance.

One algorithm to tackle this problem was proposed by Cobarto et al. in 1962, which is called **Multilevel Feedback Queue (MLFQ).** The details of this algorithm are given in the attached document. In this homework, you will implement this algorithm by using queues.

Implementation Details

At a very high level, the MLFQ algorithm you will implement for this homework should follow the following rules, where P1 and P2 are processes:

1. **Rule 1**. If Priority(P1) > Priority(P2), P1 runs P2 does not.
2. **Rule 2.** If Priority(P1) == Priority(P2), P1 and P2 run in a round-robin fashion.
3. **Rule 3.** When a process enters the system, it is placed at the highest priority (the topmost queue).
4. **Rule 4a.** If a process uses the entire time slice while running, its priority is reduced by one.

5. **Rule 4b.** If a process gives up the CPU before its time slice is completed, it stays at the same priority level.
6. **Rule 5.** After some time period *S*, all the processes are moved to the topmost queue.

Note that these rules are exactly the same rules discussed in the document attached. Note further that, for this homework, you are not going to implement **Rule 4** discussed in Section 8.4 of the attached document.

Input & Output Format

Your program needs to read the inputs from a number of text files.

Configuration File

This file, named **configuration.txt**, has three lines, which indicate the number of queues, the number of processes, and the value of *S*, respectively. Note that, for this homework, S indicates the number of time slices that need to be passed before all the processes are moved to the top priority queue (see **Rule 5** above). An example configuration file is given below:

configuration.txt

```
3    // number of queues
4    // number of processes
5    // size of S
```

Process File

In addition to the configuration file discussed above you will receive a separate file for each process, the name of which will be **p<process ID>.txt**. The process IDs, as is the case for all indices used in this document, are a consecutive sequence of integers starting from 1. That is, the ID of the first process is 1, that of the second process is 2, etc. In the remainder of the document, these files will be referred to as process files.

The process file of a process specifies what the process does in each time slice. More specifically, in a process file, each line corresponds to a time slice and the value at the line depicts what the process does in the time slice. There are three values that can be placed at a line: 0, 1, and '-'. 0 indicates that the process gives up the CPU before its time slice is completed, such that **Rule 4b** applies. 1 indicates that the process uses the entire time slice

while running, such that **Rule 4a** applies. And, '-' depicts that the process has terminated in the respective time slice. Once a process terminates it should be removed from the MLFQ structure.

Below is an example process file given for a process with ID = 1. It turns out that the process uses the entire time slice for the first time slice, but only a portion of it in the subsequent 3 time slices. And, after entirely using the 5th and 6th time slices, it terminates.

p1.txt

```
1
0
0
0
1
1
-
```

Scheduling

You need to implement the MLFQ scheduling algorithm. It will be guaranteed that none of the process files start with '-'. Therefore, to initialize your data structures you need to insert all the processes into the top priority queue in the increasing order of their IDs. Then, you schedule and execute them time slice by time slice as explained in the document attached, by assuming that only one CPU core is available for the processes. Needless to say, to figure out what the current process does in the current time slice you need to read the current line from the respective process file.

Output File

As the output we expect a trace that contains what has happened in each time slice; what type of operation is carried out, which process was involved, and what was the destination (i.e., priority queue) of the process involved, respectively. The output should be written to a file, named **output.txt**.

In the output, the symbol *B* refers to the operations done when *S* is reached (**hint:** operations marked as *B* will always end up at the topmost queue). And, the symbol *E* indicates the termination of a process, in which case the destination queue will be indicated by QX. An example output file is given below.

output.txt

```
1, P1, Q3
0, P2, Q2
B, P1, Q4
B, P2, Q4
B, P3, Q4
0, P1, Q4
1, P2, Q3
E, P3, QX
```

Frequently Asked Questions

Q: How should queues be named?
A: The name of each queue should be in the form of Q<queue ID>. The queue IDs are consecutive numbers starting from 1 and the level of priority increases as the queue ID increases. For example, in the presence of 7 queues, the top and least priority queues will be referred to as Q7 and Q1, respectively.

Q: Can I use other kinds of data structures in the implementation of MLFQs?
A: You must use queues to implement MLFQs.

Q: What happens when the highest priority queue is empty and *S* has not reached yet?
A: Queues are traversed starting from the highest priority going towards the lowest priority. For example, if Q7 is empty then the Q6 needs to be processed.

Sample Run

Please enter the process folder name: `sample_run_1`
When all processes are completed, you can find execution sequence in "sample_run_1/output.txt".

Given Files for the Sample Run:

sample_run_1/configuration.txt:

```
4
5
7
```

sample_run_1/p1.txt:

```
1
0
1
1
1
-
```

sample_run_1/p2.txt:

```
1
1
0
1
-
```

sample_run_1/p3.txt:

```
1
0
0
1
1
1
1
0
-
```

sample_run_1/p4.txt:

```
1
0
1
-
```

sample_run_1/p5.txt:

```
1
1
1
0
-
```

Output File for the Sample Run:

sample_run_1/output.txt:

```
1, PC1, Q3
```

```
1, PC2, Q3
1, PC3, Q3
1, PC4, Q3
1, PC5, Q3
0, PC1, Q3
1, PC2, Q2
B, PC3, Q4
B, PC4, Q4
B, PC5, Q4
B, PC1, Q4
B, PC2, Q4
0, PC3, Q4
0, PC4, Q4
1, PC5, Q3
1, PC1, Q3
0, PC2, Q4
0, PC3, Q4
E, PC4, QX
B, PC5, Q4
B, PC1, Q4
E, PC2, QX
1, PC3, Q3
1, PC5, Q3
1, PC1, Q3
1, PC3, Q2
E, PC5, QX
E, PC1, QX
B, PC3, Q4
1, PC3, Q3
1, PC3, Q2
E, PC3, QX
```

Note that you are given each set of sample files as a folder, you should move all files to the project document as a folder.

Your homeworks will be graded in an automatized fashion, thus:
!!! Your output format should exactly match the output format provided as a sample !!!
!!! Your program should read and write the inputs and outputs (folder paths and file names) as provided in the sample run !!!

Submission

Your code should be submitted to SUCourse+ at the deadline given on the first page. You should follow the following steps:

- Name the folder containing your source files as *XXXX-NameLastname* where *XXXX* is your student number. Make sure you do NOT use any Turkish characters in the folder name. You should remove any folders containing executables (Debug or Release), since they take up too much space.
- Compress your folder to a compressed file named, for example, *5432-AliMehmetoglu.zip*. After you compress, please make sure it decompresses properly and reproduces your folder exactly.
- You then submit this compressed file in accordance with the deadlines above. Your homework will be graded in the following way:
    - If your program does not construct and query quadtrees as described in the attached document, you will get 0 points. This will be the case even if your program works correctly otherwise.
    - We will run about 10 tests on your homework. Each correct test will earn you 10 points. Note that your outputs should be in the exact same format we described above.

Good luck