Project I: Creating A Job Round-Robin Scheduler Using Linked Lists

September 28, 2021

DEADLINE: Friday Oct 15, 2021, 11:30pm

CIIC4020-ICOM4035: Data Structures
Fall 2021

1 Overview

1. **Part I:** Create threads to print a message
2. **Part II:** Create a Round-Robin Scheduler
   (a) create circular linked list (CLL)
   (b) Make threads change the `processing_flag`
   (c) Make main process change the `processing_flag`
   (d) Make the process end when main decides to end

2 Description

2.1 Problem Statement

You are required to build a simple round-robin scheduler using *Circular Linked List (CLL)*. You will have to simulate a system where:

- A `main-process` is processing jobs submitted to the Round Robin scheduler.
- `worker-processes` are submitting jobs to the Round Robin scheduler.

2.2 What is a Round Robin scheduler?

A round robin is an arrangement of choosing all elements in a group equally in some rational order, usually from the top to the bottom of a list and then starting again at the top of the list and so on. A simple way to think of round robin is that it is about *"taking turns"*. Used as an adjective, round robin becomes *"round-robin"*.

In computer operation, one method of having different program process take turns using the resources of the computer is to limit each process to a certain short time period, then suspending that process to give another process a turn (or *"time-slice"*). This is often described as round-robin process scheduling [1]. However, in our simulation, our RoundRobin scheduler will not be perfect: it will not assign time-slices to the job. Our scheduler will only run over the jobs one by one to finish one at a time and then go to the next one to finish it and so on.

[1] https://whatis.techtarget.com/definition/round-robin

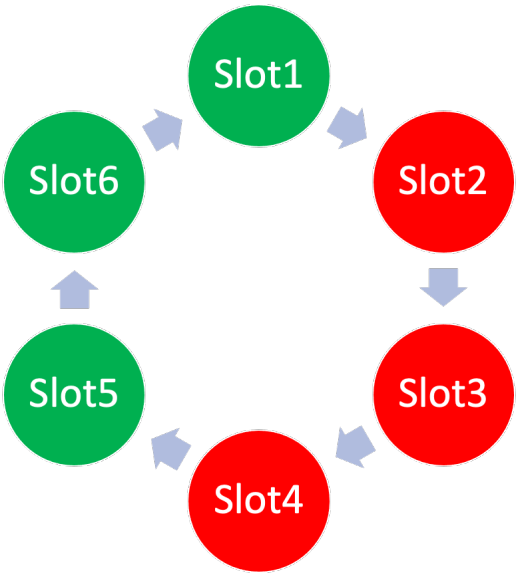

Figure 1. Round Robin Scheduler Schematic

2.3 Project's Scheduler

To simulate the utilization of the scheduler, a dummy job-generator is utilized to create a number of threads. These threads will randomly wait for some time to submit a job to the scheduler (using a random number generator to wait an amount of time equal to the randomly generated time). Each thread will check the linked list's elements one by one and when they find an element that is processed it will change its status to unprocessed. On the other side, the program's main processes will keep checking the linked list elements to do the opposite: it will changes the elements' status from unprocessed to processed. You have to bear in mind that the program's main processes and threads are independent processes which work in parallel. The only mean of communication between them is the CLL (and the termination signal provided in the supplied material, which you will not have to worry about its implementation).

3 Supplied Material

A GitHub repository is created for the project. You will recieve an invitation to accept adding a GitHub repository on your account. Please accept the invitation and then `pull` the project to your machine and start working on it. When you want save your work, `add` and `commit` the changes, and `push` it to the repository. Only pushed work to the repository will be assessed. The repository contains code for:

- **`RRScheduler` class:** this class contains the `main` method and will initiate the Round-Robin CLL, the `threads` (only for Part II... see Section 4.2 for details), and will act as the `main` processes in the scheduler simulator.
- **`RoundRobinCLL` class:** this class defines the CLL and its methods (Used only for Part II... see Section 4.2 for details).
- **`ThreadRunnable` class:** this class is used to create objects which will contain the methods that the threads will run. This class is created for you and can use it as it is (Used only for Part II... see Section 4.2 for details).
- **`Threads` class:** this class will create an object that will contain an `ArrayList` to hold the created threads. This class is created for you and can use it as it is
- A command line argument parser is created for you. The program will take the accept the parameters:

1. `-p` number of thread you want to create. If not passed to the program, a default value of 5 is used.
2. `-t` the number of iterations the `main` processes over perform over the CLL. If not passed, default value of 100 is used.
3. `-s` the part of the project you are running: *i)* 1 for Part I, and *ii)* 2 for Part II.

– **a part1.txt file:** This file is used to report the answer for Part I (see Section 4.2 for details).

4 Deliverable

You can build the project by typing in the terminal:
`gradle build`

4.1 Part I (5%)

Part I is very simple: all you have to do is to run the program using this command line arguments:
`java prj01/RRScheduler -p <number_of_threads> -t <number_of_iterations> -s 1`

then answer this question: **Q: What can you observe from the printouts of the threads on the terminal?**

then put your answer in the **part1.txt** file in provided in the repository and make a push to the GitHub repository.

4.2 Part II (95%)

from inside `prj01_template` folder.
You can run the code for Part II typing in the terminal:
`java prj01/RRScheduler -p <number_of_threads> -t <number_of_iterations> -s 2`

Implementation Details The list will have elements that contain attributes that offer information about the status of the elements: whether it was processed or not. The simulator program will have a main process and other threads, which the main process will also spawn. The threads are worker processes which process in parallel along with the main process. The main process and the worker process will access the linked list simultaneously in a round fashion: i.e.

– starting from element 1, the process's header will visit one element after the other.
– The main process will look for slots that are marked `processed` to mark them as `not_processed`.
– The main process should not stop to start from the beginning of the linked list, it should keep looping the circular linked list indefinitely (till a predetermined condition is met).
– The worker processes will look for slots that are marked `not_processed` to mark them as processed.
– The worker processes will always start from the beginning of the linked list to find an empty slot to fill it (change it from `processed` to `not_processed`).

You will:

– **Complete the code in the `RoundRobinCLL.java` class marked with comments to make this scheduler work. (80%)**
– **Add comments to document ALL the `.java` files using the Javadoc standards. (15%)**

After you are done with your solution, commit the changes and push it to the repository.

4.3 Add Your Name (if missing: -10%)

Add your First and Last names and your email in a comment at the top of **ALL** the files you will push as your final submission using this exact format:

```
// FirstName LastName username@uprm.edu
```

5 Notes

- Read the instructions carefully.. and more importantly, read the code carefully.
- Start early to give yourself time to understand the problem and finish the project in time.
- You are advised to read the `Threads.java` and `ThreadRunnable.java` to understand threads.
- You are invited to give critique about the `Threads.java` implementation and any other part of the code you think it can be improved.
- **This is a project... treat it as a challenge and try not to look for easy answers... search for solution for the problems you face in manuals, documentations, forums, books, and make asking someone's help as your lost resort.**
- **MAKE SURE TO SUBMIT THE PROJECT PARTS BEFORE THEIR DEADLINES... NO EXTENSIONS... NO GROUP WORKING.**