CSCI320 Project II - Specification

Process Manager Implementation

(Document date: 10/22/2018)
(Implementation Tips being added)

Introduction

By September 20, 2018, you are all expected to be familiar with how thread pools and process managers work in Operating Systems. In this project, you are required to implement process manager simulation using Java 8 or C++. You are free to use the one that you are most comfortable. The process manager you are going to implement will choose what process will be run on the CPU next with (1) Priority Scheduling, (2) FCFS, (3) Shortest-Job-First, (4) Round Robin. Implementation of each of these algorithms will get you 2 points (8 points in total). This system will take a folder location as an input and sort the contents of the files.

Details

In the input folder, there will be 101 text files with .txt file extension. This time, there are two types of files: config and content.

Config file: There will only be 1 config file named config.txt.

- The first line of the file will have only one integer. It represents the default length of the round robin algorithm timer in milliseconds.
- Starting from the second line, there will be 100 entries in each line, structured as "*taskID,priority*" (without the quotes), in the order of intended arrival of sorting tasks.
- You can assume that the tasks arrive at the same time with this given arrival order.
- The priorities will be given as -20 to 20, -20 being the lowest priority, and 20 being the highest priority.
- In the case that two tasks have the same priority, the taskID with the lower value should be given priority. Conversely, Round Robin should be executed starting from task 00.txt to 99.txt.
- TaskID corresponds to the integer equivalent of the file name.

Content files: The naming convention for the content files is 00.txt - 01.txt - … - 99.txt. These files include an integer value stored at every line without any punctuation. Note that these files can be of different sizes. You can determine the task length based on the length of items in the files.

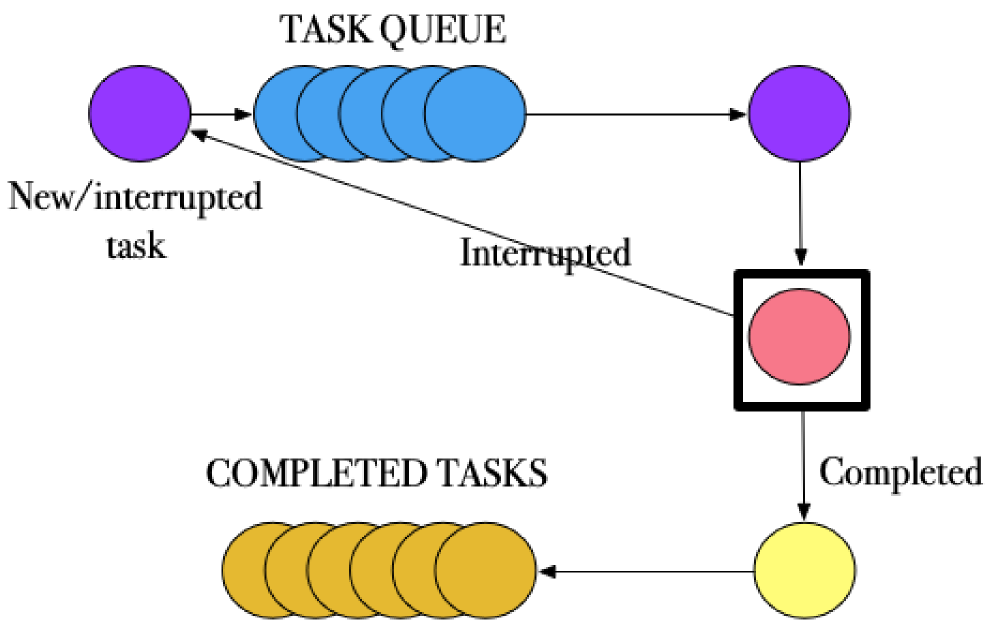

Figure 1: Process Manager Logic

We want you to implement your own Process Manager. The basic functionality of this process manager is given in Figure 1, and can be described as follows.

- There is a task queue (the type of queue depends on the algorithm)
- There can only be one process running on the CPU
- Every process performs a sort operation which consists of 3 phases: Read File, Sort, Write File
- When the process is complete and needs to write to the file, it creates a new file (it does not overwrite the file). The output file name should be constructed as originalFileName + "_" + "algorithmID" + ".txt". For example, for file "00.txt", the output for FCFS should be "00_2.txt"
- When a process starts, it should print out the status as follows:
  `System.out.println(fileName + " just started running");`
- When a process terminates, it should print out the status as follows:
  `System.out.println(fileName + " just completed running");`
- When a process is descheduled at Round Robin, it should print out the status as follows:
  `System.out.println(fileName + " is descheduled");`

Your code will be evaluated based on its correctness; which means the sorting operation and the `System.out.println()` outputs will be only indication that your code runs correctly. This project will be evaluated with blackbox testing, so it is all or nothing. Your output will be exactly the same with everyone else except Round Robin algorithm.

Step 1: Implement your own sorting algorithm: QuickSort or MergeSort

You can choose either single-threaded Quick Sort or single-threaded Merge Sort as your sorting algorithm, whichever you find easier. Your choice won't affect your grade as long is it runs correctly. Please do NOT use any Java built in sorting functions such as `Arrays.sort()` or `Collections.sort()`. When you deschedule the task, you will need to save the state of this algorithm, and you cannot do that with built-in methods. As a rule of thumb, the class you implement your sorting algorithm should implement `Serializable`. This will help you save the state very easily.

Step 2: Implement a MyProcess class (Optional)

MyProcess class may extend Java's own Thread class. You should consider this thread as your process. Its constructor takes the task as an argument. This class has an `interruptMe`() method, and when it is called, the task running in this thread is saved and sent back to the end of the task queue. Note that not all the algorithms need this method, only Round Robin.

Step 3: Implement a MyProcessManager class

The MyProcessManager class has a non-return type execute(MyProcess object) method that runs the MyProcess tasks. However, it first orders tasks in the order they should be, and then starts running them. It has to implement an internal timer (NOT the Timer class implemented in Java, it performs a completely different task), which calls the `interruptMe()` function of MyProcess when it needs to get descheduled.

Step 4: Implement any other classes you may need

This is where you are free. As long as your program runs correctly, you can add any class you can use. For example, the algorithms can be implemented as separate classes, if you choose to do so.

Step 5: The main method (Already given below)

The main class takes only one argument. Without the argument, it does not do anything. Name your main class as Project1.java so we should be able to call this command:

```
java Project2 /the/path/to/the/input/folder
```

The main method you should all be using is as follows:

```java
public class Main {
    public static void main(String[] args) {

        //You can appoint a folder location for your tests here
        String folderLocation = "";

        if (args == null || args.length == 0) {
            System.out.println("No folder name provided. Program is going to terminate.");
            System.exit(0);
        } else {
            folderLocation = args[0].toString();
        }

        MyProcessManager myProcessManager = new MyProcessManager();

        myProcessManager.setFolderLocation(folderLocation);
        //Test 1 - Priority Scheduling - 2 points
        myProcessManager.orderTasksByPriority();
        myProcessManager.runTasks();
        //Test 2 - FCFS - 2 points
        myProcessManager.orderTasksByFCFS();
        myProcessManager.runTasks();
        //Test 3 - SJF - 2 points
        myProcessManager.orderTasksBySJF();
        myProcessManager.runTasks();
        //Test 4 - RoundRobin - 2 points
        myProcessManager.orderTasksByRoundRobin();
        myProcessManager.runTasks();

        System.out.println("All tasks have been completed.");


    }
}
```

Grading

This project is 8% of your overall grade, and the total points you can get from this project is 8. Implementing the each algorithm correctly will get you 2 points. All the project members will receive the same grade.

By the end of the semester, your team needs to schedule a meeting with us, and answer questions about your projects. Each member of the team will have to answer these questions. Failing to answer these questions can result in getting a lower grade than the other members of the team. Hence, we recommend you to contribute equally, and have an idea about what your peers are working on. 4% of your overall grade will be given based on your performance at this meeting.

Submission

Step 1: You will use the same repositories that your team has not changed. If your team has changed since then, you will have to invite me as a developer to your private repository on github or bitbucket by **10/26/2018 (Friday) 11:59 pm**. Both github and bitbucket have free private repositories for students. My username on both of them is ***gkhnkul*** and this information is all you need to know to invite me as a developer. Everybody has to contribute to the project using this repository. No code exchange using USB drives or emails. We need to see each project member's contribution in the project history. If you do peer-programming, add that information to the commit comments. Any team that fails to do this step will be penalized 20 points for each late day.

Step 2: You will just have to push a stable version of your project to the repository every time you push your code, so we can try to monitor your progress. If we cannot compile your code, you will receive a zero (0) from the submission by the deadline.

Deadline: 11/05/2018 (Monday) 11:59 pm

Your projects will be downloaded to a computer automatically by a script. The deadline is firm and the script won't collect any other files after this exact minute. No late submissions are allowed since last time it did not improve your grades at all.

Warnings

- Please make sure that your project repository is private, and nobody except your team members and us can see or download your repository.
- You can discuss your strategy with other teams, but you cannot share code.
- Using code from the internet is forbidden. In the case that you do, you take the risk of being caught for cheating since another team may have used the same code in their projects. Your projects will be compared using MOSS which makes many cheating techniques basically useless.
- You are responsible for your team members.