COEN346 Programming Assignment #2

presented to

Dr. Ferhat Khendek

by

Jean-Philippe Forget (40137807)

Asha Islam (40051511)

Pavithra Sivagnanasuntharam (40117356)

Concordia University

Department of Electrical and Computer Engineering

November 2022

Introduction

This programming assignment consists in creating a Round-Robin scheduling system which will allocate time among a set of users followed by a set of processes. The input file provides the time quantum to be utilized throughout the program and the allocation of this time needs to be done among the users and processes. In order to do so, the input file needs to be read and the users and the processes need to be stored in their respective arrayList, each process representing a thread. This will then allow for the program to evaluate the time that needs to be shared among them, while updating this time throughout the entire task at the end of each cycle. The output should then display the processes run and stopped throughout the task with the Round-Robin scheduler until all the processes have been run.

Code Description

FileInput.java

This file is going to be reading the inputs of the input.txt file. It will go through each line of the file using the BufferedReader and its methods, such as readLine(). This is going to be done in the inputArrayRead() method. First, we are going to be reading the time quantum, followed by the user and the number of processes for each user. Since the BufferedReader reads the elements as strings, we are going to be using the split() method to separate the elements on the same line with a space (“ “) delimiter. Subsequently, we are going to be storing the processes’ burst time and arrival time in the object of type User (user.addProcess(p)). When the user’s process is done populating, we are going to add the user into the ArrayList<User> users.

Process.java

This class is going to be setting the elements of the process, such as the allocatedTime, serviceTime (burst time), arrivalTime, processID, currentTime, a boolean to check whether the process is ready or not, user_name, and a string indicating the state of the process. For all of these variables, getters and setters were defined. In the run() method, the system prints out the time and the state of the process while the conditional statements are respected. An example would be verifying that the burst time is zero, in which case the process is completed and removed from the queue. The removal will be done in another Java file. It is important to note that this class implements Runnable, and will include the concept of threads.

Scheduler.java

In this class, we have four methods, which are allocateTime(), run(), isNotDone() and addUser(). In the allocateTime() method, we are going to check how many users have at least one process ready to run, and according to that we are going to be dividing the time quantum with the number of users. The isNotDone() method is going to verify if the user is empty or not in order to determine whether we should allocate time to it or not. The addUser() method is going to be adding a user into an ArrayList of users. Since this class is implementing Runnable, we are going to have a run() method, in which we are going to call the isNotDone() method to make sure there is something to schedule. If so, we are going to run the process depending on its allocated time, and output it in a file. Once the process is finished running, it is going to be removed from the queue (current_user.user_processes.remove(runningProcess)).

User.java

The User class is going to define the elements of a user, such as the name and its allocated time. For these variables, we have created the getters and setters. Another method that is important to mention is the allocateTimeToProcesses(), which will check how many processes are ready and calculate the amount of time allocated to each of the processes that are ready. The addProcess() method is going to add a process in the ArrayList of processes for each User. The isReady() method is going to verify whether a process is ready or not to run. The isEmpty() method is going to verify if the ArrayList of user_processes is empty or not.

main.java

This file is going to invoke the main method in which we are going to declare the filepath in which there is the input.txt file. This file is then read and stored in the object of type Scheduler. The Scheduler is going to perform the Round Robin scheduling with a thread by invoking the method start(). Once it is done, we are going to join them with the join() method.

Conclusion

To conclude, the Round-Robin scheduler has been used in order to run processes from different users. In order to do this, each process has been stored as a thread and these threads are what were being run in order to complete the processes, according to their allocated time from the time quantum. This therefore helped in the understanding of utilizing threads to represent the processes, while also working with a scheduling method, allowing a better understanding of its functionality in an in depth manner. Multiple classes needed to be created in this case in order to represent each component that plays a role in the scheduling of processes.

Contribution Table

Members	Coding Contribution	Report Contribution
Jean-Philippe Forget	FileInput.java - inputArrayRead() - getUsers() - getTimeQuantum() Process.java - run() - Getters and setters Scheduler.java - allocateTime() - run() - isNotDone() - addUser() User.java - Getters and setters - allocateTimeToProcesses() - addProcess() - isReady() - isEmpty() Main.java	33.34%

	- main()	
Asha Islam	FileInput.java - inputArrayRead() - getUsers() - getTimeQuantum() Process.java - run() Scheduler.java - run() - addUser() User.java - Getters and setters - allocateTimeToProcesses() - addProcess() Main.java - main()	33.33%
Pavithra Sivagnanasuntharam	FileInput.java - inputArrayRead() - getUsers() - getTimeQuantum() Process.java - run() Scheduler.java - run() - addUser() User.java - Getters and setters - allocateTimeToProcesses() - addProcess() Main.java - main()	33.33%