Report

# Design

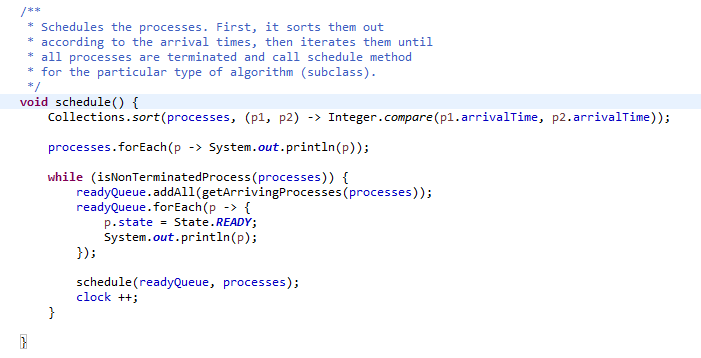
The main feature of our design is to use abstract class Scheduler that encapsulates all common properties of required algorithms, such as RR, SJF, FCFC, Priority. All classes above extend Scheduler and override a single overloaded abstract method schedule to add their own parts of algorithm. Scheduler implements method schedule without parameters to update clock (simple integer variable) until there is at least one non-terminated process (see the UML Class Diagram at the end of this Report).

The essential issue was to adjust MultilevelQueue and MultilevelFeedbackQueue classes into the class structure. The point is that, MQ and MFQ don’t require implemented schedule() method in the Scheduler class, we were forced just to override schedule() method to avoid its implementation and call overloaded schedule method, but with null parameters.

Other issue was to implement cpu/io sequence, because they should be placed in a row one by one. In my opinion, the most efficient way to do this is to use linked list data structure where every current item has a link to the next item. The last item has null link.

# Analysis

Let’s take two most important methods of Scheduler class for analysis. These are schedule() and schedule(Queue<Process>, List<Process>) methods. The first method invokes the second one in a loop, so the running time complexity of the first method consists of the running times both of methods. So, schedule() method involves 2 loops and one of them is nested loop with second method invocation:



The first loop can be neglected, so look at the second while nested loop. It iterates all processes while there is at least 1 non-terminated process, so if we assume that approximately 1 process terminated per

iteration, then = number of processes. Next, we have nested forEach loop which iterates over the processes, so now we have + schedule method invocation. Let’s go to the analysis schedule (…) method. If we look over the all implementations of this method in subclasses, then we can notice all of them can 2 nested loops: outer while loop (iterates over the processes) and inner do-while loop (iterates over the cpu / io sequence. So, for the outer loop n = number of processes and for the inner loop k = number of items in the cpu / io sequence, so the total running time complexity for our schedule algorithm is the following:

,

where **n** = number of processes to be executed and **k** = average number of items in cpu/io sequence per process.

# Implementation

See CPUScheduling Eclipse IDE project folder.

# Testing

See Test\_output.txt file for the whole output.

For testing purpose we used all examples from CpuScheduling pdf specification and found that the results are coincide with the result in the book.

Here is the partial output for the RR algorithm:

