Assignment #: 02
Course: EECS2011 E
Professor: Jia Xu
Name : Akalpit Sharma

Report

Design

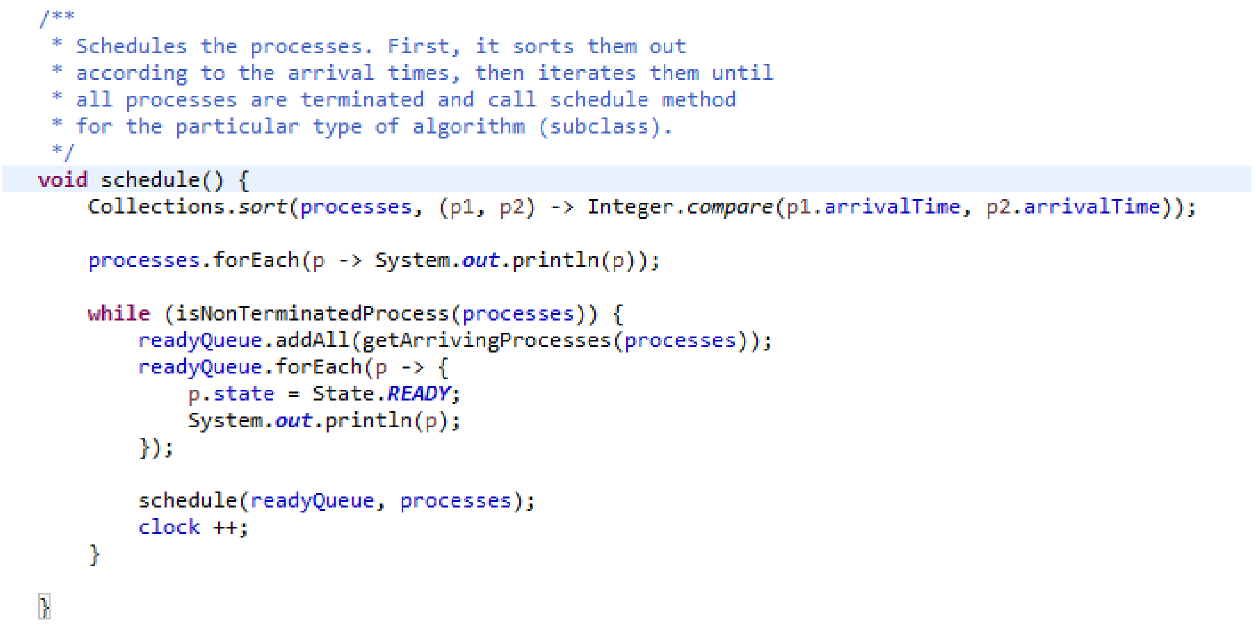
The main feature of our design is to use abstract class Scheduler that encapsulates all common properties of required algorithms, such as RR, SJF, FCFC, Priority. All classes above extend Scheduler and override a single overloaded abstract method schedule to add their own parts of algorithm. Scheduler 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:



```
/**
 * Schedules the processes. First, it sorts them out
 * according to the arrival times, then iterates them until
 * all processes are terminated and call schedule method
 * for the particular type of algorithm (subclass).
 */
void schedule() {
    Collections.sort(processes, (p1, p2) -> Integer.compare(p1.arrivalTime, p2.arrivalTime));

    processes.forEach(p -> System.out.println(p));

    while (isNonTerminatedProcess(processes)) {
        readyQueue.addAll(getArrivingProcesses(processes));
        readyQueue.forEach(p -> {
            p.state = State.READY;
            System.out.println(p);
        });

        schedule(readyQueue, processes);
        clock ++;
    }

}
```

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 n = number of processes. Next, we have nested forEach loop which iterates over the processes, so now we have n^2 + 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:

$$O\left(n * \left(n + (n * k)\right)\right) = O(n * (n + nk) = \boldsymbol{O(n^2 + n^2k)},$$

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.

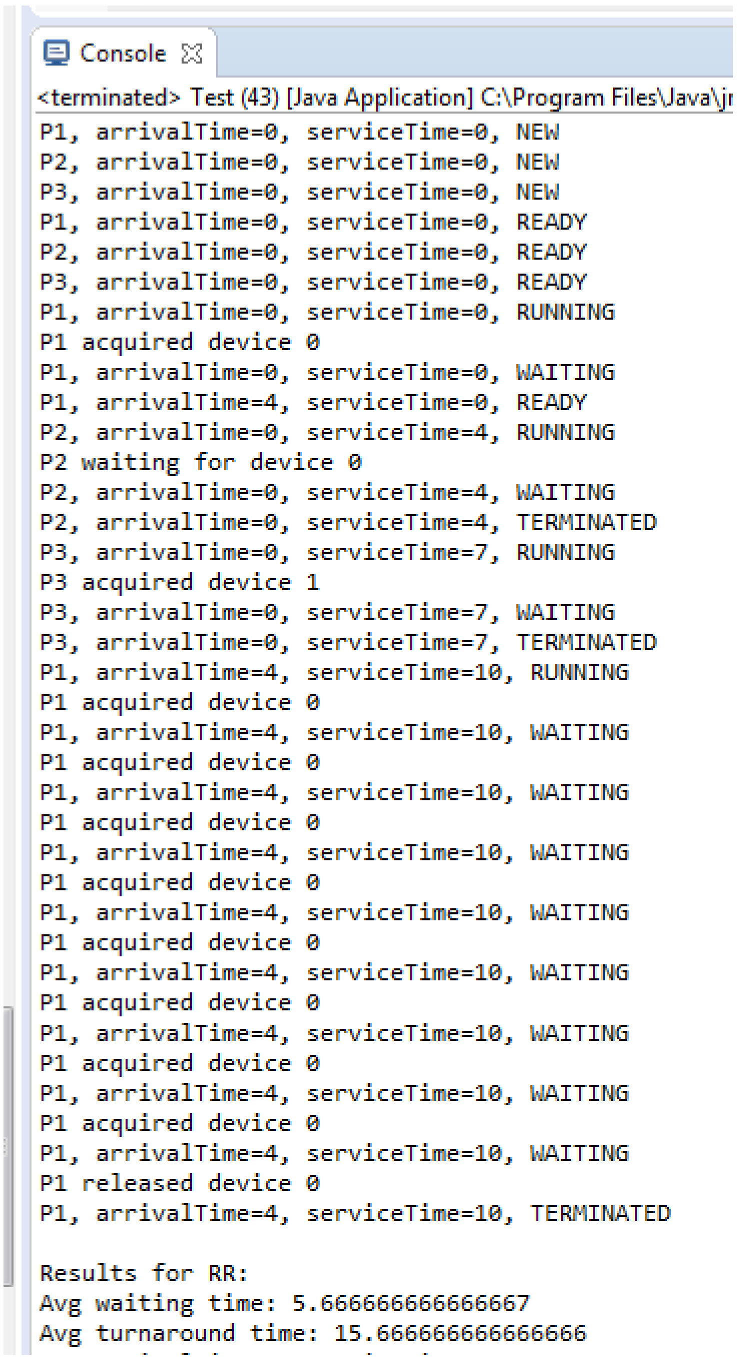
Testing

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:

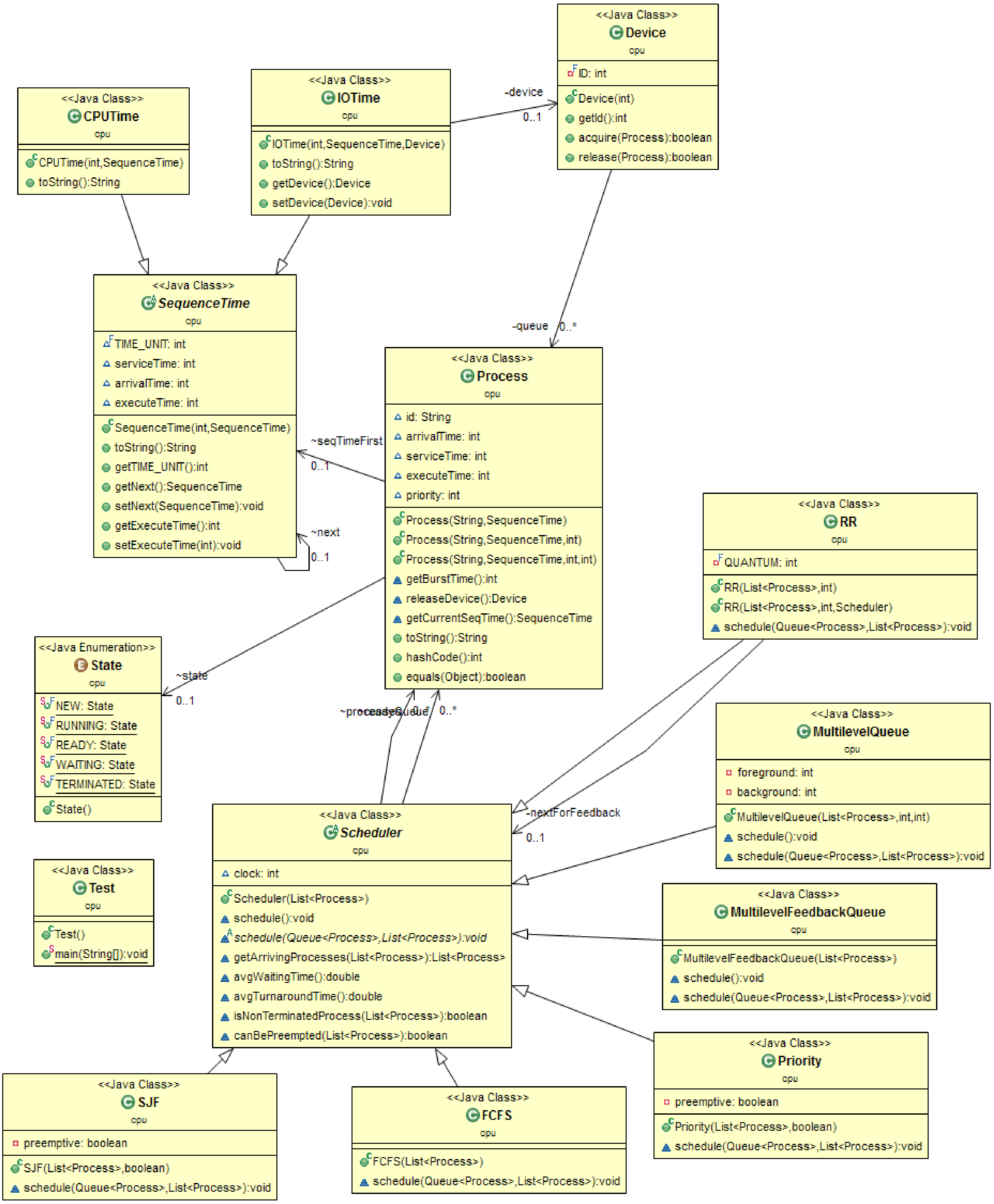
Assignment #: 02
Course: EECS2011 E
Professor: Jia Xu
Name : Akalpit Sharma



```
<terminated> Test (43) [Java Application] C:\Program Files\Java\j
P1, arrivalTime=0, serviceTime=0, NEW
P2, arrivalTime=0, serviceTime=0, NEW
P3, arrivalTime=0, serviceTime=0, NEW
P1, arrivalTime=0, serviceTime=0, READY
P2, arrivalTime=0, serviceTime=0, READY
P3, arrivalTime=0, serviceTime=0, READY
P1, arrivalTime=0, serviceTime=0, RUNNING
P1 acquired device 0
P1, arrivalTime=0, serviceTime=0, WAITING
P1, arrivalTime=4, serviceTime=0, READY
P2, arrivalTime=0, serviceTime=4, RUNNING
P2 waiting for device 0
P2, arrivalTime=0, serviceTime=4, WAITING
P2, arrivalTime=0, serviceTime=4, TERMINATED
P3, arrivalTime=0, serviceTime=7, RUNNING
P3 acquired device 1
P3, arrivalTime=0, serviceTime=7, WAITING
P3, arrivalTime=0, serviceTime=7, TERMINATED
P1, arrivalTime=4, serviceTime=10, RUNNING
P1 acquired device 0
P1, arrivalTime=4, serviceTime=10, WAITING
P1 acquired device 0
P1, arrivalTime=4, serviceTime=10, WAITING
P1 acquired device 0
P1, arrivalTime=4, serviceTime=10, WAITING
P1 acquired device 0
P1, arrivalTime=4, serviceTime=10, WAITING
P1 acquired device 0
P1, arrivalTime=4, serviceTime=10, WAITING
P1 acquired device 0
P1, arrivalTime=4, serviceTime=10, WAITING
P1 acquired device 0
P1, arrivalTime=4, serviceTime=10, WAITING
P1 acquired device 0
P1, arrivalTime=4, serviceTime=10, WAITING
P1 released device 0
P1, arrivalTime=4, serviceTime=10, TERMINATED

Results for RR:
Avg waiting time: 5.666666666666667
Avg turnaround time: 15.666666666666666
```

Assignment #: 02
Course: EECS2011 E
Professor: Jia Xu
Name : Akalpit Sharma


<<Java Class>>
Device
cpu
ID: int
Device(int)
getId():int
acquire(Process):boolean
release(Process):boolean
-device
0..1
<<Java Class>>
IOTime
cpu
IOTime(int,SequenceTime,Device)
toString():String
getDevice():Device
setDevice(Device):void
<<Java Class>>
CPUTime
cpu
CPUTime(int,SequenceTime)
toString():String
<<Java Class>>
SequenceTime
cpu
TIME_UNIT: int
serviceTime: int
arrivalTime: int
executeTime: int
SequenceTime(int,SequenceTime)
toString():String
getTIME_UNIT():int
getNext():SequenceTime
setNext(SequenceTime):void
getExecuteTime():int
setExecuteTime(int):void
~next
0..1
~seqTimeFirst
0..1
-queue
0..*
<<Java Class>>
Process
cpu
id: String
arrivalTime: int
serviceTime: int
executeTime: int
priority: int
Process(String,SequenceTime)
Process(String,SequenceTime,int)
Process(String,SequenceTime,int,int)
getBurstTime():int
releaseDevice():Device
getCurrentSeqTime():SequenceTime
toString():String
hashCode():int
equals(Object):boolean
<<Java Class>>
RR
cpu
QUANTUM: int
RR(List<Process>,int)
RR(List<Process>,int,Scheduler)
schedule(Queue<Process>,List<Process>):void
<<Java Enumeration>>
State
cpu
NEW: State
RUNNING: State
READY: State
WAITING: State
TERMINATED: State
State()
~state
0..1
~processes
~readyQueue
0..*
0..*
<<Java Class>>
MultilevelQueue
cpu
foreground: int
background: int
MultilevelQueue(List<Process>,int,int)
schedule():void
schedule(Queue<Process>,List<Process>):void
-nextForFeedback
0..1
<<Java Class>>
Scheduler
cpu
clock: int
Scheduler(List<Process>)
schedule():void
schedule(Queue<Process>,List<Process>):void
getArrivingProcesses(List<Process>):List<Process>
avgWaitingTime():double
avgTurnaroundTime():double
isNonTerminatedProcess(List<Process>):boolean
canBePreempted(List<Process>):boolean
<<Java Class>>
Test
cpu
Test()
main(String[]):void
<<Java Class>>
MultilevelFeedbackQueue
cpu
MultilevelFeedbackQueue(List<Process>)
schedule():void
schedule(Queue<Process>,List<Process>):void
<<Java Class>>
Priority
cpu
preemptive: boolean
Priority(List<Process>,boolean)
schedule(Queue<Process>,List<Process>):void
<<Java Class>>
SJF
cpu
preemptive: boolean
SJF(List<Process>,boolean)
schedule(Queue<Process>,List<Process>):void
<<Java Class>>
FCFS
cpu
FCFS(List<Process>)
schedule(Queue<Process>,List<Process>):void