|  |  |
| --- | --- |
| **الرقم الجامعي** | **الاسم** |
| **20221441990** | **حازم محمد احمد بكر** |
| **20221442040** | **مروان اشرف محمد عبدالباقي** |
| **20221150099** | **نورالدين أنور محمد محمود** |
| **20221449224** | **أحمد ناصر عبدالفضيل** |
| **20221450304** | **احمد سمير عبدالفتاح امين** |

**Scheduling algorithm:** We know that the processor can only process one task at a time. So, how it manages to process all of this? What decides what process and when is the process going to reach the processor? The answer to this question is the topic of today which is the scheduling algorithms. They decide what process should go and what should wait. There are two types of these algorithms:

1] preemptive algorithms: In this type of algorithms, when a task enters the processor it doesn’t have to finish before it can be stopped. It can stop, let some other process run, and runs again.

E.X: Round-robin, Shortest remaining time first

2] non-preemptive algorithms: In this type, when a process starts it will not stop until it’s finished

E.X: First in first out, shortest job first

**MacOS:** The macOS uses the multi-level feedback queue which is a preemptive algorithm that contains multiple queues that each queue can use a different scheduling algorithm. In the macOS, they use round-robin in the queues. Steps of the algorithm as follows:

1] when a task enters, it is pushed in the highest priority queue

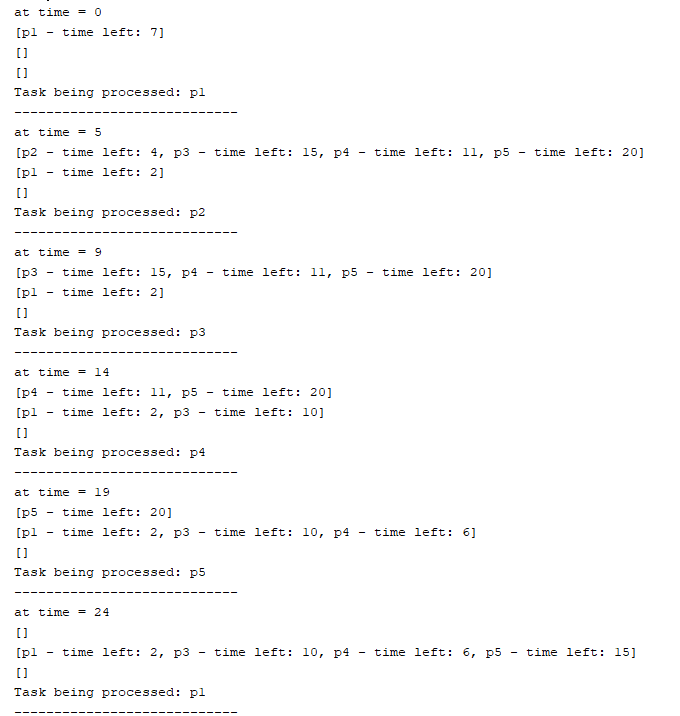
2] if the task is finished in the predefined time quantum, it’s removed. Else the task priority is reduced and it goes in the next priority queue or stays the same (if it’s already in the last queue).

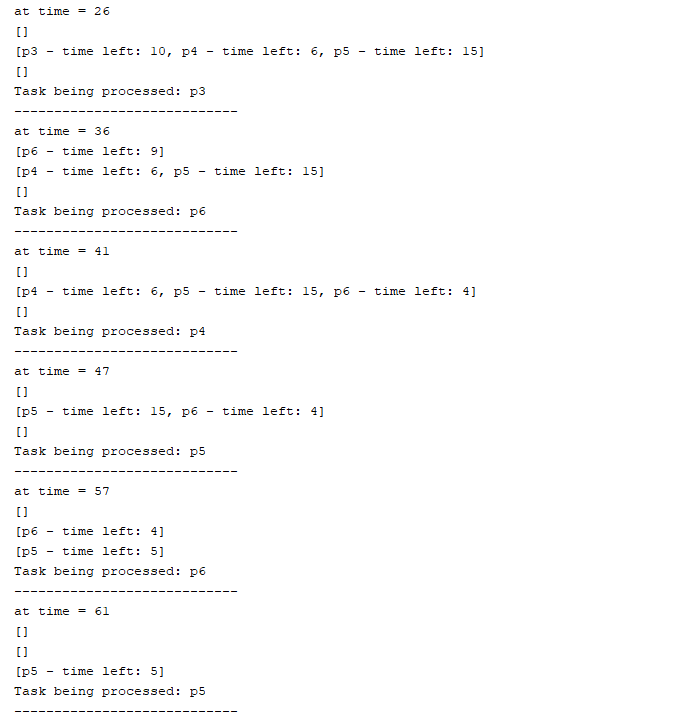
3] the algorithm continues to process tasks in the highest priority queues first and only proceed to next one if they are empty

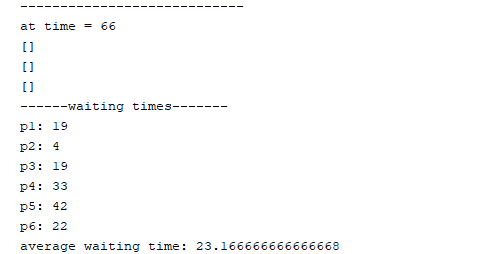
Processes we used in our code:

|  |  |  |
| --- | --- | --- |
| ID | Arriving time | Execution time |
| P1 | 0 | 7 |
| P2 | 1 | 4 |
| P3 | 2 | 15 |
| P4 | 3 | 11 |
| P5 | 4 | 20 |
| P6 | 30 | 9 |

Steps of the algorithm on our processes:







**The Code:**

**The process class:**

**package** AssignmentOS;

public class process {

***String*** id;

***int*** arrival\_time;

    // The reason we have execution time and time left is we want something fixed to be able to calculate waiting time at the end and we need something that will change when we work on our process

***int*** execution\_time;

***int*** time\_left;

    // The time the process got finished to calculate waiting time

***int*** out\_time;

    public process(***String*** *id*, ***int*** *arrival\_time*, ***int*** *execution\_time*) {

        this.id **=** id;

        this.arrival\_time **=** arrival\_time;

        this.execution\_time **=** execution\_time;

        this.time\_left**=**execution\_time;

    }

    public process(){

    }

    public ***String*** toString(){

**return** this.id **+** " - time left: " **+** this.time\_left;

    }

    public ***int*** waitingTime(){

        //To calculate waiting time we see when did the task finished, when did it arrive, how much of this time it was running (the remaining time means it was waiting)

**return** this.out\_time **-** this.arrival\_time **-** this.execution\_time;

    }

}

**The main class:**

**package** AssignmentOS;

**import** java.util.LinkedList;

**import** java.util.Queue;

public class MLFQ {

    public static ***void*** main(***String***[] *args*) {

        // pre-defining our process that will enter the queues to be able to monitor them

       process[] tasks **=** {**new** process("p1",0,7),**new** process("p2",1,4),**new** process("p3",2,15),

**new** process("p4",3,11),**new** process("p5",4,20),**new** process("p6",30,9)};

       // counter for how many tasks arrived already

***int*** added **=** 0;

       // the time quantum for each queue

***int***[] qt **=** {5,10,15};

       // counter to keep track of the current time

***int*** t **=** 0;

       // the multi queues

***Queue***[]q **=** {**new** ***LinkedList***<>(),**new** ***LinkedList***<>(),**new** ***LinkedList***<>()};

       // we will keep processing untill no new tasks arrive and all queues are emtpy

**while**(added**!=**tasks.length **|** **!**queuesEmpty(q)){

           // check which tasks have arrived and add them in the first queue

**for**(***int*** i **=** added; i **<** tasks.length; i**++**){

**if**(tasks[i].arrival\_time**<=**t){

                   q[0].add(tasks[i]);

                   added**++**;

               }

           }

           System.out.println("at time = "**+**t);

           printQueues(q);

           // looping each queue to process the first task in the highest priority one

**for**(***int*** i **=** 0 ; i **<** q.length ; i**++**){

**if**(q[i].size()**>**0){

                   // either increasing the time with a time quantum or if the task takes less time we increase it by its time left

                   process p **=** (process) q[i].element();

                   System.out.println("Task being processed: "**+**p.id);

***int*** time\_passed**=**Math.min(qt[i], p.time\_left);

                   t**+=**time\_passed;

                   p.time\_left**-=**time\_passed;

                   // if the task finished remove it otherwise add it to the next priority ueue

**if**(p.time\_left**!=**0){

                       q[Math.min(q.length**-**1, i**+**1)].add(q[i].remove());

                   }**else**{

                       p.out\_time**=**t;

                       q[i].remove();

                   }

**break**;

               }

           }

           System.out.println("----------------------------");

       }

           System.out.println("at time = "**+**t);

           printQueues(q);

           System.out.println("------waiting times-------");

***int*** total **=** 0;

**for**(***int*** i **=** 0 ; i **<** tasks.length ; i**++**){

***int*** wt **=** tasks[i].waitingTime();

               total**+=**wt;

               System.out.println(tasks[i].id **+** ": " **+** wt);

           }

           System.out.println("average waiting time: "**+**(***double***)total**/**tasks.length);

    }

    public static ***void*** printQueues(***Queue***[]*q*){

**for**(***int*** i **=** 0 ; i **<** q.length ; i**++**){

            System.out.println(q[i]);

        }

    }

    public static ***boolean*** queuesEmpty(***Queue***[]*q*){

**for**(***int*** i **=** 0 ; i **<** q.length ; i**++**){

**if**(q[i].size()**!=**0){

**return** false;

            }

        }

**return** true;

    }

}