**OPERATING SYSTEMS**

**PROJECT 1: SOURCE CODE**

**Name: Tanveen Kaur ID: 1014603 Date: 03-18-2018**

**/\* Multilevel FeedBack Queue Scheduler \*/**

**1) MultiLevelFeedbackQueueScheduler.java**

**package** os\_project1\_queueScheduler;

**import** java.util.LinkedList;

**import** java.io.\*;

**public** **class** MultiLevelFeedbackQueueScheduler

{

LinkedList<ProcessData> proc;

LinkedList<ProcessData> l1Q;

LinkedList<ProcessData> l2Q;

LinkedList<ProcessData> l3Q;

LinkedList<Integer> pCrtnTimes;

LinkedList<Integer> pStrtTimes;

LinkedList<Integer> execuTimes;

LinkedList<Integer> pTrmntnTimes;

MultiLevelFeedbackQueueScheduler() {

proc = **new** LinkedList<ProcessData>();

l1Q = **new** LinkedList<ProcessData>();

l2Q = **new** LinkedList<ProcessData>();

l3Q = **new** LinkedList<ProcessData>();

pCrtnTimes = **new** LinkedList<Integer>();

pStrtTimes = **new** LinkedList<Integer>();

execuTimes = **new** LinkedList<Integer>();

pTrmntnTimes = **new** LinkedList<Integer>();

}

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

MultiLevelFeedbackQueueScheduler mlfq = **new** MultiLevelFeedbackQueueScheduler();

mlfq.initialize();

**int** currentTime = 0;

**int** linePrint = 0;

**int** totalWaitingTime = 0, totalTurnAroundTime = 0, totalResponseTime = 0;

System.***out***.println("Time\t\tDescription");

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

**while** (**true**) {

**if** (mlfq.proc.size() == 0 && mlfq.l1Q.size() == 0

&& mlfq.l2Q.size() == 0

&& mlfq.l3Q.size() == 0) {

**break**;

}

LinkedList<ProcessData> newlyCreatedProcesses = **new** LinkedList<ProcessData>();

**for** (**int** i = 0; i < mlfq.proc.size(); i++) {

**if** (mlfq.proc.get(i).getArrTime() == currentTime) {

ProcessData process = mlfq.proc.get(i);

mlfq.l1Q.add(process);

newlyCreatedProcesses.add(process);

**if** (linePrint == 0) {

System.***out***.print(currentTime + " ms\t\t");

linePrint = 1;

}

System.***out***.print("P" + process.getPId()

+ " created. P" + process.getPId()

+ " enters Q0.");

}

}

// Remove newly created processes that are added to Q1 from

// allProcesses

**for** (**int** i = 0; i < newlyCreatedProcesses.size(); i++)

mlfq.proc.remove(newlyCreatedProcesses.get(i));

// Check level1Queue

ProcessData currentProcess;

**if** (mlfq.l1Q.size() != 0) {

currentProcess = mlfq.l1Q.get(0);

**if** (linePrint == 0) {

System.***out***.print(currentTime + " ms\t\t");

linePrint = 1;

}

**if** (currentProcess.getTimeInQueue() == 0)

mlfq.pStrtTimes.set(

currentProcess.getPId() - 1, currentTime);

**if** (currentProcess.getTimeInQueue() < 8) {

**if** (currentProcess.getExecuTime() <= 1) {

mlfq.l1Q.remove();

mlfq.pTrmntnTimes.set(

currentProcess.getPId() - 1, currentTime);

System.***out***.print("P" + currentProcess.getPId()

+ " completed execution and terminated.");

} **else** {

currentProcess.setExecuTime(currentProcess

.getExecuTime() - 1);

currentProcess

.setTimeInQueue(currentProcess

.getTimeInQueue() + 1);

System.***out***.print("P" + currentProcess.getPId()

+ " is processing in Q1.");

**if** (currentProcess.getTimeInQueue() == 8) {

currentProcess.setTimeInQueue(0);

mlfq.l2Q.add(currentProcess);

mlfq.l1Q.remove(currentProcess);

System.***out***.print("P"

+ currentProcess.getPId()

+ " aged and moved from Q1 to Q2.");

}

}

}

}

// Check level2Queue

**else** **if** (mlfq.l2Q.size() != 0) {

currentProcess = mlfq.l2Q.get(0);

**if** (linePrint == 0) {

System.***out***.print(currentTime + " ms\t\t");

linePrint = 1;

}

**if** (currentProcess.getTimeInQueue() < 16) {

**if** (currentProcess.getExecuTime() <= 1) {

mlfq.l2Q.remove();

mlfq.pTrmntnTimes.set(

currentProcess.getPId() - 1, currentTime);

System.***out***.print("P" + currentProcess.getPId()

+ " completed execution and terminated.");

} **else** {

currentProcess.setExecuTime(currentProcess

.getExecuTime() - 1);

currentProcess

.setTimeInQueue(currentProcess

.getTimeInQueue() + 1);

System.***out***.print("P" + currentProcess.getPId()

+ " is processing in Q2.");

**if** (currentProcess.getTimeInQueue() == 16) {

currentProcess.setTimeInQueue(0);

mlfq.l3Q.add(currentProcess);

mlfq.l2Q.remove(currentProcess);

System.***out***.print("P"+ currentProcess.getPId() + " aged and moved from Q2 to Q3.");

}

}

}

}

// check level3Queue

**else** **if** (mlfq.l3Q.size() != 0) {

currentProcess = mlfq.l3Q.get(0);

**if** (linePrint == 0)

{

System.***out***.print(currentTime + " ms\t\t");

linePrint = 1;

}

System.***out***.print("P" + currentProcess.getPId() + " is processing in Q3.");

**if** (currentProcess.getExecuTime() <= 1) {

mlfq.l3Q.remove();

mlfq.pTrmntnTimes.set(currentProcess.getPId() - 1, currentTime);

System.***out***.print("P" + currentProcess.getPId()+ " completed execution and terminated.");

} **else**

currentProcess.setExecuTime(currentProcess.getExecuTime() - 1);

}

currentTime++;

**if** (linePrint == 1)

System.***out***.println();

linePrint = 0;

}

**for** (**int** i = 0; i < mlfq.execuTimes.size(); i++) {

totalWaitingTime += mlfq.pTrmntnTimes.get(i) - (mlfq.pCrtnTimes.get(i) + mlfq.execuTimes.get(i));

totalTurnAroundTime += mlfq.pTrmntnTimes.get(i) - mlfq.pCrtnTimes.get(i);

totalResponseTime += mlfq.pStrtTimes.get(i) - mlfq.pCrtnTimes.get(i);

}

System.***out***.println("\nAverage waiting time = " + (totalWaitingTime / mlfq.pCrtnTimes.size()));

System.***out***.println("Average turnaround time = " + (totalTurnAroundTime / mlfq.pCrtnTimes.size()));

System.***out***.println("Average response time = " + (totalResponseTime / mlfq.pCrtnTimes.size()));

}

**public** **void** initialize() **throws** IOException {

String[] values = **null**;

// String numOfProcess = null;

**int** arrivalTime = 0, actualExecutionTime = 0;

System.***out***.println("ProcessId\tArrivalTime\tExecutionTime");

BufferedReader br = **new** BufferedReader(**new** FileReader("C:\\Users\\Guniya\\Downloads\\os\_project\_one\\os\_project\_one\\src\\os\_project1\_queueScheduler\\ProcessInputFile.txt"));

String line = br.readLine();

// int count =0;

**while** ((line = br.readLine()) != **null**)

{

values = line.split(",");

arrivalTime = Integer.*parseInt*(values[1]);

actualExecutionTime = Integer.*parseInt*(values[2]);

ProcessData newProcess = **new** ProcessData(proc.size()+1, arrivalTime, actualExecutionTime,0);

proc.add(newProcess);

System.***out***.println("p" + newProcess.getPId() + "\t\t" + newProcess.getArrTime() + "\t\t" + newProcess.getExecuTime());

pCrtnTimes.add(arrivalTime);

execuTimes.add(actualExecutionTime);

pStrtTimes.add(-1);

pTrmntnTimes.add(-1);

}

**try** {

Thread.*sleep*(3000);

} **catch** (Exception e) {

e.printStackTrace();

}

br.close();

}

}

**2) ProcessData.java**

**package** os\_project1\_queueScheduler;

**public** **class** ProcessData {

**int** pId;

**int** arrTime;

**int** execuTime;

**int** timeInQueue;

**public** **int** getTimeInQueue() {

**return** timeInQueue;

}

**public** **void** setTimeInQueue(**int** timeInQueue) {

**this**.timeInQueue = timeInQueue;

}

ProcessData(**int** pId, **int** arrTime,

**int** execuTime, **int** timeInQueue) {

**this**.pId = pId;

**this**.arrTime = arrTime;

**this**.execuTime = execuTime;

**this**.timeInQueue = timeInQueue;

}

**public** **void** setPId(**int** pId) {

**this**.pId = pId;

}

**public** **void** setArrTime(**int** arrTime) {

**this**.arrTime = arrTime;

}

**public** **int** getExecuTime() {

**return** execuTime;

}

**public** **void** setExecuTime(**int** execuTime) {

**this**.execuTime = execuTime;

}

**public** **int** getArrTime() {

**return** arrTime;

// **TODO** Auto-generated method stub

//return 0;

}

**public** **int** getPId() {

// **TODO** Auto-generated method stub

**return** pId;

}

}

**3) ProcessInputFile.txt**

p0,0,53

p1,20,17

p2,37,68

p3,57,24

p4,60,30

p5,65,40

p6,50,55

p7,70,60

p8,63,44

p9,69,50

p10,44,70

p11,30,62

p12,33,51

p13,47,58

p14,24,41

p15,29,20

p16,41,25

p17,52,30

p18,55,18

p19,15,24

p20,25,50

**4) package-info.java**

/\*\*

\*

\*/

/\*\*

\* **@author** TanveenK

\*

\*/

**package** os\_project1\_queueScheduler;