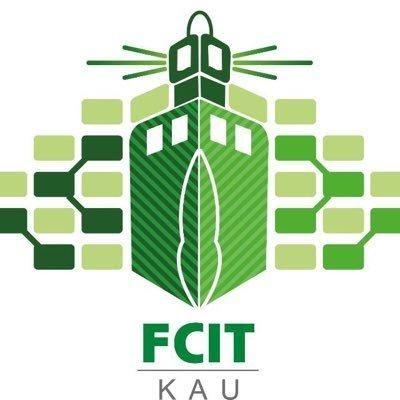
Saudi Arabia

King Abdul Aziz University

The faculty of computer and information technology

Computer Science Department

***Course Project***

CPCS 371 Operating System

*Fall 20-21*

**Prepared by:**

|  |  |  |
| --- | --- | --- |
| 1780367 | Aya Kazzaz | GAR |
| 1805645 | Hadeel aloufi | GAR |
| 1780293 | Salwa Abbara | GAR |
| 1780735 | Mona Hafez | GAR |
| 1806494 | Aisha Baskran | GAR |

Table of Contents

**1.Introduction 3**

**2 Design based on UML Diagrams 4**

**3 A copy of a printout of your source code5**

osMain(Dynamic)38

CPU 21

osStaticMain (Static) 22

Job 33

CPU 37

**4 A copy of the program output on each test data file38**

Screenshots for dynamic roundrobin 38

Screenshots for brute force round robin 45

**5 The Comparative study of PART II 52**

Input 1 52

Input 2 53

Input 3 54

**3.Refrences** **55**

1. **Introduction**

In this report, we'll talk a little bit about CPU process scheduling, and more Specifically we'll be discussing Round Robin “RR” and Dynamic Round Robin .

So, what is round robin? what is dynamic round robin ? , and we provide a code written in Java.

And also, We will compare the RR and DRR at the end of the report, and finally we will review the results.

**1.2 What is round robin ?**

Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way, CPU is assigned to the process based on FCFS for a fixed amount of time. This fixed amount of time is called as **time quantum** or **time slice**.

After the time quantum expires, the running process is preempted and sent to the ready queue. Then, the processor is assigned to the next arrived process.

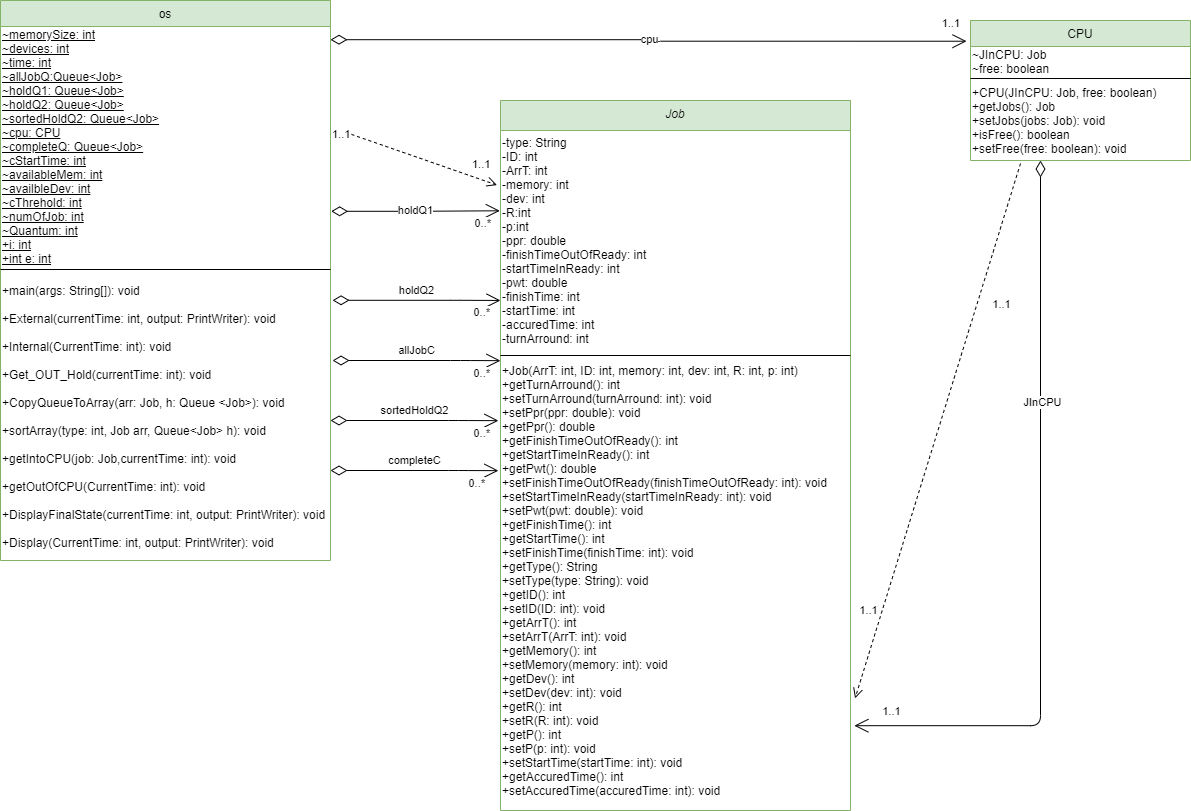
**1.3 What is dynamic round robin ?**

Reducing time cost in shared OS time is the main goal of researchers interested in CPU scheduling.

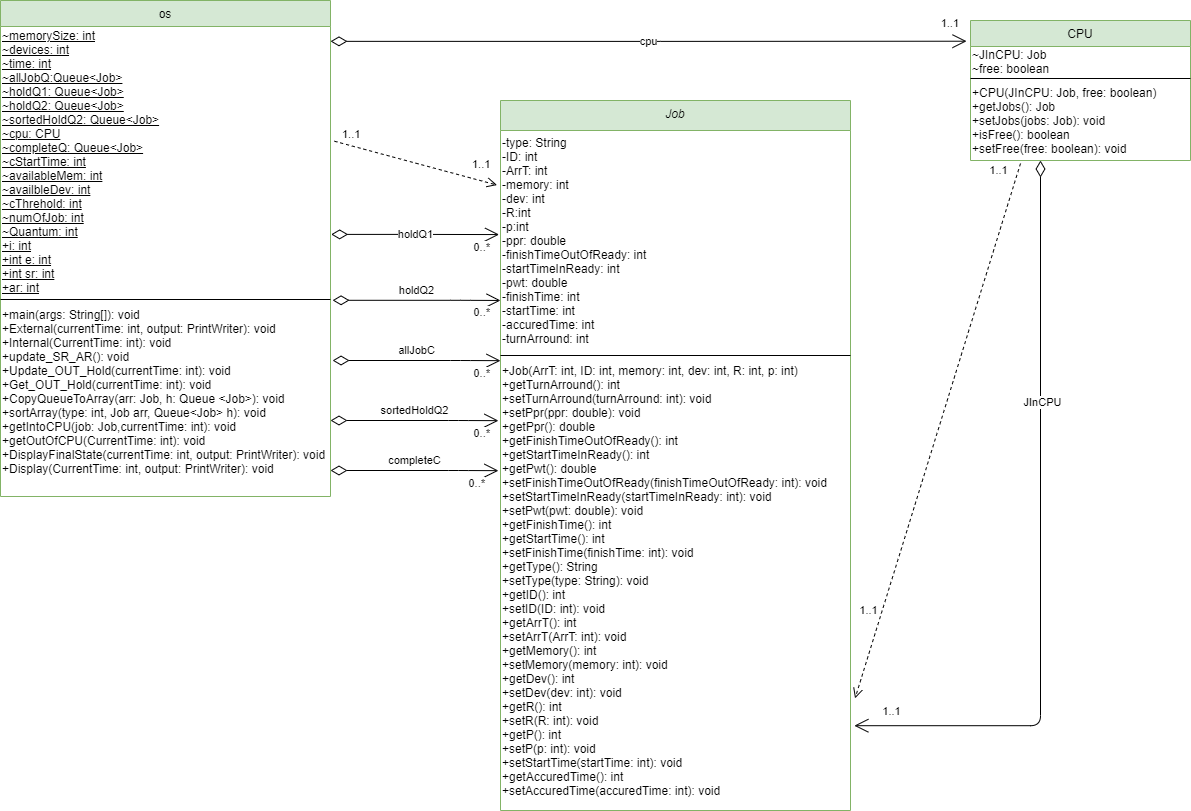
A modified version of the RR algorithm was introduced to combine the advantages of the preferred short process and lower scheduling expense of the RR in order to reduce the average wait time and completion time. Each process in the block is assigned the same time slot based on the batch weight and CPU lifetime.

1. **Design based on UML Diagrams**

* **UML static RR**

****

* **UML Dynamic RR**

****

1. **A copy of a printout of your source code**

**osMain(Dynamic)**

**import java.io.FileNotFoundException;**

**import java.io.PrintWriter;**

**import java.util.Iterator;**

**import java.util.LinkedList;**

**import java.util.Queue;**

**import java.util.Scanner;**

**//Aya Kazzaz,Hadeel aloufi,Salwa Abbara,Mona Hafez,Aisha Baskran,windows10,version1909(OS Build 18363.1256)**

**//processor:Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz**

**//Installed memory(RAM): 8.00 GB(7.90 GB usable)**

**//compiler name and version:java version "1.8.0\_191" ,Java(TM) SE Runtime Environment**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**/\*\***

**\***

**\* @author WinDows**

**\*/**

**public class osMain {**

**static int memorySize;**

**static int devices;**

**static int time;**

**static Queue<Job>allJobQ=new LinkedList();**

**static Queue<Job>holdQ1=new LinkedList();**

**static Queue<Job>holdQ2=new LinkedList();**

**static Queue<Job>sortedHoldQ2=new LinkedList();**

**static CPU cpu=new CPU(null,true);**

**static Queue<Job>completeQ=new LinkedList();**

**static int cStartTime=0;**

**static int availableMem=0;**

**static int availbleDev=0;**

**static int cThrehold=0;**

**static int numOfJob=0;**

**static int Quantum=0;**

**public static int i;**

**public static int e;**

**public static int sr;**

**public static int ar;**

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

**java.io.File file = new java.io.File("input1.txt");**

**// java.io.File file = new java.io.File("input2.txt");you can change between these 3 input files to check outputs files**

**//java.io.File file = new java.io.File("input3.txt");**

**java.io.File outputFile = new java.io.File("output.txt");**

**PrintWriter output=new PrintWriter(outputFile);//print writer to write to the outputfile**

**Scanner input=new Scanner(file);**

**while (input.hasNext()) {**

**String command1=input.next();**

**time=input.nextInt();**

**memorySize=Integer.parseInt(input.next().substring(2));**

**availableMem=memorySize;**

**devices=Integer.parseInt(input.next().substring(2));**

**availbleDev=devices;**

**Job job;**

**numOfJob=0;**

**int number = 0;**

**do{**

**String command2=input.next();**

**if(command2.equals("A")){**

**int ArrT= Integer.parseInt(input.next());**

**int ID=Integer.parseInt(input.next().substring(2));**

**int memory=Integer.parseInt(input.next().substring(2));**

**int dev=Integer.parseInt(input.next().substring(2));**

**int R=Integer.parseInt(input.next().substring(2));**

**int p=Integer.parseInt(input.next().substring(2));**

**if(memory<=memorySize&&dev<=devices){**

**job=new Job(ArrT,ID,memory,dev,R,p);**

**allJobQ.add(job);**

**numOfJob++;//counter to count the number of jobs**

**}**

**}else if(command2.equals("D")){**

**int time1=input.nextInt();**

**if(time1<999999){**

**job=new Job(time1,-1,-1,-1,-1,-1);**

**allJobQ.add(job);**

**}**

**else{**

**break;**

**}**

**}**

**} while(input.hasNext());**

**Job j=allJobQ.poll();**

**int currentTime=j.getArrT();**

**availableMem-=j.getMemory();**

**Quantum=j.getR();//take the first job and set the quantum to its burst time**

**getIntoCPU(j,currentTime);**

**i = 0;**

**e = 0;**

**do {**

**if (!(allJobQ.isEmpty())) {**

**i = allJobQ.peek().getArrT();**

**} else {**

**i = Integer.MAX\_VALUE;**

**}**

**if (cpu.isFree()==false) {**

**e = cpu.JInCPU.getFinishTime();**

**} else {**

**e = Integer.MAX\_VALUE;**

**}**

**currentTime = Math.min(i, e);**

**if (i < e) {**

**External(currentTime, output);**

**} else if (e < i) {**

**Internal(currentTime);**

**} else {**

**Internal(currentTime);**

**External(currentTime, output);**

**}**

**Iterator<Job> value = completeQ.iterator();**

**} while (completeQ.size() != numOfJob);**

**DisplayFinalState(currentTime,output);//displaying the final state**

**completeQ.clear();**

**}**

**input.close();**

**output.close();**

**}**

**public static void External(int currentTime, PrintWriter output){**

**if(!allJobQ.isEmpty()){**

**if(allJobQ.peek().getID()== -1){**

**allJobQ.poll();**

**Display(currentTime,output);**

**}else {**

**Job job=allJobQ.poll();**

**if(job.getMemory()>availableMem || job.getDev()>availbleDev){**

**holdQ2.add(job);**

**job.setStartTimeInReady(currentTime);//start timer when a job enter the holdQ2**

**}else if(job.getMemory()<=availableMem && job.getDev()<=availbleDev){**

**holdQ1.add(job);**

**availableMem-=job.getMemory();**

**availbleDev-=job.getDev();**

**update\_SR\_AR();**

**//**

**//**

**}**

**}**

**}**

**}**

**public static void Internal(int CurrentTime) {**

**getOutOfCPU(CurrentTime);**

**if (!holdQ1.isEmpty()) {**

**update\_SR\_AR();**

**Quantum=ar;**

**getIntoCPU(holdQ1.poll(), CurrentTime);**

**}**

**}**

**public static void update\_SR\_AR() {**

**if (holdQ1.isEmpty()) {//if no element , to avoid dividing over 0**

**ar = sr = 0;**

**return;**

**}**

**Iterator<Job> value = holdQ1.iterator();**

**Job o ;**

**while (value.hasNext()) {**

**o = value.next();**

**if (o.getP() == 1) {**

**sr += (o.getR()-o.getAccuredTime()) \* 2;**

**} else {**

**sr += (o.getR()-o.getAccuredTime()) \* 1;**

**}**

**}**

**ar = sr / (holdQ1.size());**

**Quantum=ar;**

**sr=0;**

**}**

**public static void Update\_OUT\_Hold(int currentTime){**

**Iterator<Job> value = holdQ2.iterator();**

**Job o ;**

**double avgWT;**

**double sumWT=0;**

**while (value.hasNext()) {**

**o = value.next();**

**o.setFinishTimeOutOfReady(currentTime);//set finish time in holdQ2**

**o.setPwt(o.getFinishTimeOutOfReady()-o.getStartTimeInReady());//set proccess waiting time**

**sumWT+=o.getPwt();**

**}**

**avgWT=sumWT/holdQ2.size();**

**value = holdQ2.iterator();**

**while (value.hasNext()) {**

**o = value.next();**

**if(o.getPpr()==-1){**

**o.setPpr(o.getP());**

**}else{**

**if(o.getPwt()-avgWT>0){**

**double DP = (o.getPwt()-avgWT)\*0.2+o.getPpr()\*0.8;**

**o.setPpr(DP);**

**}**

**}**

**}**

**}**

**public static void Get\_OUT\_Hold(int currentTime) {**

**if(!holdQ2.isEmpty()){**

**Update\_OUT\_Hold(currentTime);**

**Job [] sortingArray= new Job[holdQ2.size()];**

**Job o;**

**Iterator<Job> value =holdQ2.iterator();**

**while (value.hasNext()){**

**o=value.next();**

**}**

**value =holdQ1.iterator();**

**while (value.hasNext()){//copy element to array**

**o=value.next();**

**}**

**CopyQueueToArray(sortingArray , holdQ2);**

**for (int j = 0; j < sortingArray.length; j++) {**

**}**

**sortArray(1,sortingArray, holdQ2);**

**holdQ2.clear();**

**for (int j = 0; j < sortingArray.length; j++) {**

**holdQ2.add(sortingArray[j]);**

**}**

**int sizeq2=holdQ2.size();**

**for(int i=0;i<sizeq2;i++){**

**Job temp=holdQ2.poll();**

**if(temp.getMemory()<=availableMem && temp.getDev()<=availbleDev){**

**holdQ1.add(temp);**

**availableMem-=temp.getMemory();**

**availbleDev-=temp.getDev();**

**update\_SR\_AR();**

**}else{**

**holdQ2.add(temp);**

**}**

**}**

**}**

**}**

**public static void CopyQueueToArray(Job arr[] , Queue<Job> h){**

**int k =0;**

**Job o;**

**Iterator<Job> value = h.iterator();**

**while (value.hasNext()){**

**o=value.next();**

**arr[k]=o;**

**k++;**

**}**

**}**

**public static void sortArray(int type,Job arr[], Queue<Job> h) {**

**Job temp;**

**if(type==1){**

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

**{**

**for (int j = i+ 1; j < arr.length; j++)**

**{**

**if (arr[i].getPpr() < arr[j].getPpr())**

**{**

**temp = arr[i];**

**arr[i] = arr[j];**

**arr[j] = temp;**

**}**

**else if(arr[i].getPpr() == arr[j].getPpr()){//if same jobs have the same ppr then check the IDs**

**if (arr[i].getID() > arr[j].getID()) {**

**temp = arr[i];**

**arr[i] = arr[j];**

**arr[j] = temp;**

**}**

**}**

**}**

**}**

**}else{**

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

**{**

**for (int j = i + 1; j < arr.length; j++)**

**{**

**if (arr[i].getID() > arr[j].getID())**

**{**

**temp = arr[i];**

**arr[i] = arr[j];**

**arr[j] = temp;**

**}**

**}**

**}**

**}**

**h.clear();**

**for (int j = 0; j < arr.length; j++) {**

**h.add(arr[j]);**

**}**

**}**

**public static void getIntoCPU(Job job,int currentTime){**

**cpu.setJobs(job);**

**cpu.setFree(false);**

**cpu.JInCPU.setStartTime(currentTime);**

**int RemainingTime=cpu.JInCPU.getR()-cpu.JInCPU.getAccuredTime();//set remaining time**

**if(RemainingTime>= Quantum){**

**cpu.JInCPU.setFinishTime(currentTime+Quantum);**

**}else{**

**cpu.JInCPU.setFinishTime(currentTime+RemainingTime);**

**}**

**}**

**public static void getOutOfCPU(int CurrentTime){**

**if (cpu.isFree()== false) {**

**cpu.JInCPU.setAccuredTime(cpu.JInCPU.getAccuredTime()+(cpu.JInCPU.getFinishTime() - cpu.JInCPU.getStartTime()));**

**if (cpu.JInCPU.getAccuredTime() == cpu.JInCPU.getR()) {**

**availableMem += cpu.JInCPU.getMemory();**

**availbleDev += cpu.JInCPU.getDev();**

**completeQ.add(cpu.JInCPU);**

**cpu.setFree(true);**

**cpu.JInCPU=null;**

**if(!holdQ2.isEmpty()){//when the job left the cpu it release its memory and devices so we can move job from holdQ2 to holdQ1**

**Get\_OUT\_Hold(CurrentTime);**

**}**

**}else {//if the process not terminated**

**Quantum=ar;**

**holdQ1.add(cpu.JInCPU);**

**cpu.setFree(true);**

**update\_SR\_AR();**

**}**

**}**

**}**

**public static void DisplayFinalState(int currentTime,PrintWriter output){**

**//print all the system**

**output.println("<< Final state of system: ");**

**output.println(" Current Available Main Memory = "+availableMem);**

**output.println(" Current Devices = "+availbleDev+"\n");**

**output.println(" Completed jobs:");**

**output.println(" ----------------");**

**output.println(" Job ID Arrival Time Finish Time Turnaround Time ");**

**output.println(" ==================================================================");**

**Job [] sortingArray= new Job[completeQ.size()];**

**CopyQueueToArray(sortingArray , completeQ);**

**sortArray(2,sortingArray, completeQ);**

**completeQ.clear();**

**for (int j = 0; j < sortingArray.length; j++) {**

**completeQ.add(sortingArray[j]);**

**}**

**Iterator<Job> value = completeQ.iterator();**

**double turaround=0;**

**value = completeQ.iterator();**

**while (value.hasNext()){**

**Job job = value.next();**

**output.printf("%5d %10d %15d %15d \n", job.getID(), job.getArrT(), job.getFinishTime(), (job.getFinishTime()-job.getArrT()));**

**turaround+=job.getFinishTime()-job.getArrT();**

**}**

**output.println("");**

**output.println("");**

**output.printf("System Turnaround Time = %.3f\n",turaround/completeQ.size());**

**output.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**}**

**public static void Display(int CurrentTime, PrintWriter output) {**

**output.println("<< At time " + CurrentTime + ": ");**

**output.println(" Current Available Main Memory =" + availableMem );**

**output.println(" Current Devices =" + availbleDev);**

**output.println("");**

**output.println(" Completed jobs:");**

**output.println(" ----------------");**

**output.println(" Job ID Arrival Time Finish Time Turnaround Time ");**

**output.println(" =================================================================");**

**Job [] sortingArray1= new Job[completeQ.size()];**

**CopyQueueToArray(sortingArray1 , completeQ);**

**sortArray(2,sortingArray1, completeQ);**

**completeQ.clear();**

**for (int j = 0; j < sortingArray1.length; j++) {**

**completeQ.add(sortingArray1[j]);**

**}**

**Queue<Job> Temp = new LinkedList();**

**Iterator<Job> value = completeQ.iterator();**

**value = completeQ.iterator();**

**while (value.hasNext()){**

**Job job = value.next();**

**output.printf("%5d %10d %13d %15d \n", job.getID(), job.getArrT(), job.getFinishTime(), (job.getFinishTime()-job.getArrT()));**

**}**

**output.println("\n");**

**output.println(" Hold Queue 2: ");**

**output.println( " --------------");**

**Job [] sortingArray= new Job[holdQ2.size()];**

**CopyQueueToArray(sortingArray , holdQ2);**

**sortArray(2,sortingArray, holdQ2);**

**holdQ2.clear();**

**for (int j = 0; j < sortingArray.length; j++) {**

**holdQ2.add(sortingArray[j]);**

**}**

**value = holdQ2.iterator();**

**while (value.hasNext()){**

**Job job = value.next();**

**output.printf(" %3d ", job.getID());**

**}**

**output.println("");**

**output.println("");**

**output.println("");**

**output.println(" Hold Queue 1 (Ready Queue):");**

**output.println( " ----------------");**

**output.println(" JobID NeedTime Total Execution Time ");**

**output.println(" ================================");value = holdQ1.iterator();**

**value = holdQ1.iterator();**

**while (value.hasNext()){**

**Job job = value.next();**

**output.printf("%5d %10d %10d\n", job.getID(), (job.getR()-job.getAccuredTime()), job.getAccuredTime());//run time, time accrued**

**}**

**output.println("");**

**output.println("");**

**output.println(" Process running on the CPU: ");**

**output.println(" ----------------------------");**

**output.println(" Job ID NeedTime Total Execution Time");**

**output.printf("%4d %8d %10d \n", cpu.JInCPU.getID(), (cpu.JInCPU.getR()-cpu.JInCPU.getAccuredTime()), cpu.JInCPU.getAccuredTime());//run time, time accrued**

**output.println("");**

**output.println("");**

**}**

**}**

**Job**

**public class Job {**

**private String type;**

**private int ID;**

**private int ArrT;**

**private int memory;**

**private int dev;**

**private int R;**

**private int p;**

**private double ppr;**

**private int finishTimeOutOfReady;**

**private int startTimeInReady;**

**private double pwt;**

**private int finishTime;**

**private int startTime;**

**private int accuredTime;**

**//private boolean PToSort;**

**private int turnArround=0;**

**public Job(int ArrT, int ID,int memory,int dev,int R,int p) {**

**this.ArrT=ArrT;**

**this.ID=ID;**

**this.memory= memory;**

**this.dev=dev;**

**this.p=p;**

**this.R=R;**

**this.startTime=0;**

**this.finishTime=0;**

**this.accuredTime=0;**

**this.ppr=-1;**

**//this.PToSort=true;**

**this.turnArround=this.finishTime-this.ArrT;**

**this.pwt=0;**

**}**

**public int getTurnArround() {**

**return turnArround;**

**}**

**public void setTurnArround(int turnArround) {**

**this.turnArround = turnArround;**

**}**

**//**

**// public void setPToSort(boolean PToSort) {**

**// this.PToSort = PToSort;**

**// }**

**// public boolean getPToSort() {**

**// return PToSort;**

**// }**

**public void setPpr(double ppr) {**

**this.ppr = ppr;**

**}**

**public double getPpr() {**

**return ppr;**

**}**

**public int getFinishTimeOutOfReady() {**

**return finishTimeOutOfReady;**

**}**

**public int getStartTimeInReady() {**

**return startTimeInReady;**

**}**

**public double getPwt() {**

**return pwt;**

**}**

**public void setFinishTimeOutOfReady(int finishTimeOutOfReady) {**

**this.finishTimeOutOfReady = finishTimeOutOfReady;**

**}**

**public void setStartTimeInReady(int startTimeInReady) {**

**this.startTimeInReady = startTimeInReady;**

**}**

**public void setPwt(double pwt) {**

**this.pwt = pwt;**

**}**

**public int getFinishTime() {**

**return finishTime;**

**}**

**public int getStartTime() {**

**return startTime;**

**}**

**public void setFinishTime(int finishTime) {**

**this.finishTime = finishTime;**

**}**

**public String getType() {**

**return type;**

**}**

**public void setType(String type) {**

**this.type = type;**

**}**

**public int getID() {**

**return ID;**

**}**

**public void setID(int ID) {**

**this.ID = ID;**

**}**

**public int getArrT() {**

**return ArrT;**

**}**

**public void setArrT(int ArrT) {**

**this.ArrT = ArrT;**

**}**

**public int getMemory() {**

**return memory;**

**}**

**public void setMemory(int memory) {**

**this.memory = memory;**

**}**

**public int getDev() {**

**return dev;**

**}**

**public void setDev(int dev) {**

**this.dev = dev;**

**}**

**public int getR() {**

**return R;**

**}**

**public void setR(int R) {**

**this.R = R;**

**}**

**public int getP() {**

**return p;**

**}**

**public void setP(int p) {**

**this.p = p;**

**}**

**public void setStartTime(int startTime) {**

**this.startTime = startTime;**

**}**

**public int getAccuredTime() {**

**return accuredTime;**

**}**

**public void setAccuredTime(int accuredTime) {**

**this.accuredTime = accuredTime;**

**}**

**}**

**CPU**

public class CPU {

Job JInCPU;

boolean free;

public CPU(Job JInCPU, boolean free) {

this.JInCPU = JInCPU;

this.free = free;

}

public Job getJobs() {

return JInCPU;

}

public void setJobs(Job jobs) {

this.JInCPU = jobs;

}

public boolean isFree() {

return free;

}

public void setFree(boolean free) {

this.free = free;

}

}

**osStaticMain (Static)**

import java.io.FileNotFoundException;

import java.io.PrintWriter;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

//Aya Kazzaz,Hadeel aloufi,Salwa Abbara,Mona Hafez,Aisha Baskran,windows10,version1909(OS Build 18363.1256)

//processor:Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz

//Installed memory(RAM): 8.00 GB(7.90 GB usable)

//compiler name and version:java version "1.8.0\_191" ,Java(TM) SE Runtime Environment

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author WinDows

\*/

public class osStaticMain {

static int memorySize;

static int devices;

static int time;

static Queue<Job>allJobQ=new LinkedList();

static Queue<Job>holdQ1=new LinkedList();

static Queue<Job>holdQ2=new LinkedList();

static Queue<Job>sortedHoldQ2=new LinkedList();

static CPU cpu=new CPU(null,true);

static Queue<Job>completeQ=new LinkedList();

static int cStartTime=0;

static int availableMem=0;

static int availbleDev=0;

static int cThrehold=0;

static int numOfJob=0;

static int Quantum=5+4;

public static int i;

public static int e;

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

java.io.File file = new java.io.File("input1.txt");

// java.io.File file = new java.io.File("input2.txt");you can change between these 3 input files to check outputs files

//java.io.File file = new java.io.File("input3.txt");

java.io.File outputFile = new java.io.File("output.txt");

PrintWriter output=new PrintWriter(outputFile);//print writer to write to the outputfile

Scanner input=new Scanner(file);

while (input.hasNext()) {

String command1=input.next();

time=input.nextInt();

memorySize=Integer.parseInt(input.next().substring(2));

availableMem=memorySize;

devices=Integer.parseInt(input.next().substring(2));

availbleDev=devices;

Job job;

numOfJob=0;

int number = 0;

do{

String command2=input.next();

if(command2.equals("A")){

int ArrT= Integer.parseInt(input.next());

int ID=Integer.parseInt(input.next().substring(2));

int memory=Integer.parseInt(input.next().substring(2));

int dev=Integer.parseInt(input.next().substring(2));

int R=Integer.parseInt(input.next().substring(2));

int p=Integer.parseInt(input.next().substring(2));

if(memory<=memorySize&&dev<=devices){

job=new Job(ArrT,ID,memory,dev,R,p);

allJobQ.add(job);

numOfJob++;//counter to count the number of jobs

}

}else if(command2.equals("D")){

int time1=input.nextInt();

if(time1<999999){

job=new Job(time1,-1,-1,-1,-1,-1);

allJobQ.add(job);

}

else{

break;

}

}

} while(input.hasNext());

Job j=allJobQ.poll();

int currentTime=j.getArrT();

availableMem-=j.getMemory();

getIntoCPU(j,currentTime);

i = 0;

e = 0;

do {

if (!(allJobQ.isEmpty())) {

i = allJobQ.peek().getArrT();

} else {

i = Integer.MAX\_VALUE;

}

if (cpu.isFree()==false) {

e = cpu.JInCPU.getFinishTime();

} else {

e = Integer.MAX\_VALUE;

}

currentTime = Math.min(i, e);

if (i < e) {

External(currentTime, output);

} else if (e < i) {

Internal(currentTime);

} else {

Internal(currentTime);

External(currentTime, output);

}

} while (completeQ.size() != numOfJob);

DisplayFinalState(currentTime,output);//displaying the final state

completeQ.clear();

}

input.close();

output.close();

}

public static void External(int currentTime, PrintWriter output){

if(!allJobQ.isEmpty()){

if(allJobQ.peek().getID()== -1){

allJobQ.poll();

Display(currentTime,output);

}else {

Job job=allJobQ.poll();

if(job.getMemory()>availableMem || job.getDev()>availbleDev){

holdQ2.add(job);

}else if(job.getMemory()<=availableMem && job.getDev()<=availbleDev){

holdQ1.add(job);

availableMem-=job.getMemory();

availbleDev-=job.getDev();

}

}

}

}

public static void Internal(int CurrentTime) {

getOutOfCPU(CurrentTime);

if (!holdQ1.isEmpty()) {

getIntoCPU(holdQ1.poll(), CurrentTime);

}

}

public static void Get\_OUT\_Hold(int currentTime) {

//timer start

if(!holdQ2.isEmpty()){

Job [] sortingArray= new Job[holdQ2.size()];

Job o;

Iterator<Job> value =holdQ2.iterator();

while (value.hasNext()){

o=value.next();

}

value =holdQ1.iterator();

while (value.hasNext()){

o=value.next();

}

CopyQueueToArray(sortingArray , holdQ2);

for (int j = 0; j < sortingArray.length; j++) {

}

sortArray(2,sortingArray, holdQ2);

holdQ2.clear();

for (int j = 0; j < sortingArray.length; j++) {

holdQ2.add(sortingArray[j]);

}

int sizeq2=holdQ2.size();

for(int i=0;i<sizeq2;i++){

Job temp=holdQ2.poll();

if(temp.getMemory()<=availableMem && temp.getDev()<=availbleDev){

holdQ1.add(temp);

availableMem-=temp.getMemory();

availbleDev-=temp.getDev();

}else{

holdQ2.add(temp);

}

}

}

}

public static void CopyQueueToArray(Job arr[] , Queue<Job> h){

int k =0;

Job o;

Iterator<Job> value = h.iterator();

while (value.hasNext()){

o=value.next();

arr[k]=o;

k++;

}

}

public static void sortArray(int type, Job arr[], Queue<Job> h) {

Job temp;

if(type==1){

for (int i = 0; i < arr.length; i++)

{

for (int j = i + 1; j < arr.length; j++)

{

if (arr[i].getID() > arr[j].getID())

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}else{

for (int i = 0; i < arr.length; i++)

{

for (int j = i + 1; j < arr.length; j++)

{

if (arr[i].getP() < arr[j].getP())

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}else if(arr[i].getP() == arr[j].getP()){//if same jobs have the same priority then check the IDs

if (arr[i].getID() > arr[j].getID())

{

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

}

h.clear();

for (int j = 0; j < arr.length; j++) {

h.add(arr[j]);

}

}

public static void getIntoCPU(Job job,int currentTime){

cpu.setJobs(job);

cpu.setFree(false);

cpu.JInCPU.setStartTime(currentTime);

int RemainingTime=cpu.JInCPU.getR()-cpu.JInCPU.getAccuredTime();

if(RemainingTime>= Quantum){

cpu.JInCPU.setFinishTime(currentTime+Quantum);

}else{

cpu.JInCPU.setFinishTime(currentTime+RemainingTime);

}

}

public static void getOutOfCPU(int CurrentTime){

if (cpu.isFree()== false) {

cpu.JInCPU.setAccuredTime(cpu.JInCPU.getAccuredTime()+(cpu.JInCPU.getFinishTime() - cpu.JInCPU.getStartTime()));

if (cpu.JInCPU.getAccuredTime() == cpu.JInCPU.getR()) {

availableMem += cpu.JInCPU.getMemory();

availbleDev += cpu.JInCPU.getDev();

completeQ.add(cpu.JInCPU);

cpu.setFree(true);

cpu.JInCPU=null;

if(!holdQ2.isEmpty()){//when the job left the cpu it release its memory and devices so we can move job from holdQ2 to holdQ1

Get\_OUT\_Hold(CurrentTime);

}

}else {//if the process not terminated

holdQ1.add(cpu.JInCPU);

cpu.setFree(true);

}

}

}

public static void DisplayFinalState(int currentTime,PrintWriter output){

//print all the system

output.println("<< Final state of system: ");

output.println(" Current Available Main Memory = "+availableMem);

output.println(" Current Devices = "+availbleDev);

output.println("");

output.println(" Completed jobs:");

output.println(" ----------------");

output.println(" Job ID Arrival Time Finish Time Turnaround Time ");

output.println(" ===============================================================");

Job [] sortingArray= new Job[completeQ.size()];

CopyQueueToArray(sortingArray , completeQ);

sortArray(1,sortingArray, completeQ);

completeQ.clear();

for (int j = 0; j < sortingArray.length; j++) {

//sortingArray[j].setPToSort(true);

//System.out.println("Array---------------"+sortingArray[j].getID());

completeQ.add(sortingArray[j]);

}

Iterator<Job> value = completeQ.iterator();

double turaround=0;

value = completeQ.iterator();

while (value.hasNext()){

Job job = value.next();

output.printf("%5d %10d %15d %15d \n", job.getID(), job.getArrT(), job.getFinishTime(), (job.getFinishTime()-job.getArrT()));

turaround+=job.getFinishTime()-job.getArrT();

}

//printJobscompleteQue(sortingArray);

// printJobscompleteQue();

output.println("");

output.println("");

output.printf("System Turnaround Time = %.3f\n",turaround/completeQ.size());

output.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

public static void Display(int CurrentTime, PrintWriter output) {

output.println("<< At time " + CurrentTime + ": ");

output.println(" Current Available Main Memory =" + availableMem );

output.println(" Current Devices = " + availbleDev);

output.println("");

output.println(" Completed jobs:");

output.println(" ----------------");

output.println(" Job ID Arrival Time Finish Time Turnaround Time ");

output.println(" ================================================================");

// Queue<Job> Temp = new LinkedList();

Job [] sortingArray1= new Job[completeQ.size()];

CopyQueueToArray(sortingArray1 , completeQ);

sortArray(1,sortingArray1, completeQ);

completeQ.clear();

for (int j = 0; j < sortingArray1.length; j++) {

completeQ.add(sortingArray1[j]);

}

Iterator<Job> value = completeQ.iterator();

value = completeQ.iterator();

while (value.hasNext()){

Job job = value.next();

output.printf("%5d %10d %13d %15d \n", job.getID(), job.getArrT(), job.getFinishTime(), (job.getFinishTime()-job.getArrT()));

}

//////////////////////////////////

output.println("\n");

output.println(" Hold Queue 2: ");

output.println( " --------------");

Job [] sortingArray= new Job[holdQ2.size()];

CopyQueueToArray(sortingArray , holdQ2);

sortArray(1,sortingArray, holdQ2);

holdQ2.clear();

for (int j = 0; j < sortingArray.length; j++) {

holdQ2.add(sortingArray[j]);

}

value = holdQ2.iterator();

while (value.hasNext()){

Job job = value.next();

output.printf(" %3d ", job.getID());

}

output.println("");

output.println("");

output.println("");

output.println(" Hold Queue 1 (Ready Queue): ");

output.println( " ----------------");

output.println(" JobID NeedTime Total Execution Time ");

output.println(" ===============================");value = holdQ1.iterator();

value = holdQ1.iterator();

while (value.hasNext()){

Job job = value.next();

output.printf("%5d %10d %10d\n", job.getID(), (job.getR()-job.getAccuredTime()), job.getAccuredTime());//run time, time accrued

}

output.println("");

output.println("");

output.println(" Process running on the CPU: ");

output.println(" ----------------------------");

output.println(" Job ID NeedTime Total Execution Time");

output.printf("%5d %10d %10d \n", cpu.JInCPU.getID(), (cpu.JInCPU.getR()-cpu.JInCPU.getAccuredTime()), cpu.JInCPU.getAccuredTime());//run time, time accrued

output.println("");

output.println("");

}

}

**Job**

public class Job {

private String type;

private int ID;

private int ArrT;

private int memory;

private int dev;

private int R;

private int p;

private double ppr;

private int finishTimeOutOfReady;

private int startTimeInReady;

private double pwt;

private int finishTime;

private int startTime;

private int accuredTime;

//private boolean PToSort;

private int turnArround=0;

public Job(int ArrT, int ID,int memory,int dev,int R,int p) {

this.ArrT=ArrT;

this.ID=ID;

this.memory= memory;

this.dev=dev;

this.p=p;

this.R=R;

this.startTime=0;

this.finishTime=0;

this.accuredTime=0;

this.ppr=-1;

//this.PToSort=true;

this.turnArround=this.finishTime-this.ArrT;

this.pwt=0;

}

public int getTurnArround() {

return turnArround;

}

public void setTurnArround(int turnArround) {

this.turnArround = turnArround;

}

//

// public void setPToSort(boolean PToSort) {

// this.PToSort = PToSort;

// }

// public boolean getPToSort() {

// return PToSort;

// }

public void setPpr(double ppr) {

this.ppr = ppr;

}

public double getPpr() {

return ppr;

}

public int getFinishTimeOutOfReady() {

return finishTimeOutOfReady;

}

public int getStartTimeInReady() {

return startTimeInReady;

}

public double getPwt() {

return pwt;

}

public void setFinishTimeOutOfReady(int finishTimeOutOfReady) {

this.finishTimeOutOfReady = finishTimeOutOfReady;

}

public void setStartTimeInReady(int startTimeInReady) {

this.startTimeInReady = startTimeInReady;

}

public void setPwt(double pwt) {

this.pwt = pwt;

}

public int getFinishTime() {

return finishTime;

}

public int getStartTime() {

return startTime;

}

public void setFinishTime(int finishTime) {

this.finishTime = finishTime;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public int getID() {

return ID;

}

public void setID(int ID) {

this.ID = ID;

}

public int getArrT() {

return ArrT;

}

public void setArrT(int ArrT) {

this.ArrT = ArrT;

}

public int getMemory() {

return memory;

}

public void setMemory(int memory) {

this.memory = memory;

}

public int getDev() {

return dev;

}

public void setDev(int dev) {

this.dev = dev;

}

public int getR() {

return R;

}

public void setR(int R) {

this.R = R;

}

public int getP() {

return p;

}

public void setP(int p) {

this.p = p;

}

public void setStartTime(int startTime) {

this.startTime = startTime;

}

public int getAccuredTime() {

return accuredTime;

}

public void setAccuredTime(int accuredTime) {

this.accuredTime = accuredTime;

}

}

**CPU**

public class CPU {

Job JInCPU;

// private int Quantum;

boolean free;

public CPU(Job JInCPU, boolean free) {

this.JInCPU = JInCPU;

this.free = free;

}

public Job getJobs() {

return JInCPU;

}

public void setJobs(Job jobs) {

this.JInCPU = jobs;

}

public boolean isFree() {

return free;

}

public void setFree(boolean free) {

this.free = free;

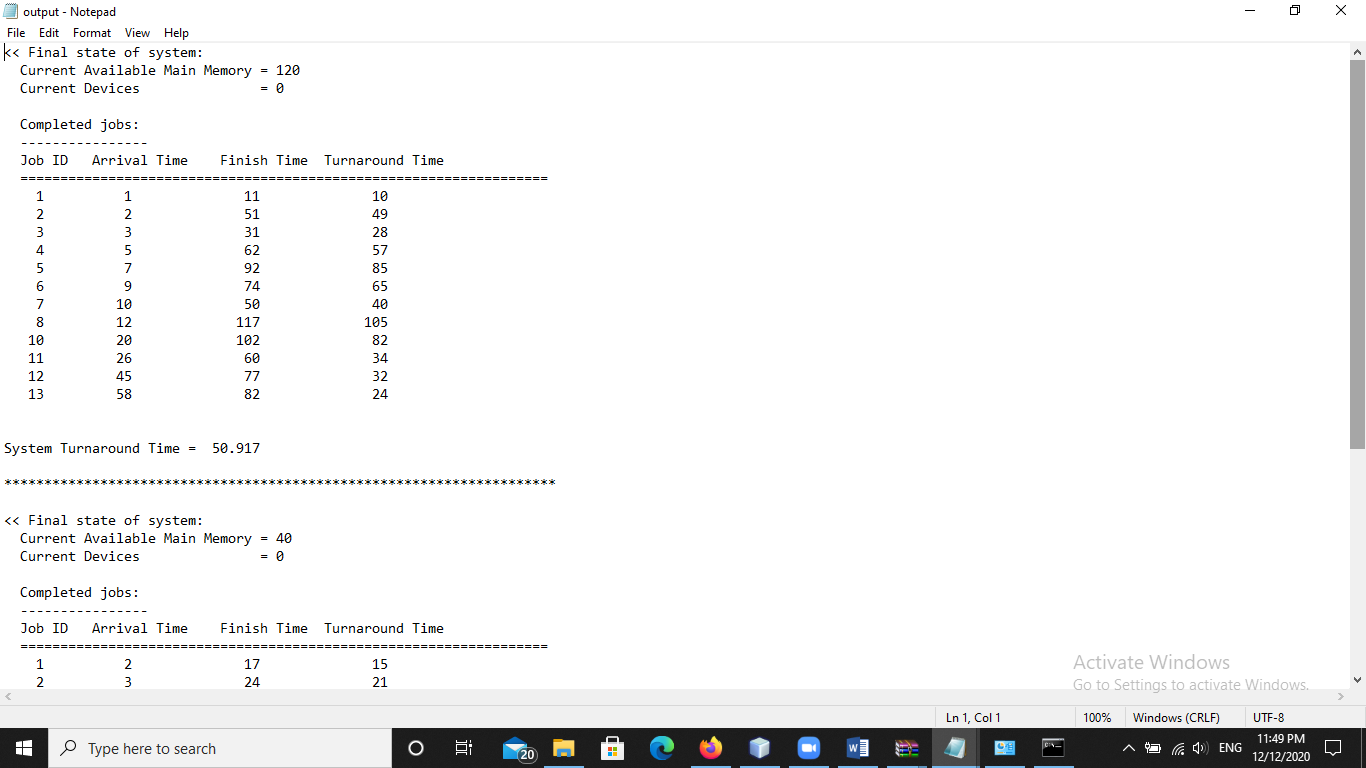
}

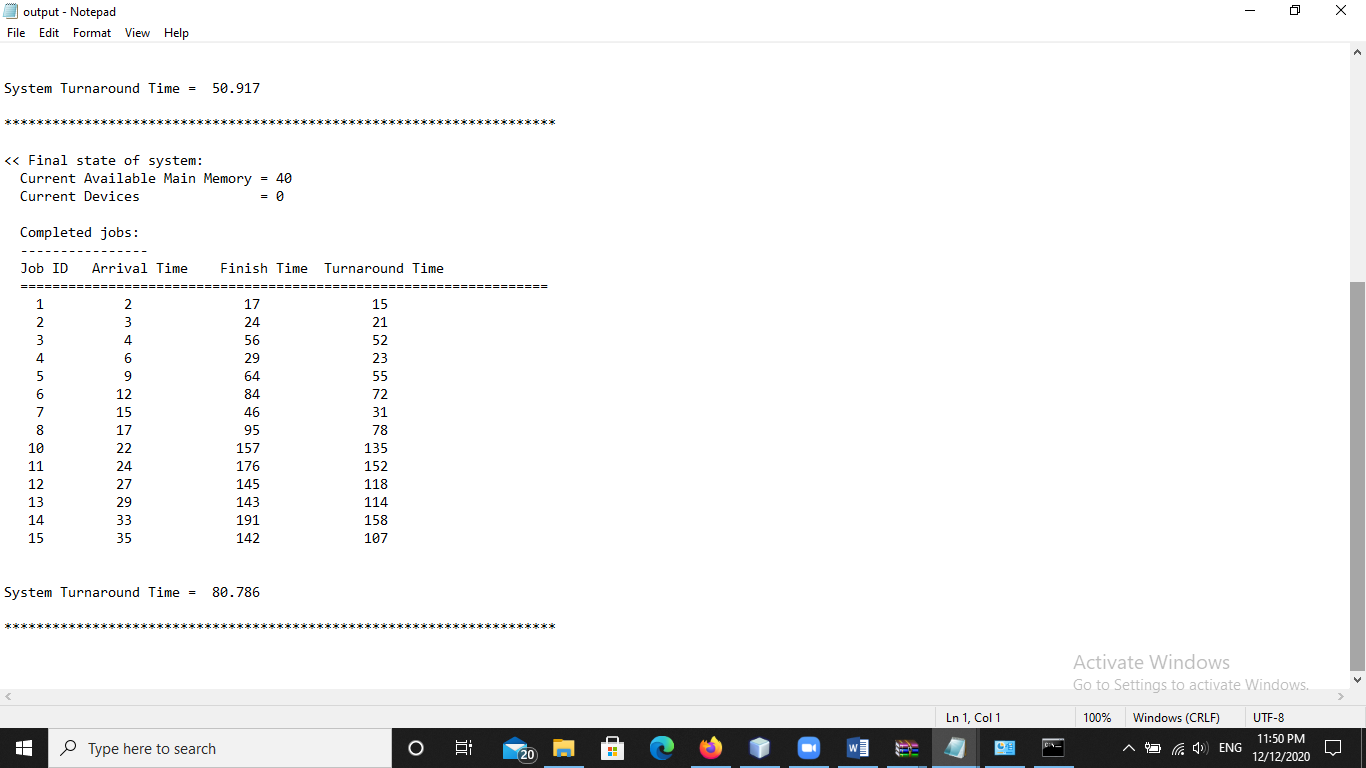
}

1. **A copy of the program output on each test data file**

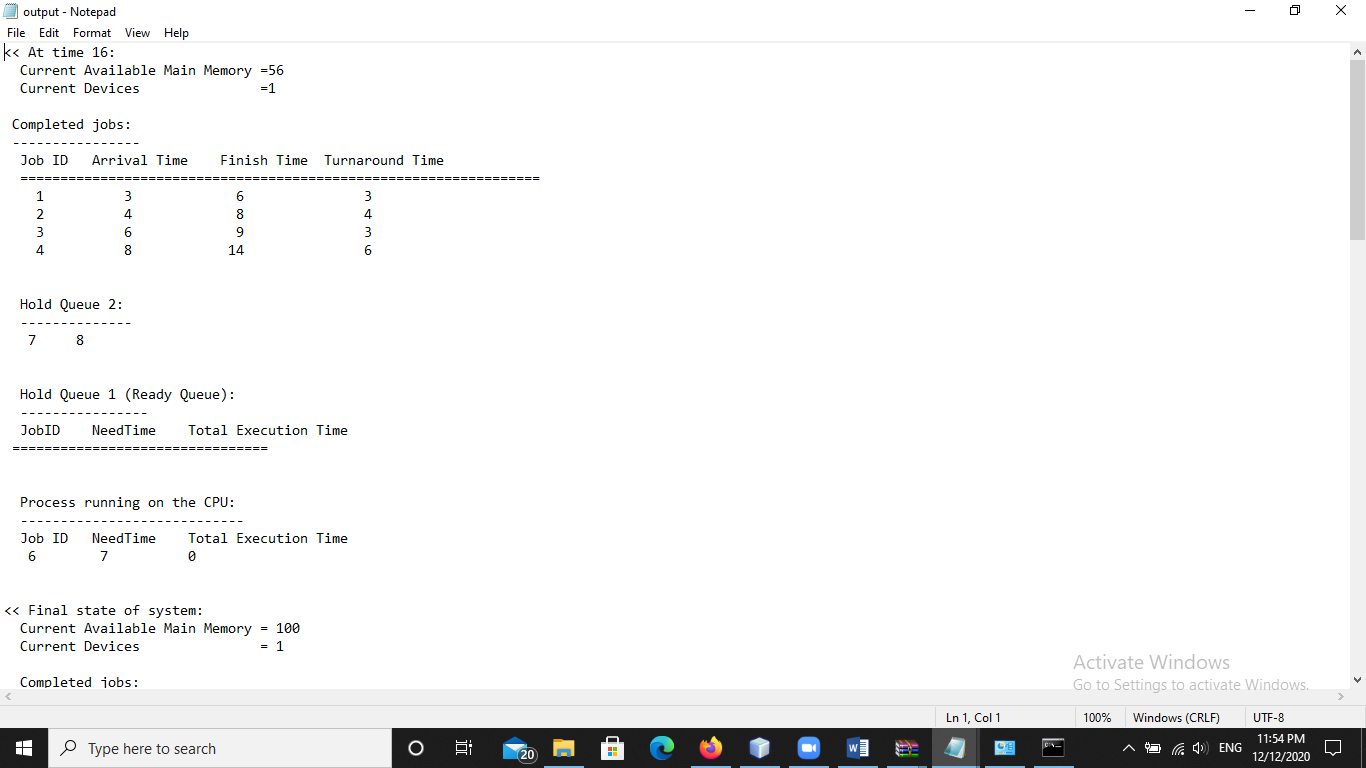
**Screenshots for dynamic roundrobin**

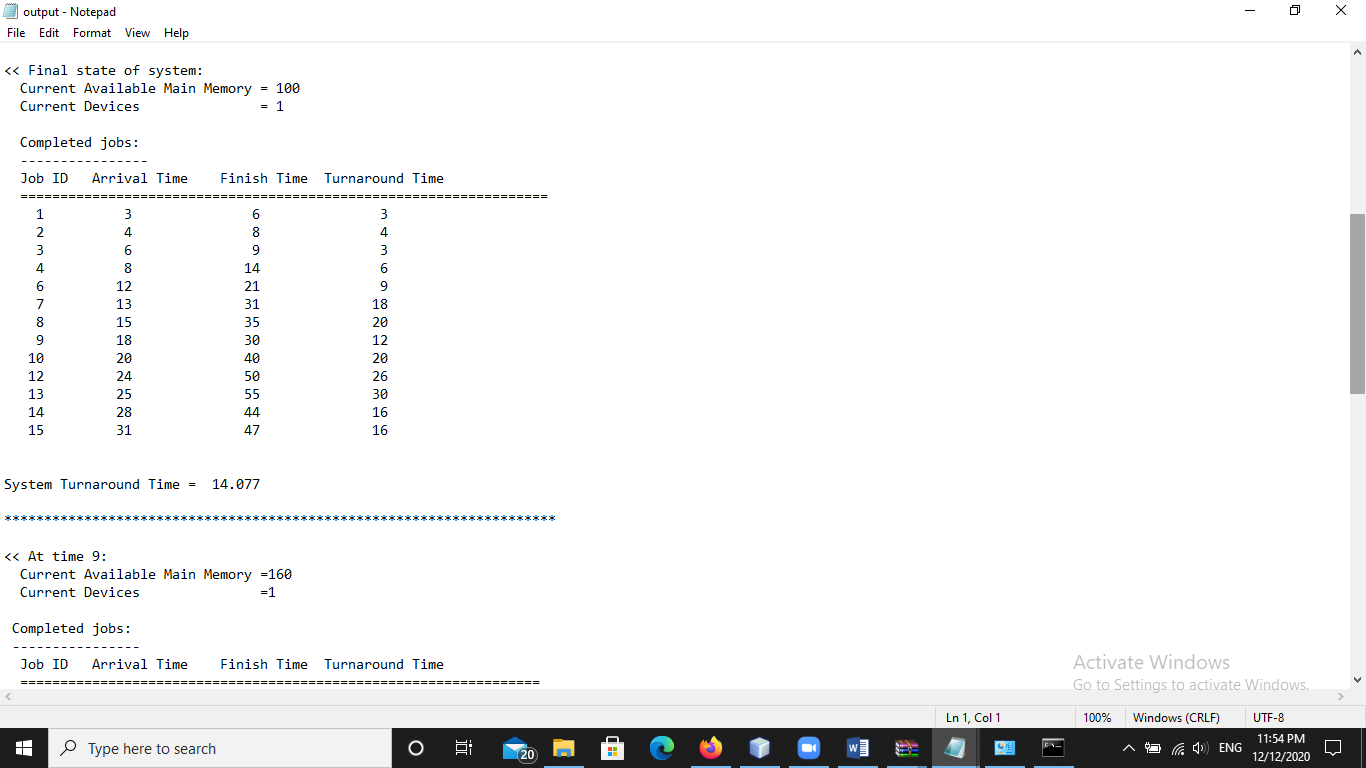
* **Input 1**

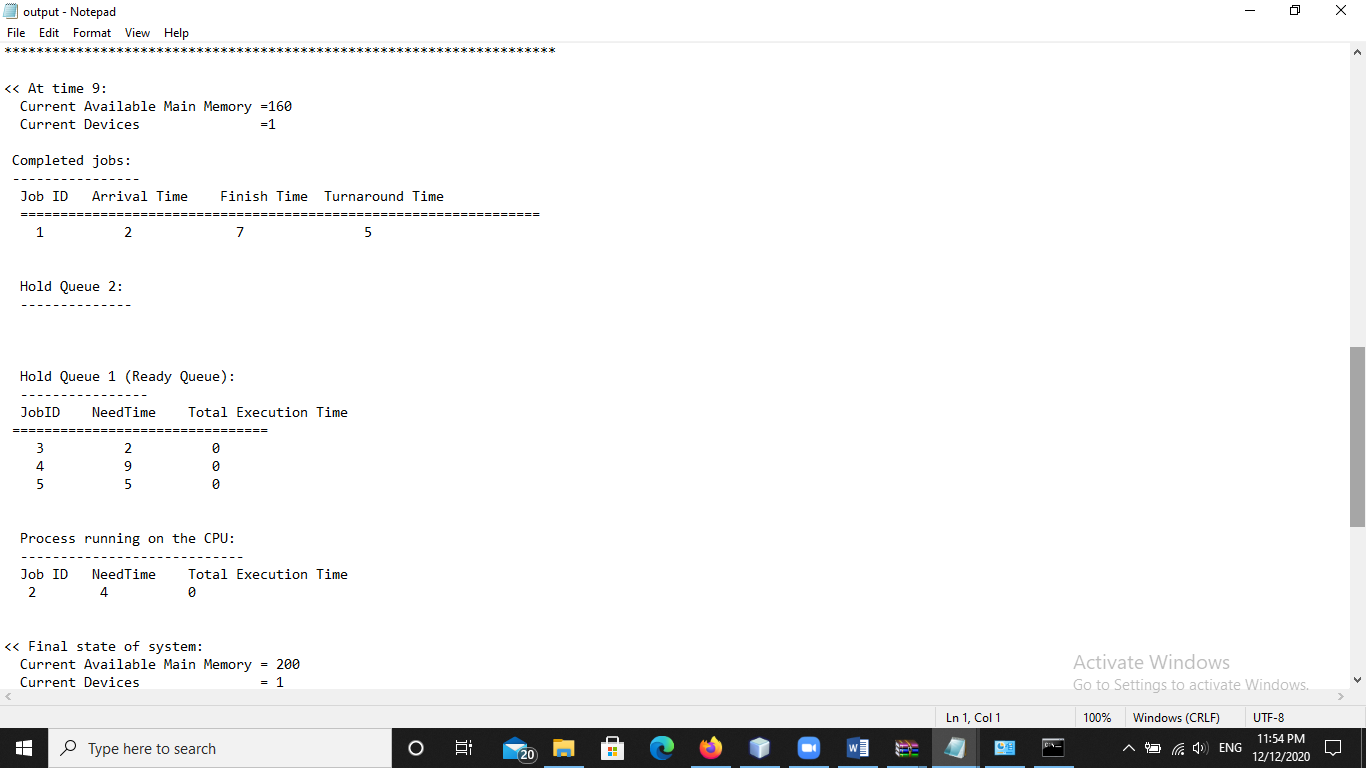
****

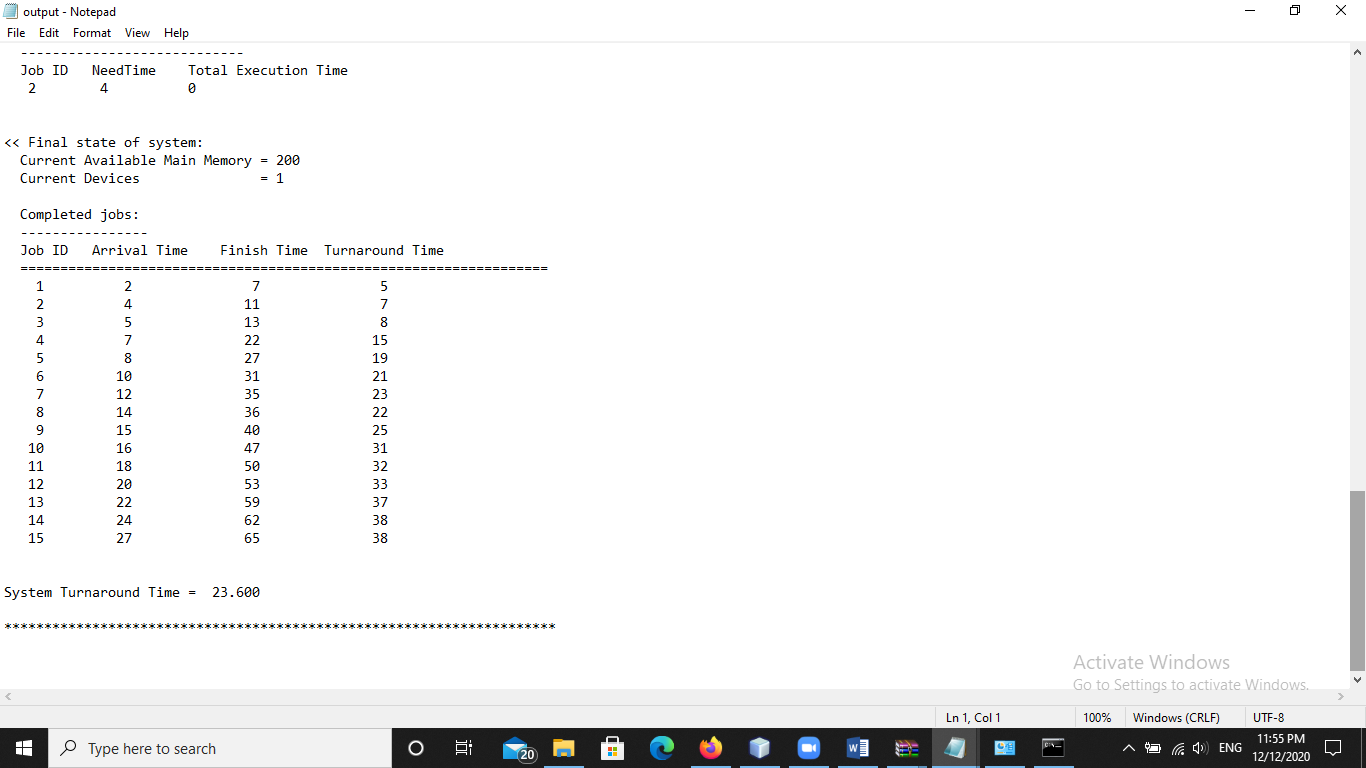
****

* **Input 2**

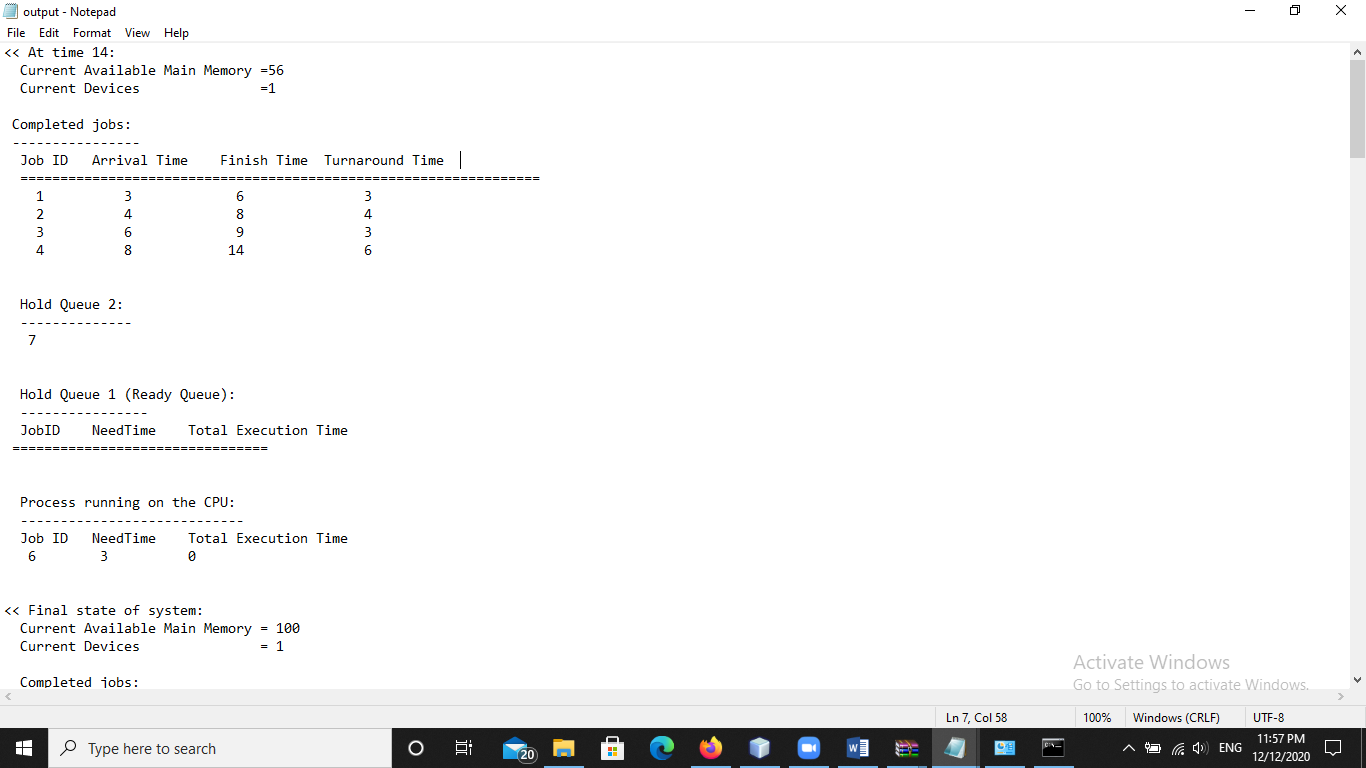
****

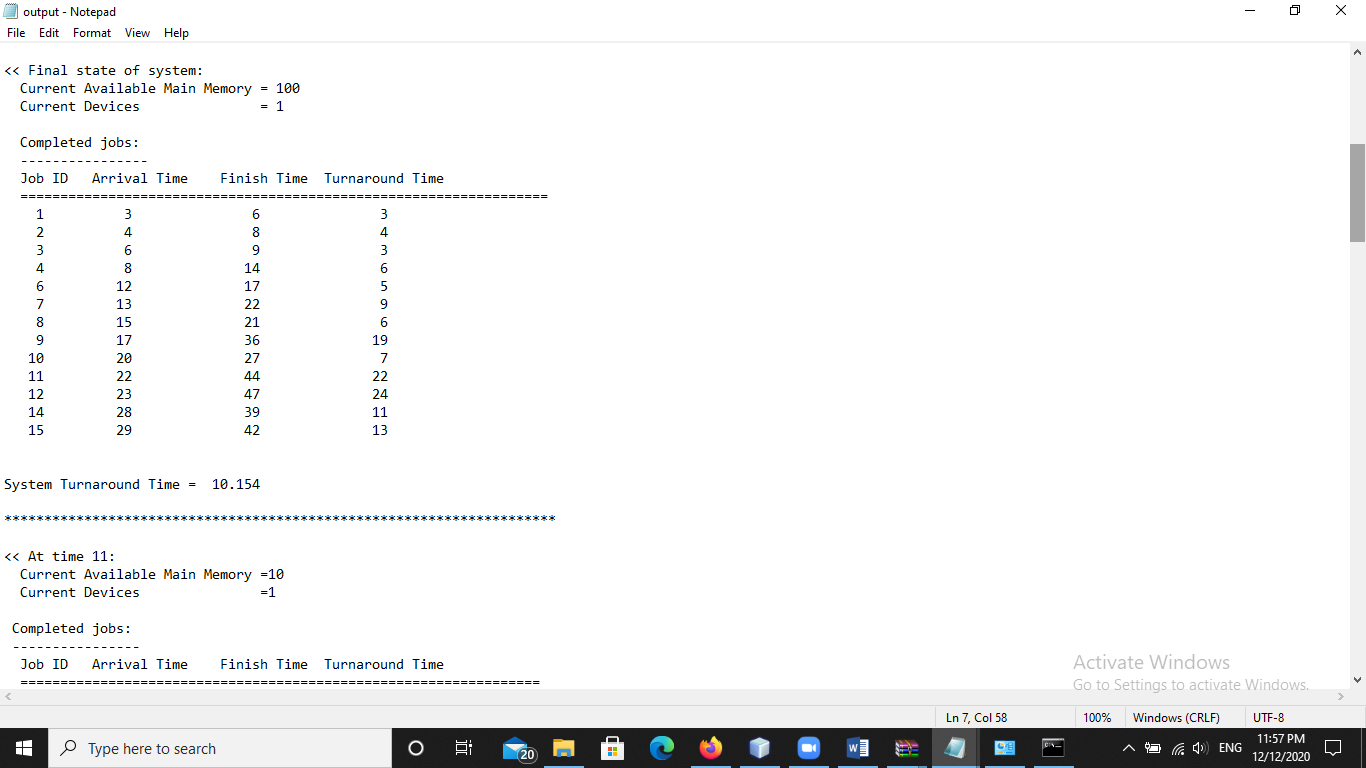
****

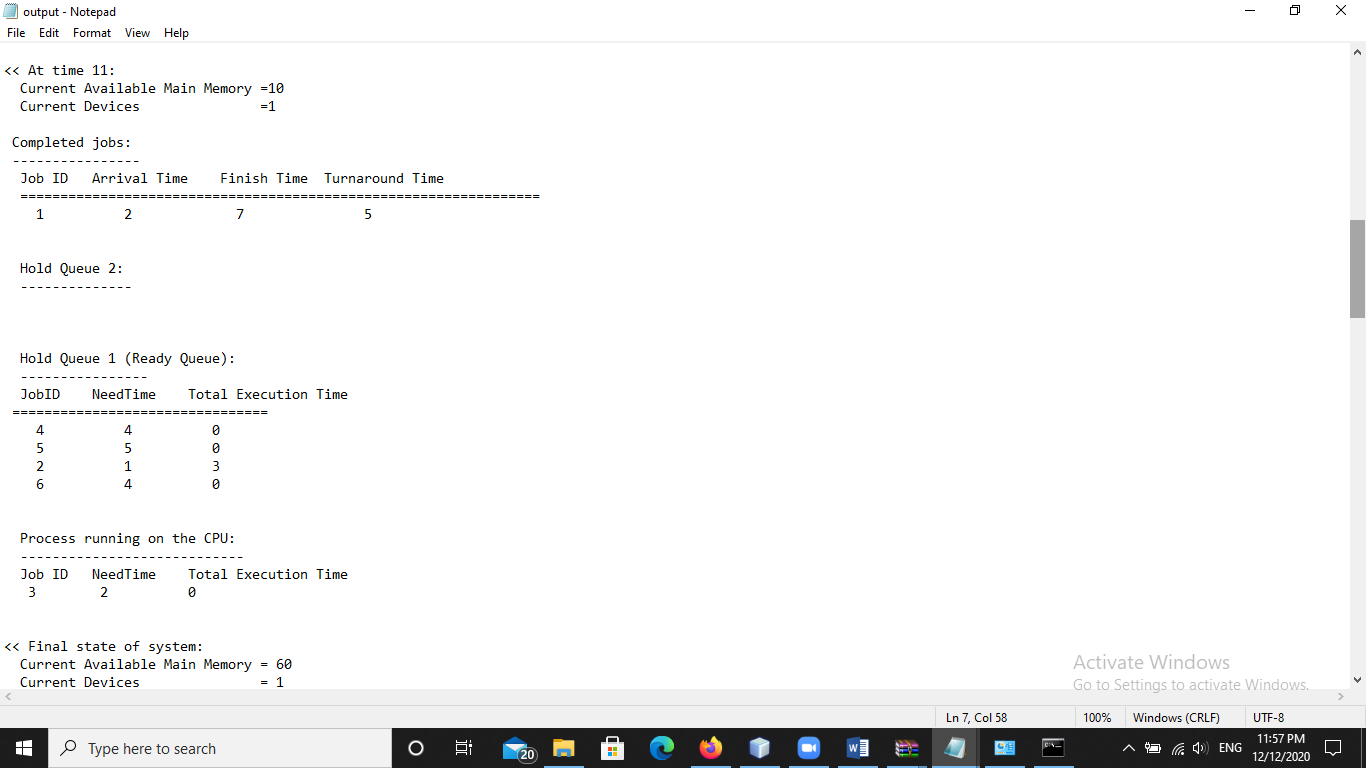
****

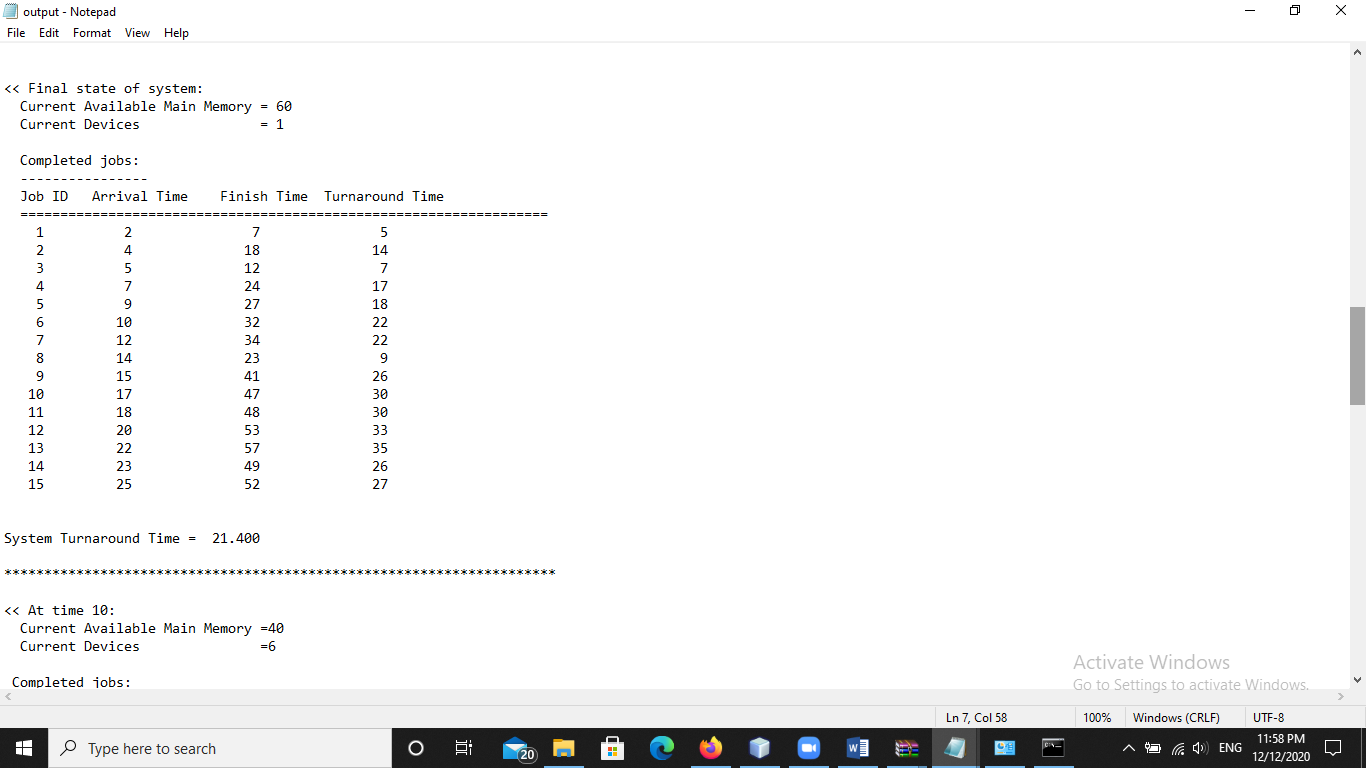
****

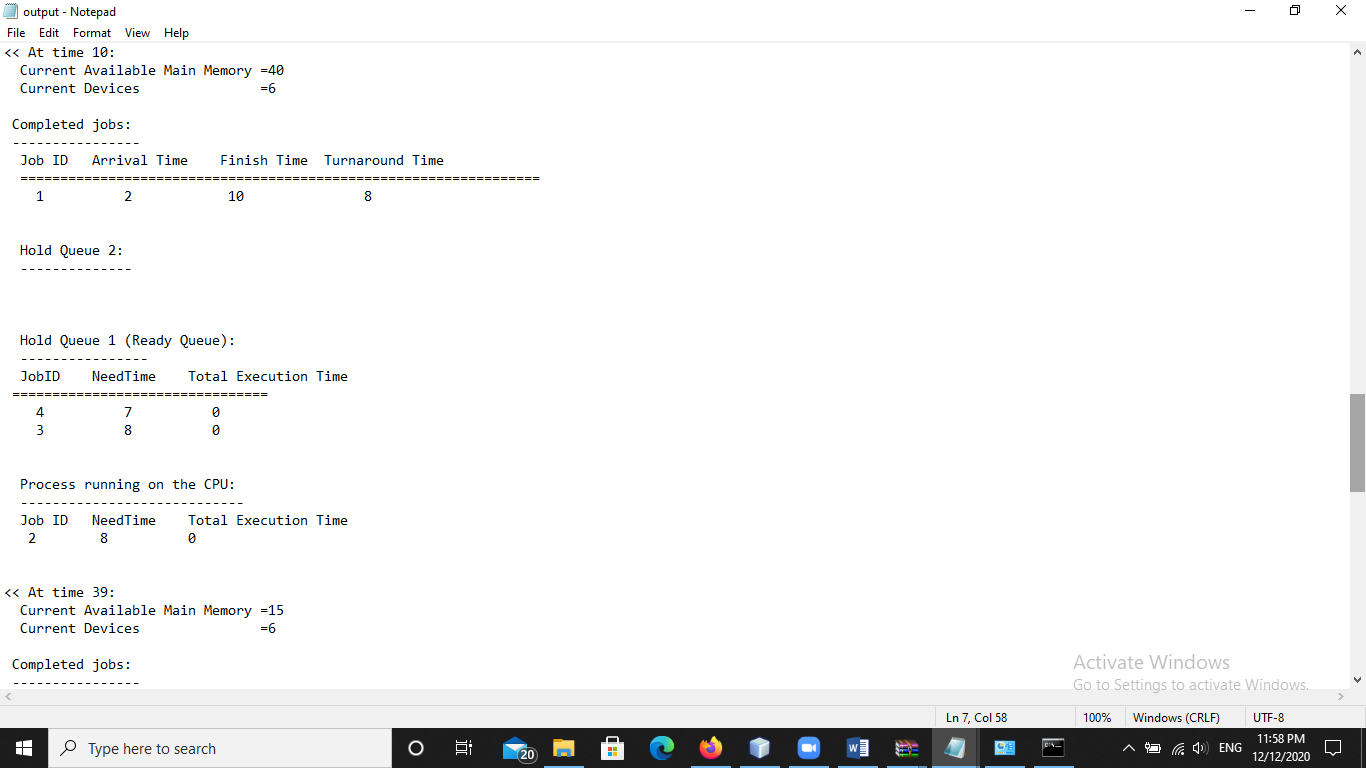
* **Input 3**

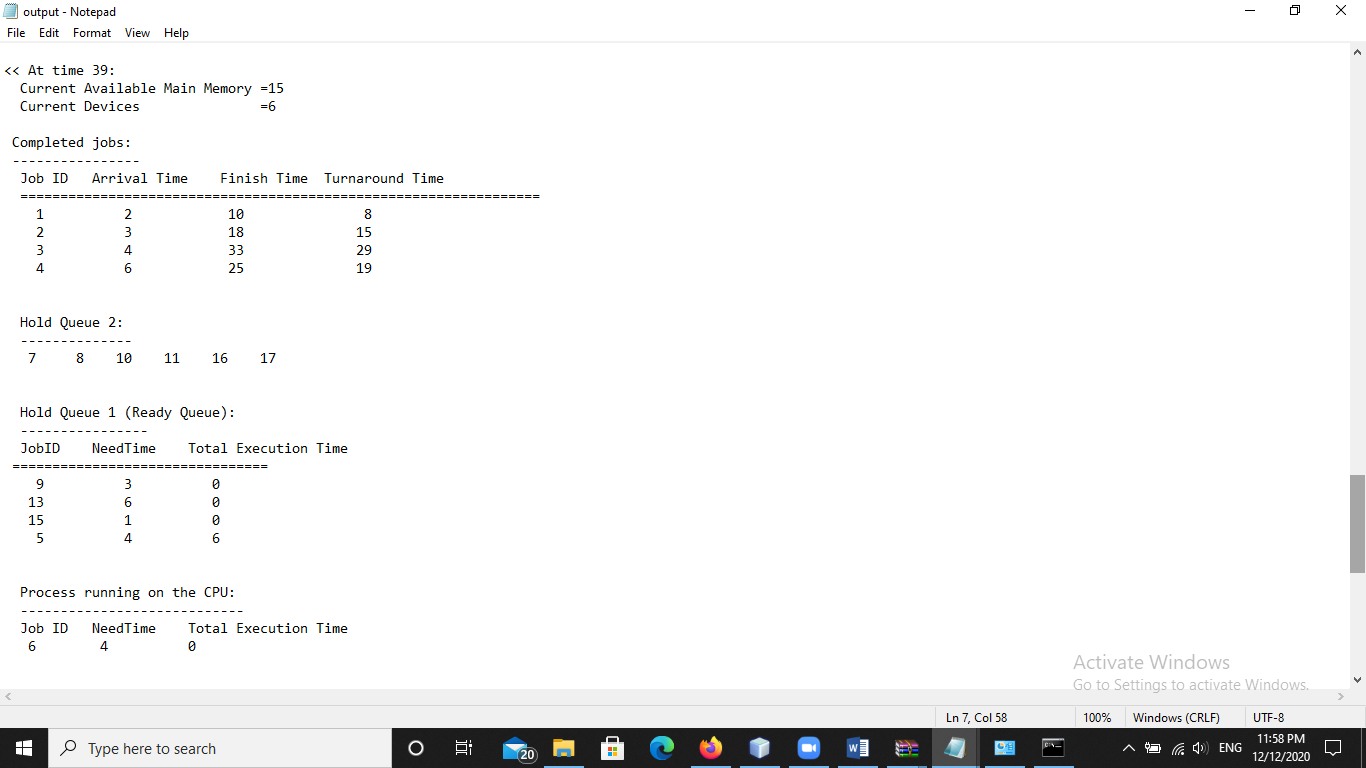
****

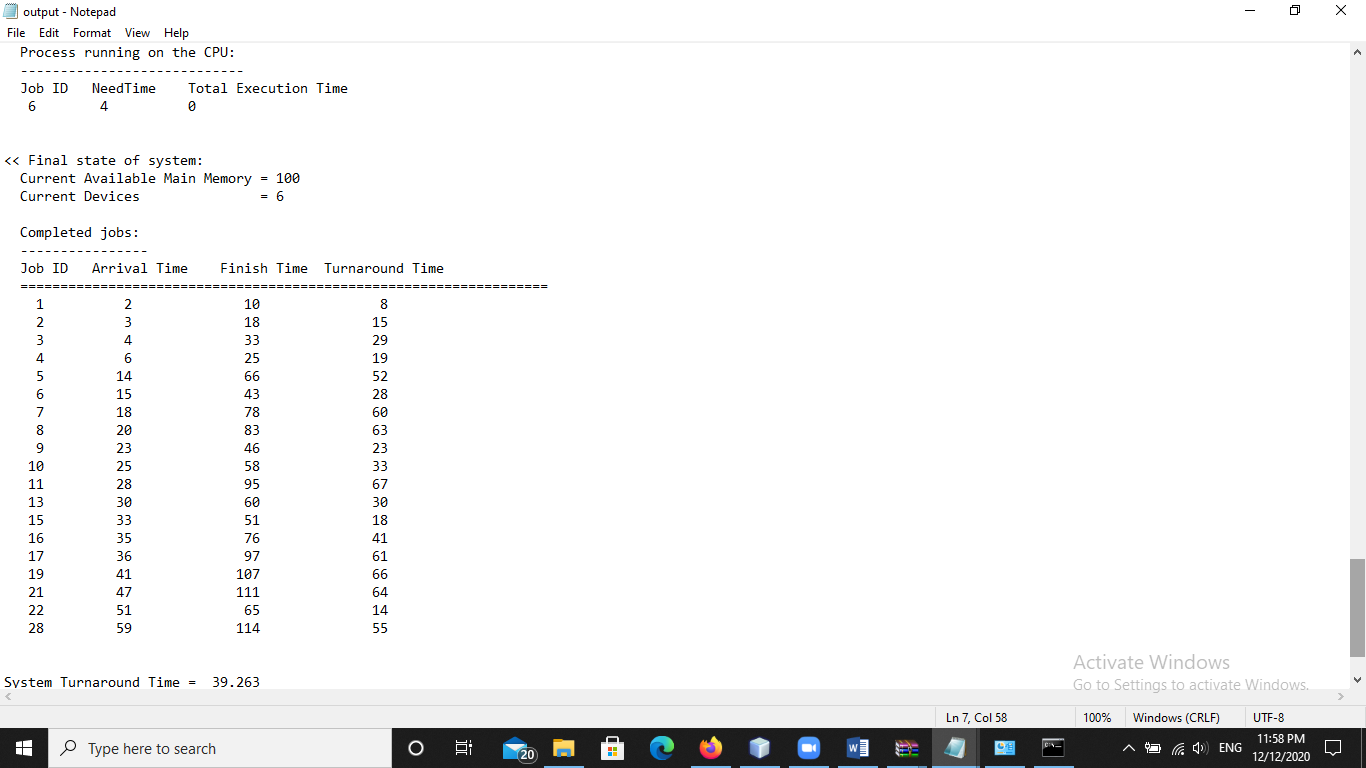
****

****

****

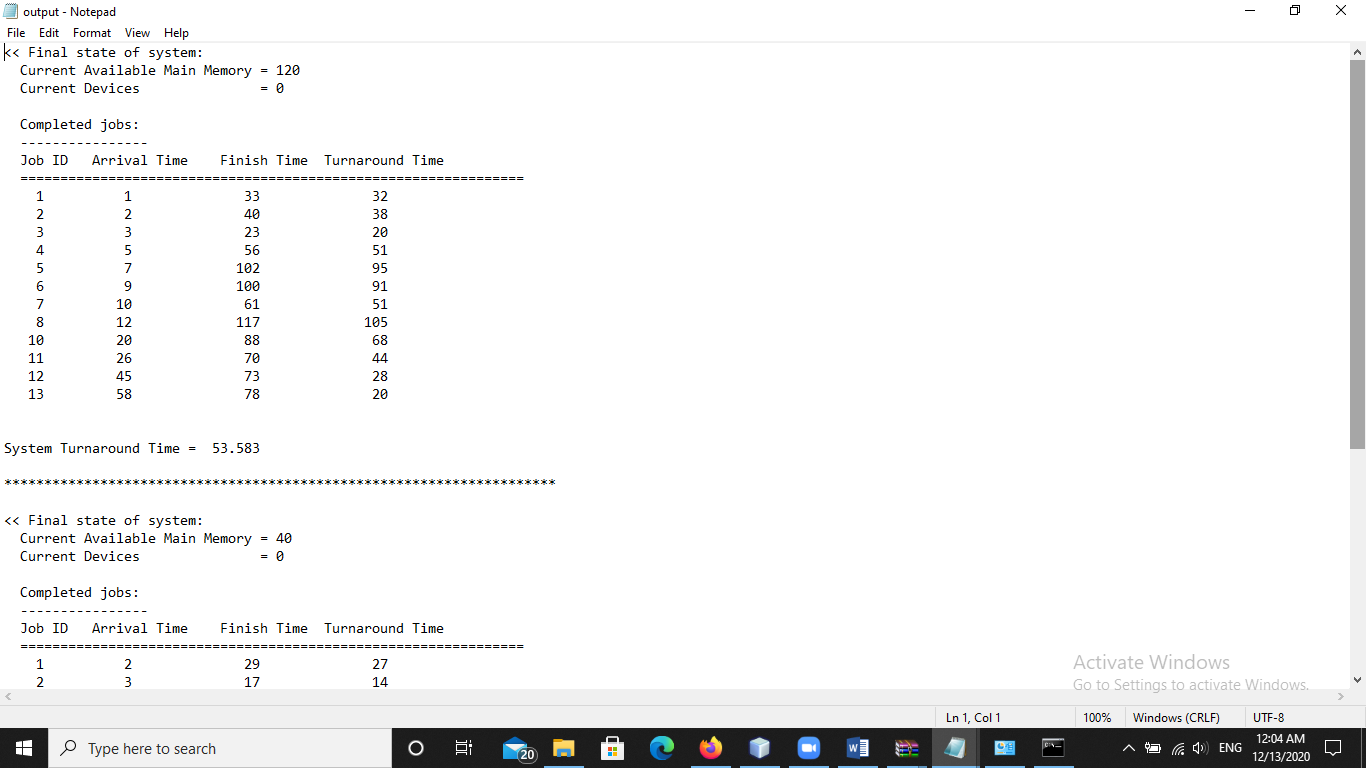
****

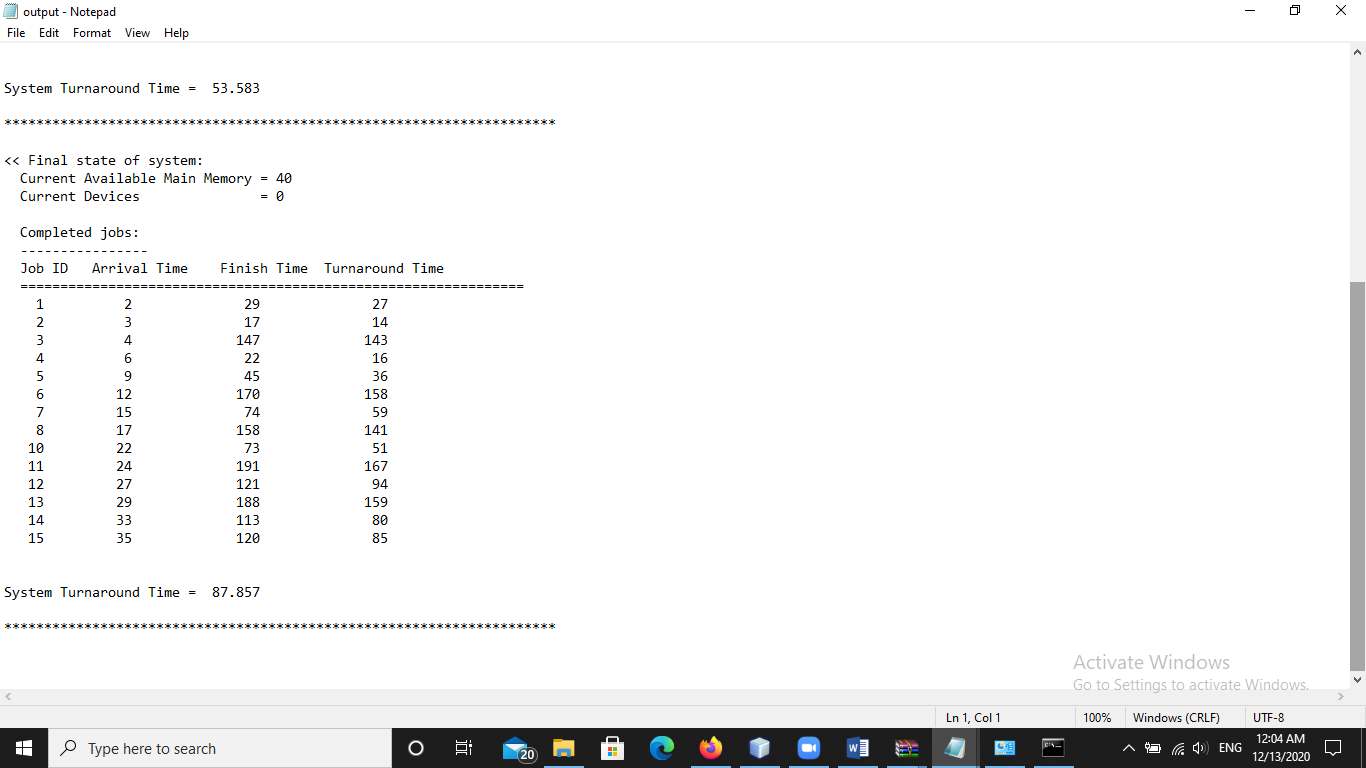
****

****

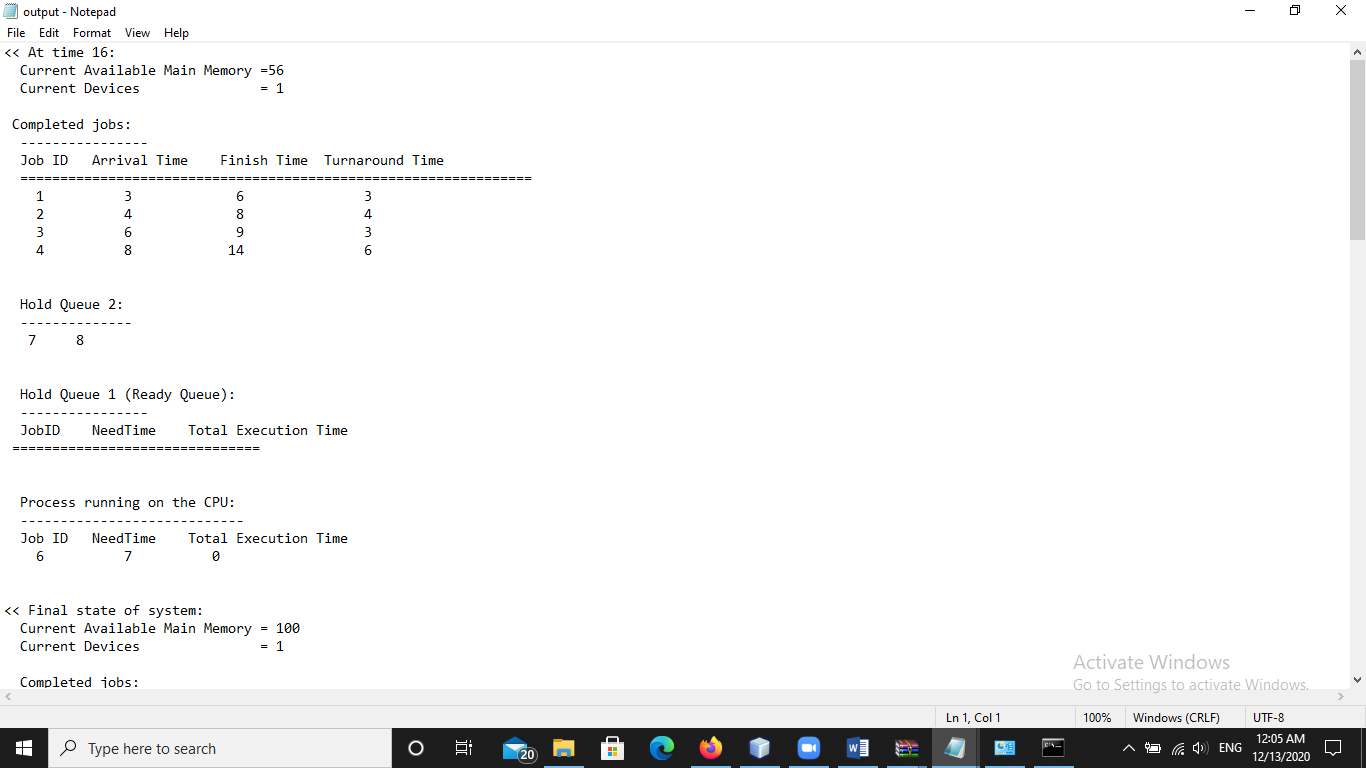
**Screenshots for brute force round robin**

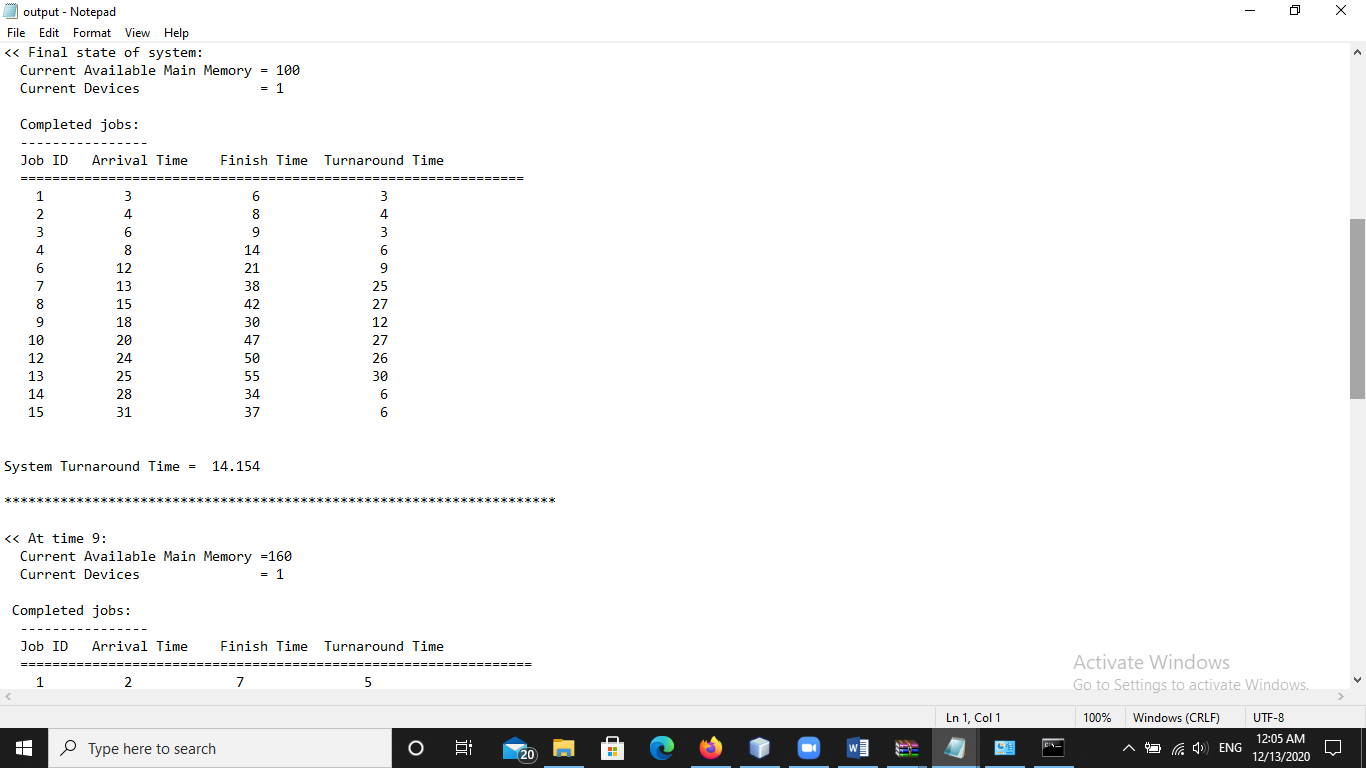
* **Input 1**

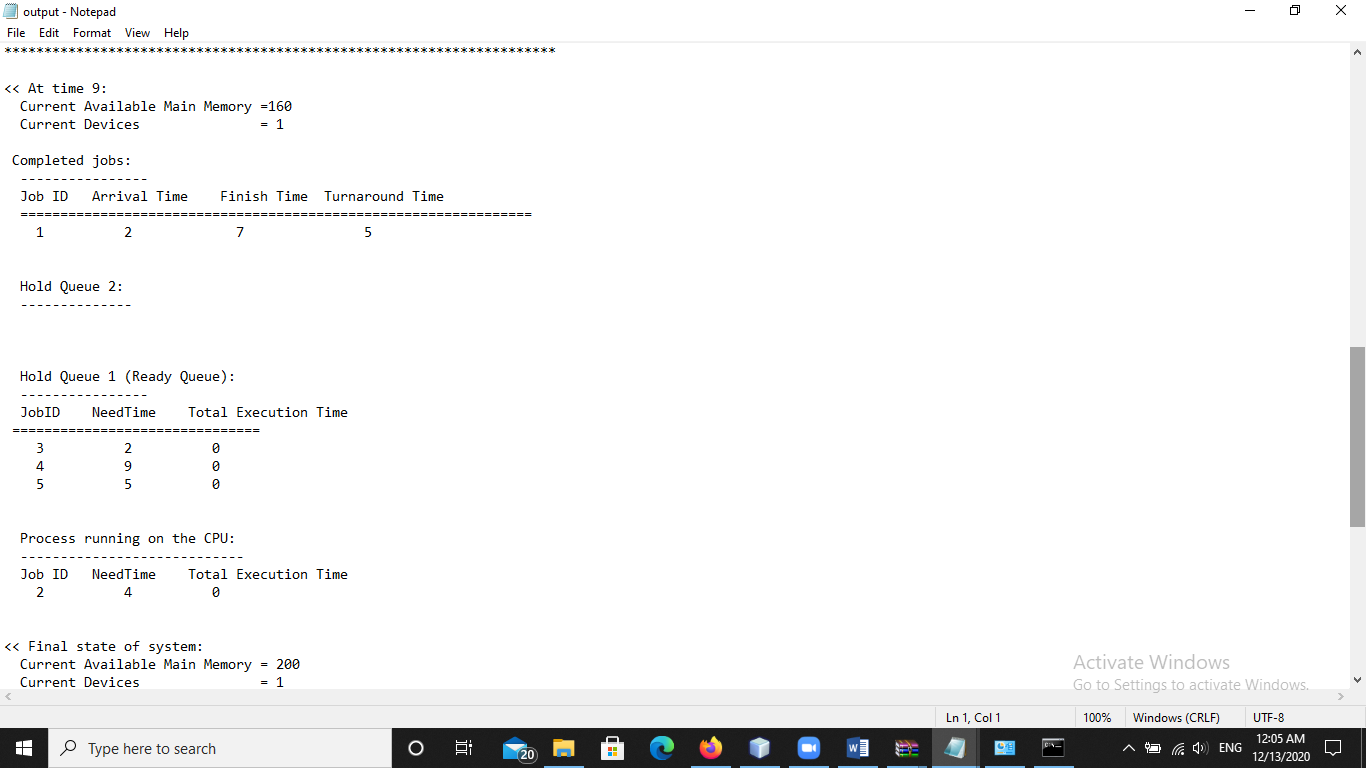
****

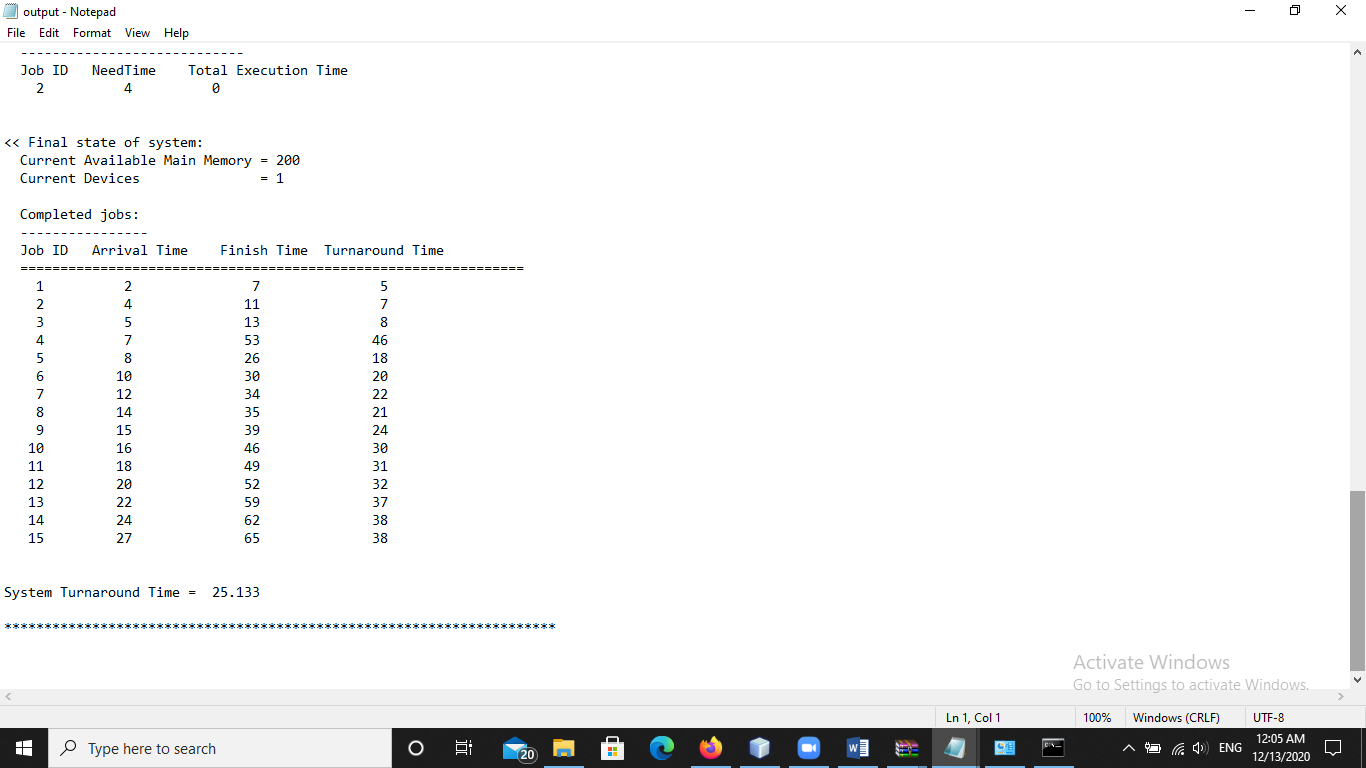
****

* **Input 2**

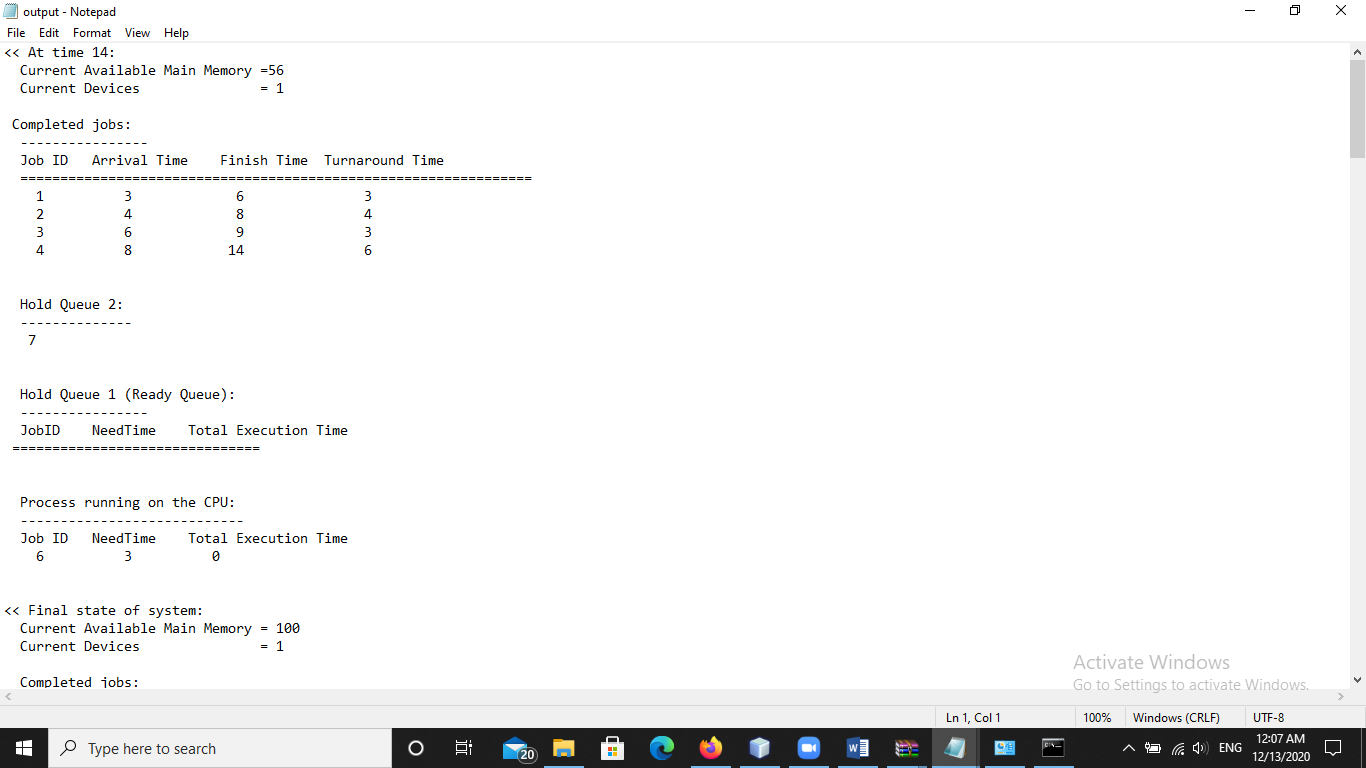
****

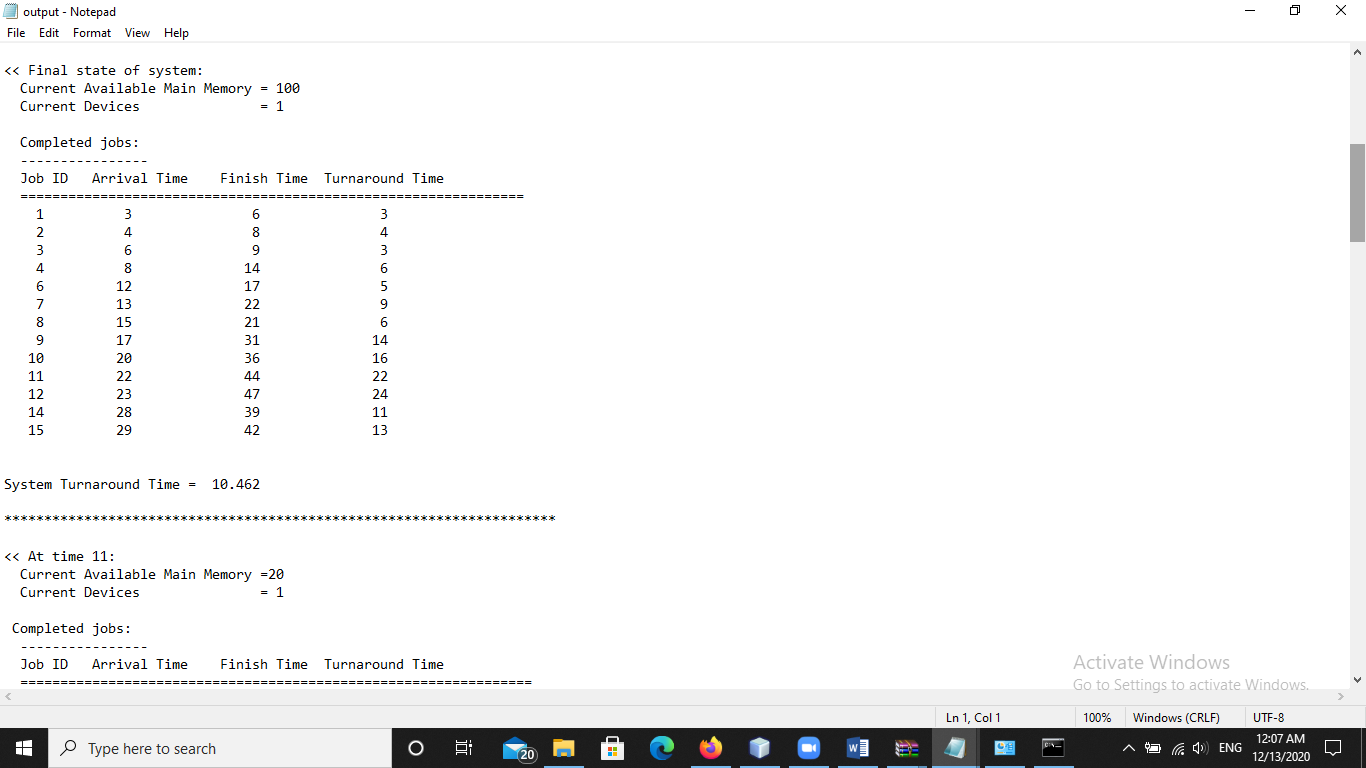
****

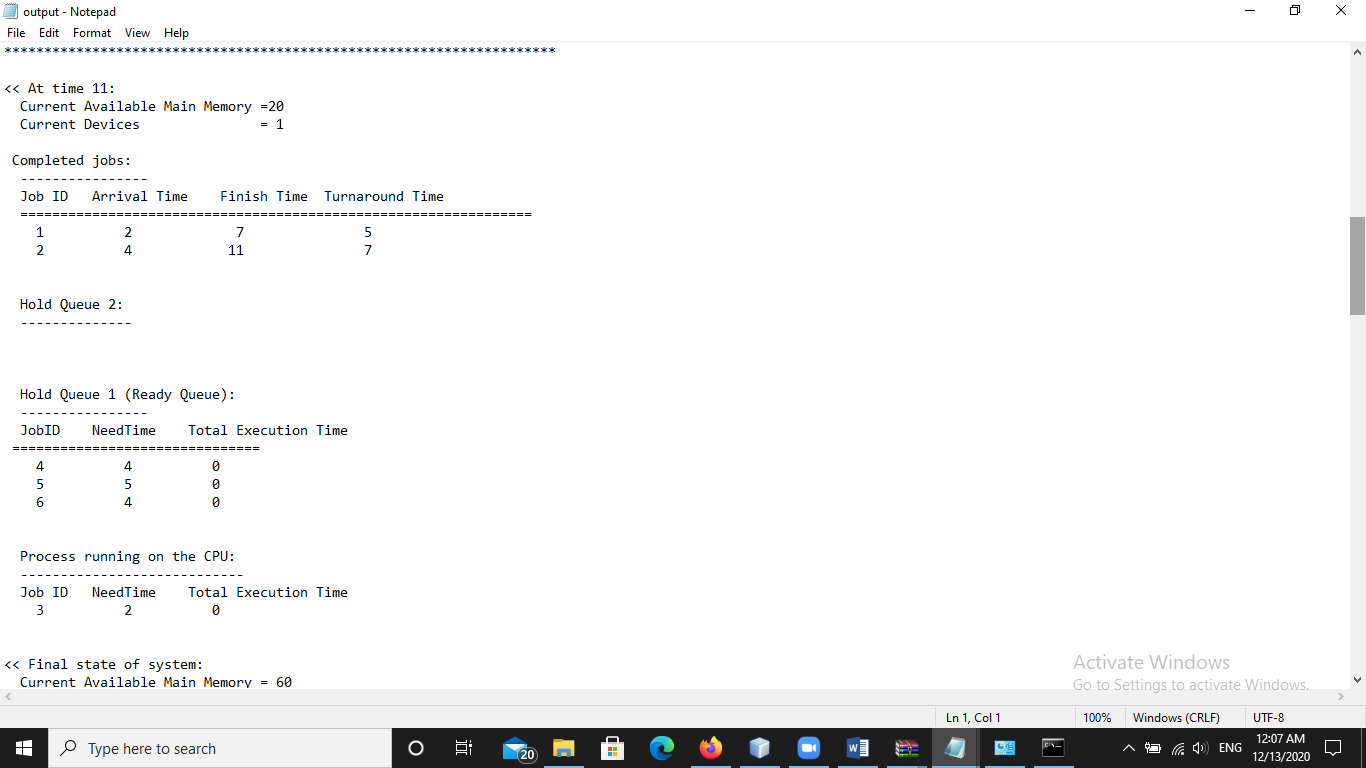
****

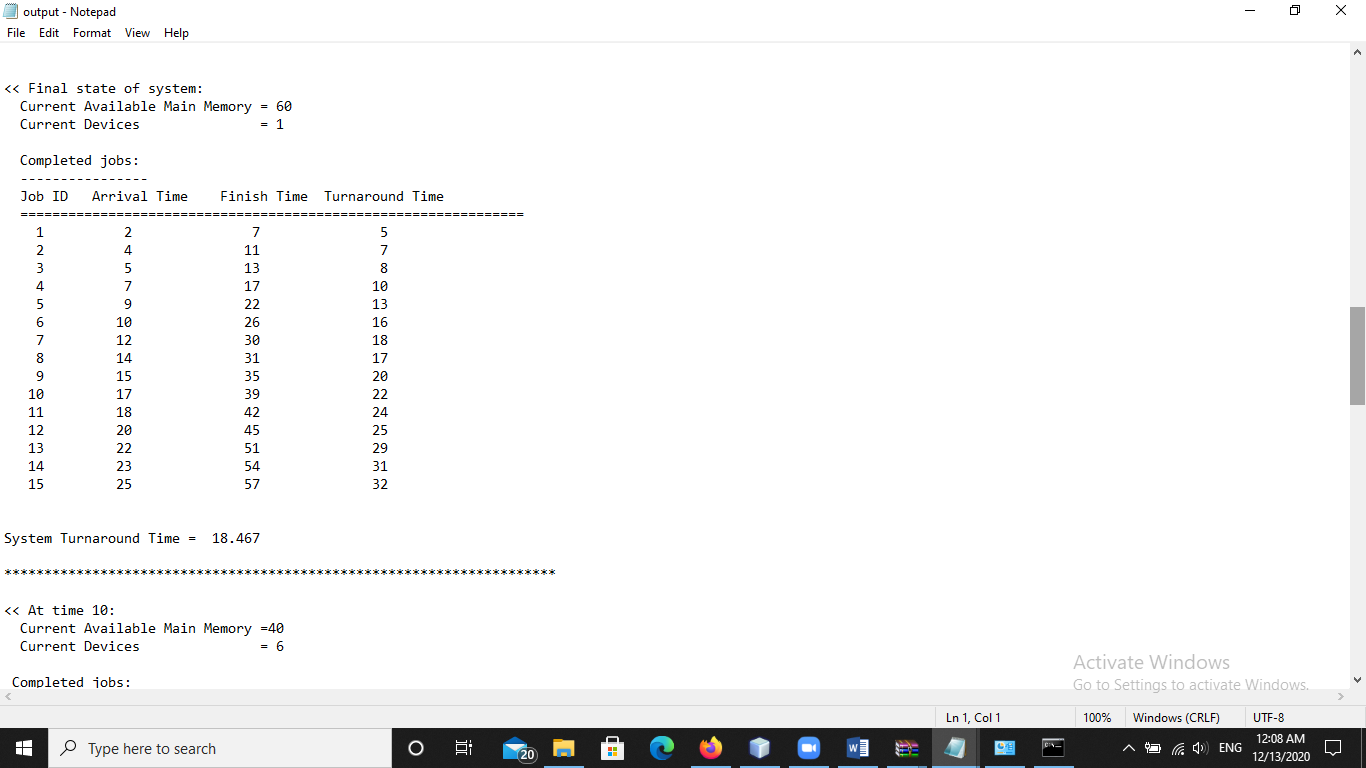
****

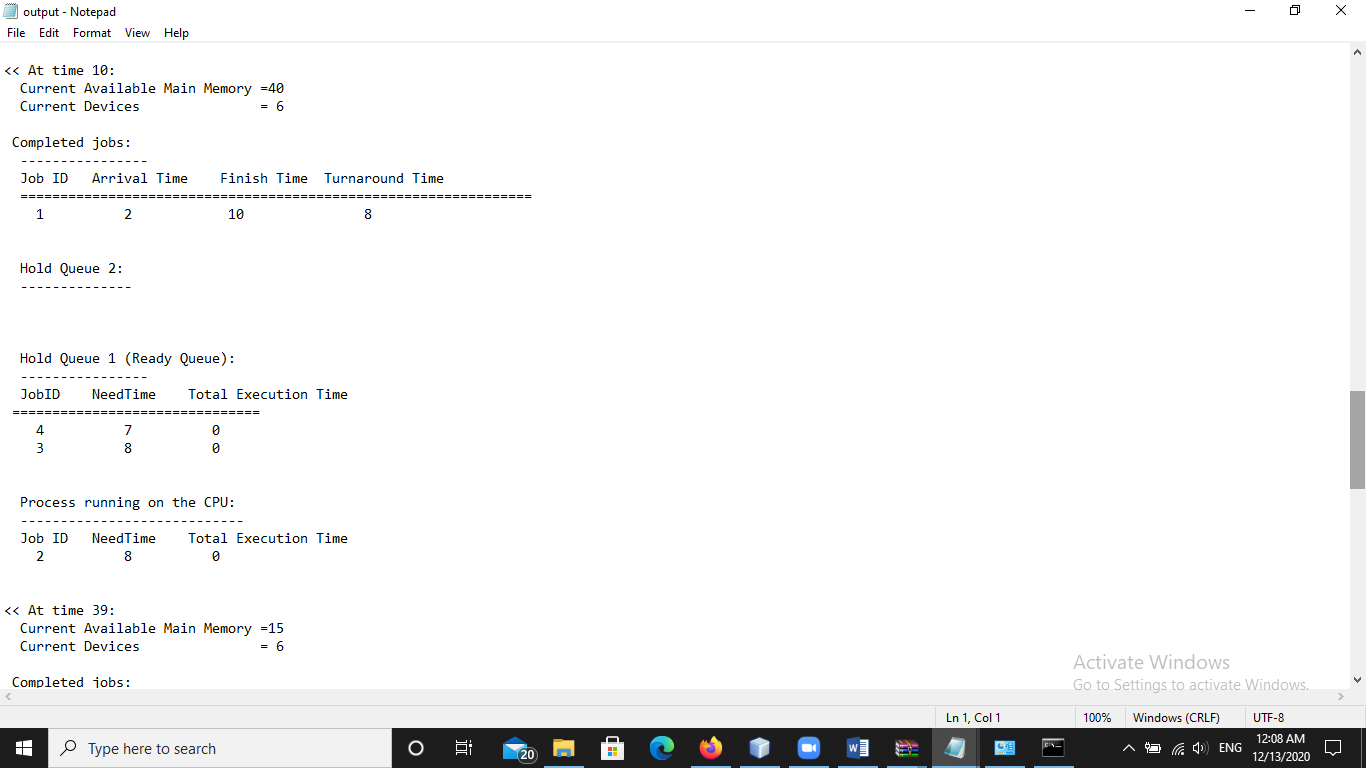
* **Input 3**

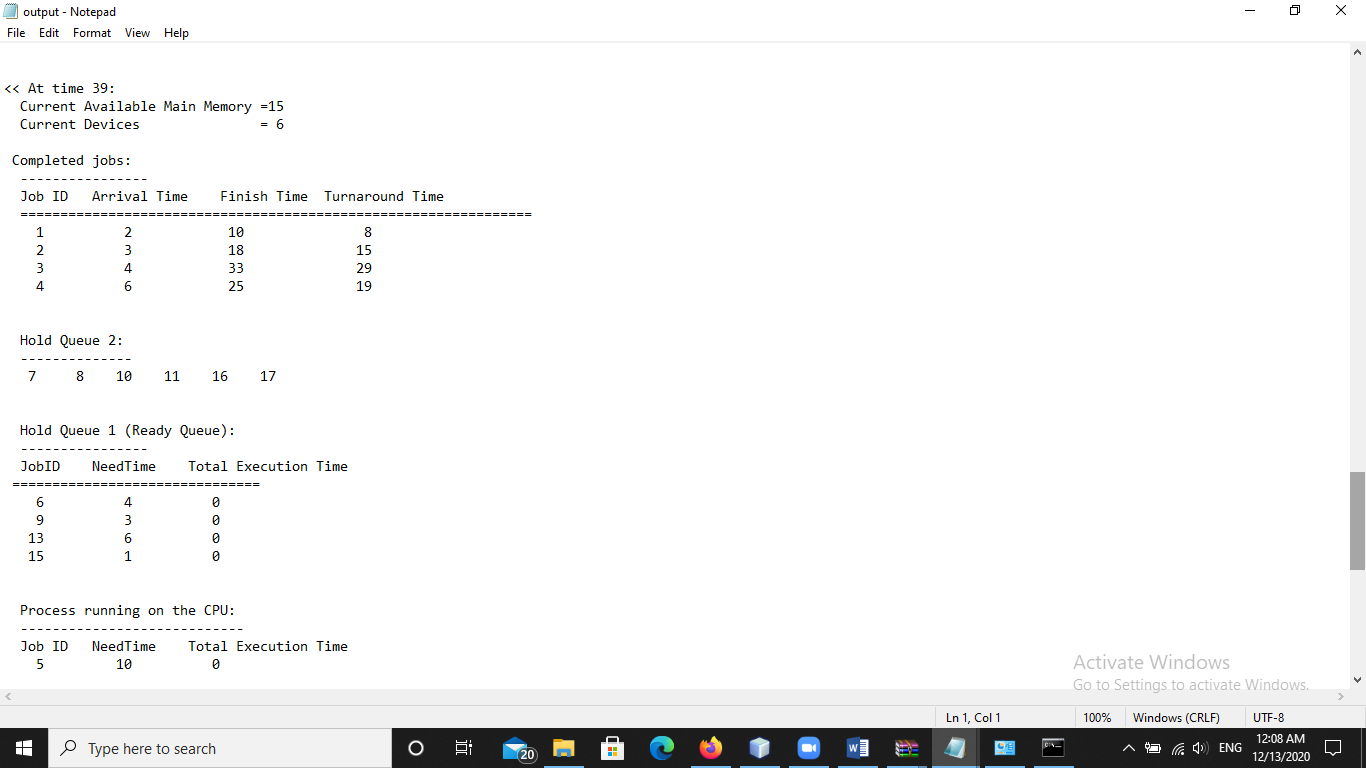
****

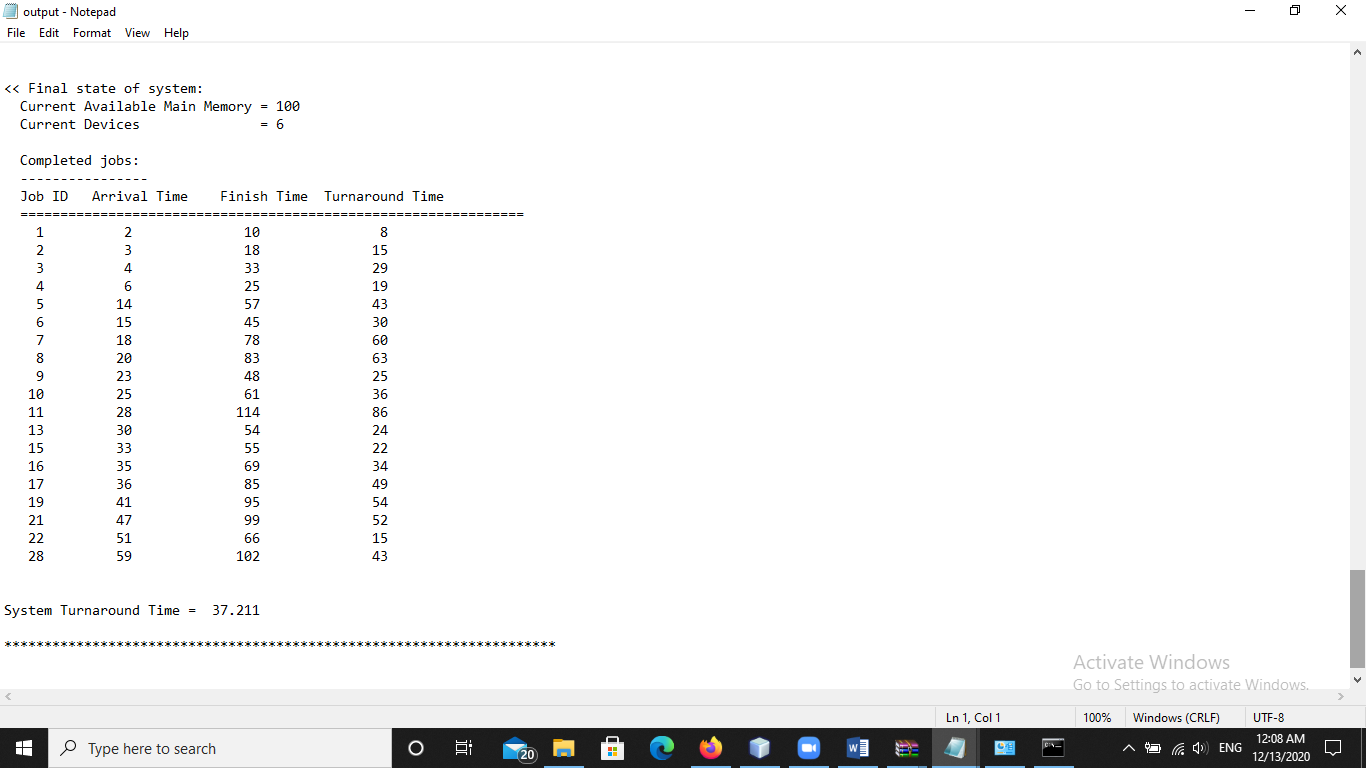
****

****

****

****

****

****

1. **The Comparative study of PART II**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input 1** | | | | |
| **Current Available Main Memory = 120, Current Device= 0** | | | | |
| **Dynamic RR** | | **Static RR** | |  |
| Turnaround Time | Waiting time | Turnaround Time | Waiting time | Job ID |
| 10 | 0 | 32 | 22 | 1 |
| 49 | 34 | 38 | 23 | 2 |
| 28 | 22 | 20 | 14 | 3 |
| 57 | 41 | 51 | 35 | 4 |
| 85 | 75 | 95 | 85 | 5 |
| 65 | 53 | 91 | 79 | 6 |
| 40 | 35 | 51 | 46 | 7 |
| 105 | 90 | 105 | 90 | 8 |
| it has no available memory. | | | | 9 |
| 82 | 72 | 68 | 58 | 10 |
| 34 | 25 | 44 | 35 | 11 |
| 32 | 29 | 28 | 25 | 12 |
| 24 | 19 | 20 | 15 | 13 |

* **Here the memory stops at J=9 because its memory exceeds 120, it equal 130, so here the LOOP stops it.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input 2** | | | |  |
| **Current Available Main Memory= 56, Current Device= 1** | | | |  |
| **Dynamic RR** | | **Static RR** | |  |
| Turnaround Time | Waiting time | Turnaround Time | Waiting time | Job ID |
| 3 | 0 | 3 | 0 | 1 |
| 4 | 2 | 4 | 2 | 2 |
| 3 | 2 | 3 | 2 | 3 |
| 6 | 1 | 6 | 1 | 4 |
| it has no available memory. | | | | 5 |
| The memory required in the J=5 equals 241, so this is not available for the existing memory. | | | | 6 |
| 7 |
| Everything that follows after the J=5 does not enter memory and is canceled. | | | | 8 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input 3** | | | | |
| **Current Available Main Memory= 56, Current Device= 1** | | | | |
| **Dynamic RR** | | **Static RR** | |  |
| Turnaround Time | Waiting time | Turnaround Time | Waiting time | Job ID |
| 3 | 0 | 3 | 0 | 1 |
| 4 | 2 | 4 | 2 | 2 |
| 3 | 2 | 3 | 2 | 3 |
| 6 | 1 | 6 | 1 | 4 |
| it has no available memory. | | | | 5 |
| The memory required in the J=5 equals 241, so this is not available for the existing memory. | | | | 6 |
| Everything that follows after the J=5 does not enter memory and is canceled. | | | | 7 |

* **When choosing a main memory of 56, both input 2&3 have the same turnaround time and waiting time in the Dynamic and Static RR.**
* **And for each of them, memory is not allowed at the same memory capacity in J=5, so both are equal to 241, so the memory stops then and the LOOp stops.**

|  |  |  |
| --- | --- | --- |
|  | **Waiting time** | **Turnaround time** |
| **Static RR** | There is no Static time waiting only there is a priority for time. | It takes more time. |
| **Dynamic RR** | There is time to wait, whoever waits more takes it out. | It take less time. |

* **The time improved in terms of Dynamic RR more than Static RR, because the Dynamic RR takes out those who wait more in time and takes it out first, while the static doesn’t wait , but rather uses the priority of time.**

1. **Reference**

**https://www.scientific.net/AMM.347-350.2203**