1. **The program with documentation:**

* **The PCB class:**

enum STATE { New, Ready, Running, Waiting, Terminated }

public class PCB {

public int pid;

public STATE state = STATE.New;

public int burstTime;

public int RemBurstTime;

public int memorySize;

public int waitingTime;

public int turnAroundTime;

public int completionTime;

public PCB(int id, int burstTime, int memorySize){

this.pid = id;

this.burstTime = burstTime;

this.memorySize = memorySize;

this.RemBurstTime = burstTime;

}

}

Description: A class for the Process Control Block (PCB) created with the following information:

• Process ID: Contains the process ID.

• Process state as an enum value containing (New, Ready, Running, Waiting, Terminated).

• Burst Time.

• Memory Required.

Along with the remaining burst time, waiting time, turnaround time, and completion time for each process.

* **The readFileToJobQueue class:**

public class readFileToJobQueue extends Thread {

static PCB[] arrOfProcesses = new PCB[30];

static Queue<PCB> jobQueue = new LinkedList<>();

static int n = 0;

static int numberOffProcesses = 0;

public void run() {

try {

File file = new File("C:\\Users\\Huawei\\Desktop\\testdata1.txt");

Scanner scan = new Scanner(file);

while (scan.hasNextLine()) {

scan.nextLine();

String[] numbers = scan.nextLine().trim().split(", ");

int processID = Integer.parseInt(numbers[0]);

int burstTime = Integer.parseInt(numbers[1]);

int memorySize = Integer.parseInt(numbers[2]);

PCB p = new PCB(processID, burstTime, memorySize);

arrOfProcesses[n++] = p;

}

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

jobQueue.add(arrOfProcesses[i]);

numberOffProcesses++;

}

} catch (FileNotFoundException e) {

System.out.println("invalid file path");

} catch (NumberFormatException e) {

System.out.println("invalid file format");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("invalid process must be less than or equal 30");

}

}

}

Description: A class that serves as a thread for reading the text file and input the values of each process into a PCB; then adding all PCBs to a Job Queue.

* **The jobToReadyQueue class:**

public class jobToReadyQueue extends Thread {

public static Queue<PCB> readyQueue = new LinkedList<>();

public void run() {

try {

while (!readFileToJobQueue.jobQueue.isEmpty()) {

int memorySizeOfJob = readFileToJobQueue.jobQueue.peek().memorySize;

if (Main.memory - memorySizeOfJob >= 0) {

Main.memory -= memorySizeOfJob;

readFileToJobQueue.jobQueue.peek().state = STATE.Ready;

readyQueue.add(readFileToJobQueue.jobQueue.remove());

}

if (memorySizeOfJob > 8192) {

readFileToJobQueue.jobQueue.remove();

}

Thread.sleep(10);

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

Description: A class that serves a thread for adding all PCBs of Job Queue to Ready Queue with the memory restriction of 8192 MB for the whole Ready Queue main memory (fill until Job Queue is empty or until there is not enough main memory; if a PCB's memory is above 8192, remove).

* **The Scheduler class:**

public class Scheduler {

public static LinkedList<PCB> terminatedProcesses = new LinkedList<>();

static int start = 0;

public static void Ganttchart(PCB p, int st, int ft) {

System.out.println("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|");

System.out.println("| ( p" + p.pid + " ) |");

System.out.println(" "+st + " =====> " + ft);

}

public static void Table(Queue<PCB> queue) {

System.out.println(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println(

"| pID | Burst Time | Turnaround Time | Waiting Time | Memory | Status |");

int totalwaitingTime = 0;

int totalTurnaroundTime = 0;

int totalcompletionTime = 0;

int numOfProcess = 0;

int size = terminatedProcesses.size();

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

numOfProcess += 1;

totalwaitingTime += terminatedProcesses.peek().waitingTime;

totalTurnaroundTime += terminatedProcesses.peek().turnAroundTime;

totalcompletionTime += terminatedProcesses.peek().completionTime;

System.out.println(" "+terminatedProcesses.peek().pid + " "

+ terminatedProcesses.peek().burstTime

+ " " + terminatedProcesses.peek().turnAroundTime + " "

+ terminatedProcesses.peek().waitingTime + " "

+ terminatedProcesses.peek().memorySize

+ " " + terminatedProcesses.peek().state.toString());

terminatedProcesses.remove();

}

System.out.println("|\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

System.out.println("Average Waiting Time = " + ((double) totalwaitingTime / numOfProcess)+" msec");

System.out.println("Average Turnaround Time = " + ((double) totalTurnaroundTime / numOfProcess)+" msec");

}

public static void FCFS(Queue<PCB> Q) {

while (!Q.isEmpty()) {

Ganttchart(Q.peek(), start, start + Q.peek().RemBurstTime);

Q.peek().state = STATE.Running;

Q.peek().completionTime = start + Q.peek().RemBurstTime;

Q.peek().turnAroundTime = Q.peek().completionTime;

Q.peek().waitingTime = Q.peek().turnAroundTime - Q.peek().burstTime;

start += Q.peek().burstTime;

Q.peek().RemBurstTime = 0;

System.out.println(" p" + Q.peek().pid + " Memory usage: " + Q.peek().memorySize);

Q.peek().state = STATE.Terminated;

Main.memory += Q.peek().memorySize;

terminatedProcesses.add(Q.remove());

try {

Thread.sleep(20);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("");

Table(terminatedProcesses);

}

public static void roundrobin(Queue<PCB> Q, int q) {

while (!Q.isEmpty()) {

PCB job = Q.remove();

job.state = STATE.Running;

if (job.RemBurstTime > q) {

job.RemBurstTime -= q;

Ganttchart(job, start, start + q);

System.out.println(" p" + job.pid + " Memory usage: " + job.memorySize);

start += q;

job.state = STATE.Ready;

Q.add(job);

} else {

Ganttchart(job, start, start + job.RemBurstTime);

System.out.println(" p" + job.pid + " Memory usage: " + job.memorySize);

job.completionTime = start + job.RemBurstTime;

start += job.RemBurstTime;

job.turnAroundTime = job.completionTime;

job.waitingTime = job.turnAroundTime - job.burstTime;

job.RemBurstTime -= job.RemBurstTime;

job.state = STATE.Terminated;

Main.memory += job.memorySize;

terminatedProcesses.add(job);

}

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

Table(terminatedProcesses);

}

public static void SJF(Queue<PCB> Q) {

List<PCB> jobList = new ArrayList<>(Q);

jobList.sort(Comparator.comparingInt(p -> p.burstTime));

for (PCB job : jobList) {

Ganttchart(job, start, start + job.burstTime);

job.state = STATE.Running;

job.completionTime = start + job.burstTime;

job.turnAroundTime = job.completionTime;

job.waitingTime = job.turnAroundTime - job.burstTime;

start += job.burstTime;

job.RemBurstTime = 0;

System.out.println(" p" + job.pid + " Memory usage: " + job.memorySize);

job.state = STATE.Terminated;

Main.memory += job.memorySize;

terminatedProcesses.add(job);

try {

Thread.sleep(20);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.out.println("");

Table(terminatedProcesses);

}

}

Description: A class that simulates FCFS, SJF, RR-3, and RR-5 for all processes, and inputs them in a Linked-List called TerminationProcesses after termination; then after choosing the scheduling algorithm, the class displays the whole procedure in terms of Gantt Chart.

* **The Main class:**

public class Main {

public static int memory = 8192;

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

readFileToJobQueue firstThread = new readFileToJobQueue();

jobToReadyQueue secondThread = new jobToReadyQueue();

firstThread.start();

firstThread.join();

secondThread.start();

Scanner scan = new Scanner(System.in);

System.out.println("\n=============================================");

System.out.println("Choose which scheduling algorithm (Exit -1): \n\n1.First-Come-First-Serve (FCFS) \n2.Shortest-Job-First (SJF) \n3.Round-Robin with time slice = 3 (RR-3)"

+ " \n4.Round-Robin with time slice = 5 (RR-5)");

System.out.println("\nEnter number from 1 to 4");

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

String input = scan.next();

boolean flag = true;

do {

switch (input) {

case "1":

Scheduler.FCFS(jobToReadyQueue.readyQueue);

flag = false;

break;

case "2":

Scheduler.SJF(jobToReadyQueue.readyQueue);

flag = false;

break;

case "3":

Scheduler.roundrobin(jobToReadyQueue.readyQueue, 3);

flag = false;

break;

case "4":

Scheduler.roundrobin(jobToReadyQueue.readyQueue, 5);

flag = false;

break;

case "-1":

flag = false;

System.out.println("DONE");

break;

default:

System.out.println("Enter a number from 1 to 3");

input = scan.next();

}

} while (flag);

scan.close();

}

}

Description: the main class that starts with readFileToJobQueue thread, then jobToReadyQueue thread. Then it displays the algorithms (from 1 to 4) telling the user to choose (if not, then input -1).