**CPU Scheduling Techniques**

1. **FCFS (First Come First Serve)**

import java.util.Scanner;

class Fcfs {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of processes: ");

int n = sc.nextInt();

int[] at = new int[n];

int[] bt = new int[n];

int[] ct = new int[n];

int[] tat = new int[n];

int[] wt = new int[n];

int[] pid = new int[n];

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

pid[i] = i + 1;

System.out.println("\nFor Process " + pid[i] + ":");

System.out.print("Enter Arrival Time: ");

at[i] = sc.nextInt();

System.out.print("Enter Burst Time: ");

bt[i] = sc.nextInt();

}

int currTime = 0;

int totalTAT = 0, totalWT = 0;

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

if (currTime < at[i]) {

currTime = at[i];

}

ct[i] = currTime + bt[i];

tat[i] = ct[i] - at[i];

wt[i] = tat[i] - bt[i];

currTime = ct[i];

totalTAT += tat[i];

totalWT += wt[i];

}

System.out.println("\nProcess\tAT\tBT\tCT\tTAT\tWT");

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

System.out.println("P" + pid[i] + "\t" + at[i] + "\t" + bt[i] + "\t" +

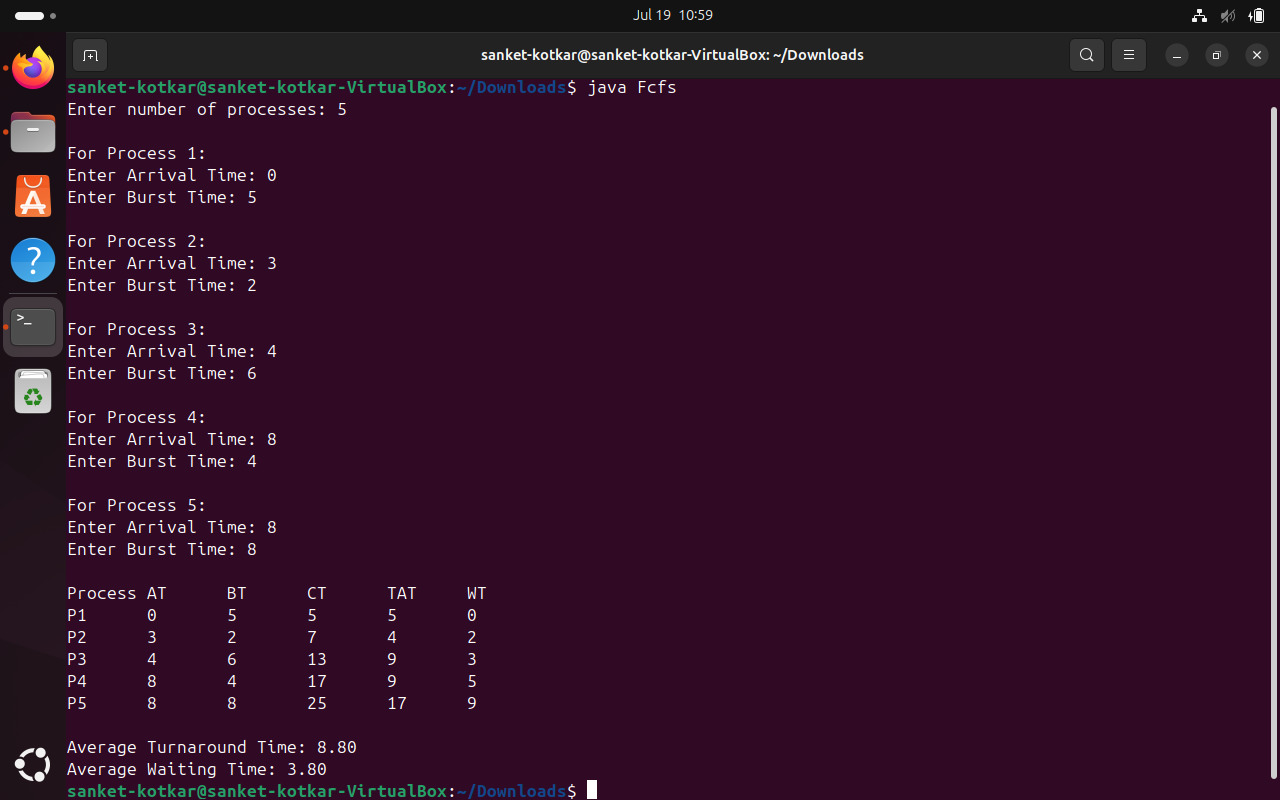
ct[i] + "\t" + tat[i] + "\t" + wt[i]);

}

System.out.printf("\nAverage Turnaround Time: %.2f\n", totalTAT / (double)n);

System.out.printf("Average Waiting Time: %.2f\n", totalWT / (double)n);

}

}

1. **SJF (Shortest Job First)**

import java.util.Scanner;

class Sjf {

static class Process {

int id, at, bt, ct, tat, wt;

boolean completed;

Process(int id, int at, int bt) {

this.id = id;

this.at = at;

this.bt = bt;

this.completed = false;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the Number of Processes:");

int n = sc.nextInt();

Process[] processes = new Process[n];

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

System.out.println("For Job " + (i + 1) + ": ");

System.out.print("Enter Arrival Time: ");

int at = sc.nextInt();

System.out.print("Enter Burst Time: ");

int bt = sc.nextInt();

processes[i] = new Process(i + 1, at, bt);

}

int currTime = 0, completedCount = 0;

int avg\_tt = 0, avg\_wt = 0;

while (completedCount < n) {

int idx = -1;

int min\_bt = Integer.MAX\_VALUE;

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

if (!processes[i].completed && processes[i].at <= currTime) {

if (processes[i].bt < min\_bt) {

min\_bt = processes[i].bt;

idx = i;

}

}

}

if (idx == -1) {

currTime++;

continue;

}

Process p = processes[idx];

currTime += p.bt;

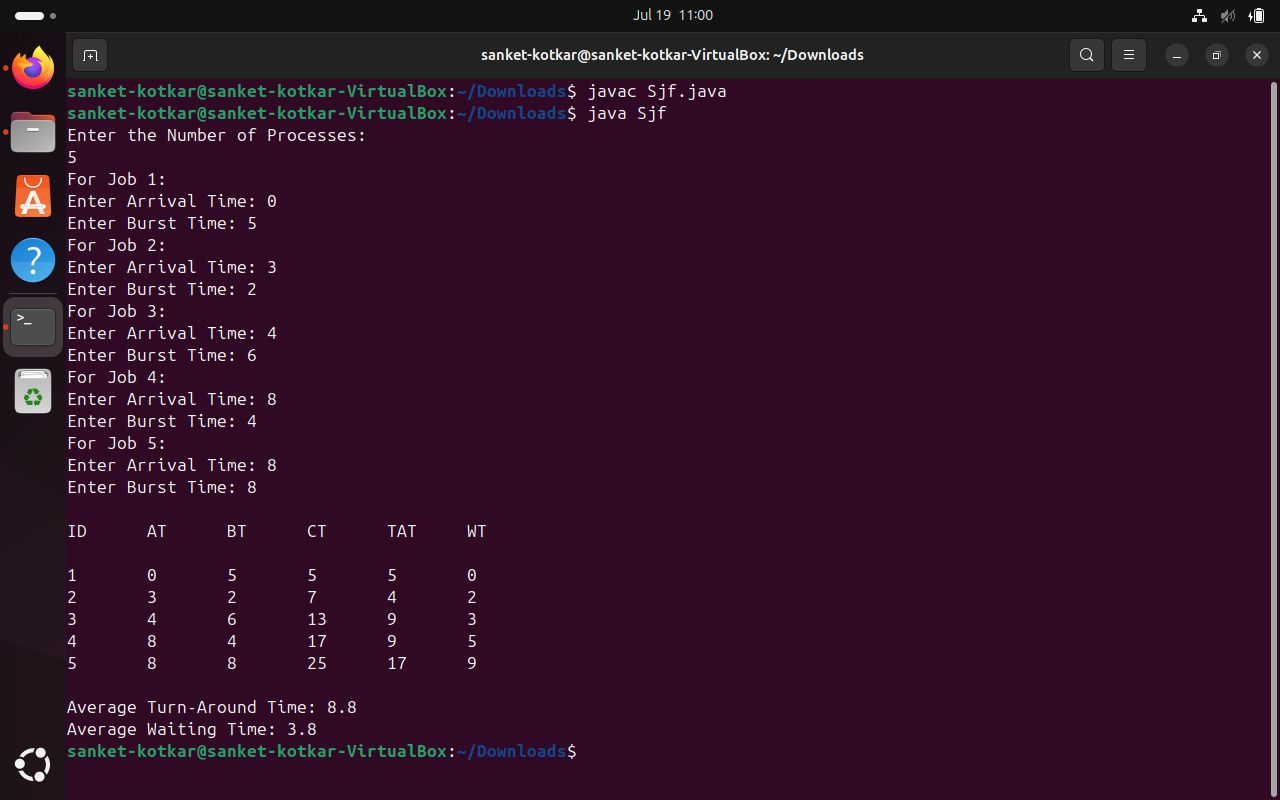
p.ct = currTime;

p.tat = p.ct - p.at;

p.wt = p.tat - p.bt;

p.completed = true;

completedCount++;

 avg\_tt += p.tat;

avg\_wt += p.wt;

}

System.out.println("\nID\tAT\tBT\tCT\tTAT\tWT\n");

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

System.out.println(

processes[i].id + "\t" +

processes[i].at + "\t" +

processes[i].bt + "\t" +

processes[i].ct + "\t" +

processes[i].tat + "\t" +

processes[i].wt

);

}

System.out.println("\nAverage Turn-Around Time: " + (avg\_tt / (float) n));

System.out.println("Average Waiting Time: " + (avg\_wt / (float) n));

}

}

1. **Priority Scheduling (Non-Preemptive)**

import java.util.Scanner;

class Priority {

static class Process {

int id, at, bt, ct, tat, wt, priority;

boolean completed;

Process(int id, int at, int bt, int priority) {

this.id = id;

this.at = at;

this.bt = bt;

this.priority = priority;

this.completed = false;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the Number of Processes:");

int n = sc.nextInt();

Process[] processes = new Process[n];

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

System.out.println("For Process " + (i + 1) + ":");

System.out.print("Enter Arrival Time: ");

int at = sc.nextInt();

System.out.print("Enter Burst Time: ");

int bt = sc.nextInt();

System.out.print("Enter Priority (lower number = higher priority): ");

int priority = sc.nextInt();

processes[i] = new Process(i + 1, at, bt, priority);

}

int currTime = 0, completedCount = 0;

int avg\_tat = 0, avg\_wt = 0;

while (completedCount < n) {

int idx = -1;

int highestPriority = Integer.MAX\_VALUE;

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

if (!processes[i].completed && processes[i].at <= currTime) {

if (processes[i].priority < highestPriority) {

highestPriority = processes[i].priority;

idx = i;

} else if (processes[i].priority == highestPriority) {

// Tie-breaking: shorter burst time

if (idx == -1 || processes[i].bt < processes[idx].bt) {

idx = i;

}

}

}

}

if (idx == -1) {

currTime++;

continue;

}

Process p = processes[idx];

currTime += p.bt;

p.ct = currTime;

p.tat = p.ct - p.at;

p.wt = p.tat - p.bt;

p.completed = true;

completedCount++;

avg\_tat += p.tat;

avg\_wt += p.wt;

}

System.out.println("\nID\tAT\tBT\tPriority\tCT\tTAT\tWT\n");

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

Process p = processes[i];

System.out.println(

p.id + "\t" +

p.at + "\t" +

p.bt + "\t" +

p.priority + "\t\t" +

p.ct + "\t" +

p.tat + "\t" +

p.wt

);

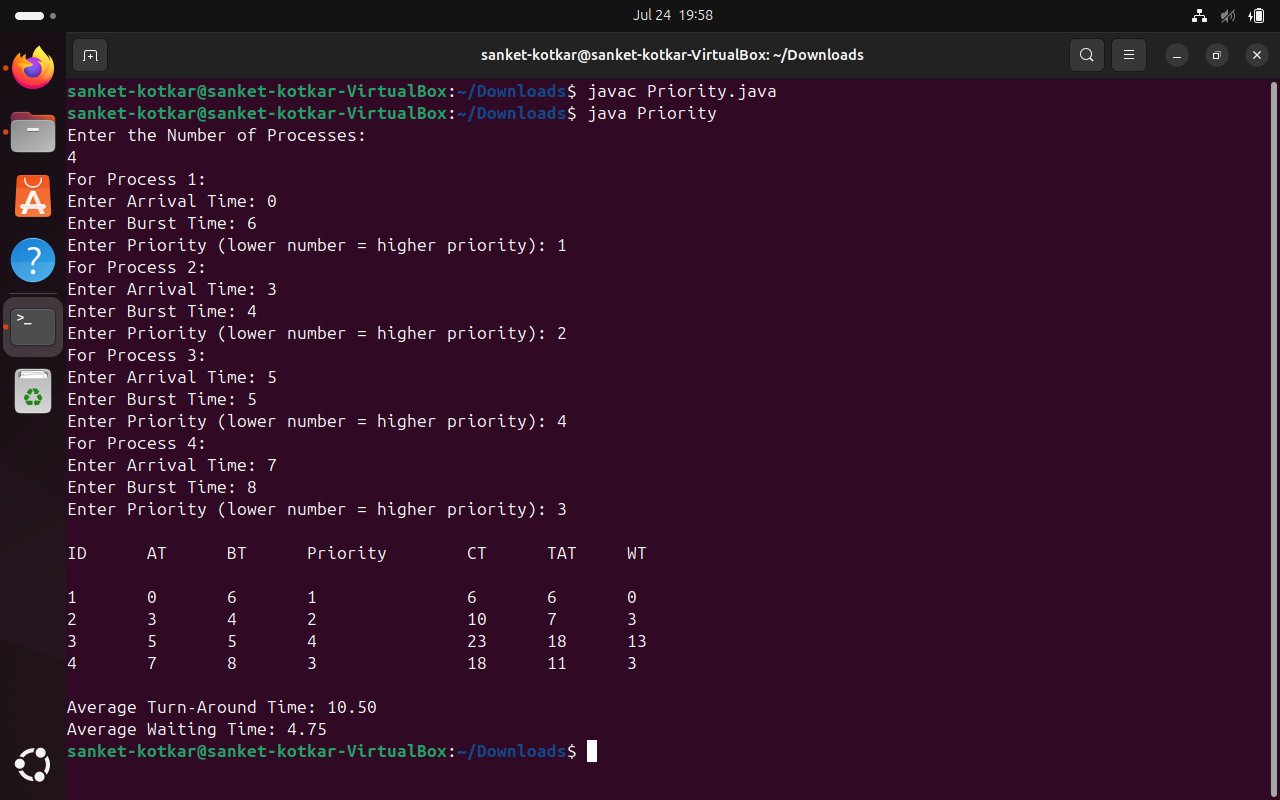
}

System.out.printf("\nAverage Turn-Around Time: %.2f\n", (avg\_tat / (float) n));

System.out.printf("Average Waiting Time: %.2f\n", (avg\_wt / (float) n));

}

}



1. **Round Robin (RR)**

import java.util.\*;

class RR {

static class Process {

int id, at, bt, remainingBt, ct, tat, wt;

boolean completed;

Process(int id, int at, int bt) {

this.id = id;

this.at = at;

this.bt = bt;

this.remainingBt = bt;

this.completed = false;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of processes: ");

int n = sc.nextInt();

Process[] processes = new Process[n];

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

System.out.println("For Process " + (i + 1) + ":");

System.out.print("Enter Arrival Time: ");

int at = sc.nextInt();

System.out.print("Enter Burst Time: ");

int bt = sc.nextInt();

processes[i] = new Process(i + 1, at, bt);

}

System.out.print("Enter Time Quantum: ");

int tq = sc.nextInt();

Queue<Process> queue = new LinkedList<>();

int currTime = 0, completedCount = 0;

int avgTat = 0, avgWt = 0;

Arrays.sort(processes, Comparator.comparingInt(p -> p.at));

int index = 0;

while (index < n && processes[index].at <= currTime) {

queue.add(processes[index]);

index++;

}

while (!queue.isEmpty()) {

Process p = queue.poll();

if (p.remainingBt > tq) {

currTime += tq;

p.remainingBt -= tq;

} else {

currTime += p.remainingBt;

p.remainingBt = 0;

p.ct = currTime;

p.tat = p.ct - p.at;

p.wt = p.tat - p.bt;

p.completed = true;

completedCount++;

avgTat += p.tat;

avgWt += p.wt;

}

while (index < n && processes[index].at <= currTime) {

queue.add(processes[index]);

index++;

}

if (!p.completed) {

queue.add(p);

}

if (queue.isEmpty() && index < n) {

currTime = processes[index].at;

queue.add(processes[index]);

index++;

}

}

System.out.println("\nID\tAT\tBT\tCT\tTAT\tWT\n");

for (Process p : processes) {

System.out.println(p.id + "\t" + p.at + "\t" + p.bt + "\t" + p.ct + "\t" + p.tat + "\t" + p.wt);

}

System.out.printf("\nAverage Turn-Around Time: %.2f\n", (avgTat / (float) n));

System.out.printf("Average Waiting Time: %.2f\n", (avgWt / (float) n));

}

}

