**PROGRAM CODE:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Scanner;

class Process implements Comparable<Process> {

int index;

int arrTime;

int burstTime;

Process(int i, int a, int b) {

this.arrTime = a;

this.burstTime = b;

this.index = i;

}

public int compareTo(Process o) {

return this.arrTime - o.arrTime;

}

}

public class fcfsfinal {

public static void avgWaitTime(ArrayList<Process> process) {

int wt[] = new int[process.size()];

int tat[] = new int[process.size()];

int ct[] = new int[process.size()];

int st[] = new int[process.size()];

startingTime(process, st);

compleitionTime(process, ct, st);

tat(process, ct, tat);

waitingTime(process, tat, wt);

System.out.println("\nProcesses ||" + " Burst Time ||" + " Arrival Time || Starting Time ||" + " Waiting Time || Compleition Time ||" + " Turn-Around Time ");

int totalWaitingTime = 0;

int totalTurnaroundTime = 0;

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

System.out.println((process.get(i).index + 1) + "\t\t" + process.get(i).burstTime + "\t\t" + process.get(i).arrTime + "\t\t" + st[i] + "\t\t" + wt[i] + "\t\t" + ct[i] + "\t\t" + tat[i]);

totalWaitingTime += wt[i];

totalTurnaroundTime += tat[i];

}

double avgWaitingTime = (double) totalWaitingTime / process.size();

double avgTurnaroundTime = (double) totalTurnaroundTime / process.size();

double throughput = (double) process.size() / ct[process.size() - 1];

System.out.println("\nAverage Waiting Time: " + avgWaitingTime);

System.out.println("Average Turnaround Time: " + avgTurnaroundTime);

System.out.println("Throughput: " + throughput);

}

public static void startingTime(ArrayList<Process> process, int[] st) {

st[0] = process.get(0).arrTime;

for (int i = 1; i < process.size(); i++) {

st[i] = st[i - 1] + process.get(i - 1).burstTime;

}

}

public static void compleitionTime(ArrayList<Process> process, int[] ct, int[] st) {

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

ct[i] = st[i] + process.get(i).burstTime;

}

}

public static void tat(ArrayList<Process> process, int[] ct, int[] tat) {

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

tat[i] = ct[i] - process.get(i).arrTime;

}

}

public static void waitingTime(ArrayList<Process> process, int[] tat, int[] wt) {

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

wt[i] = tat[i] - process.get(i).burstTime;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

ArrayList<Process> process = new ArrayList<>();

System.out.println("Enter Arrival and Burst time");

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

System.out.print("For process " + (i + 1) + " : ");

process.add(new Process(i, sc.nextInt(), sc.nextInt()));

}

Collections.sort(process);

avgWaitTime(process);

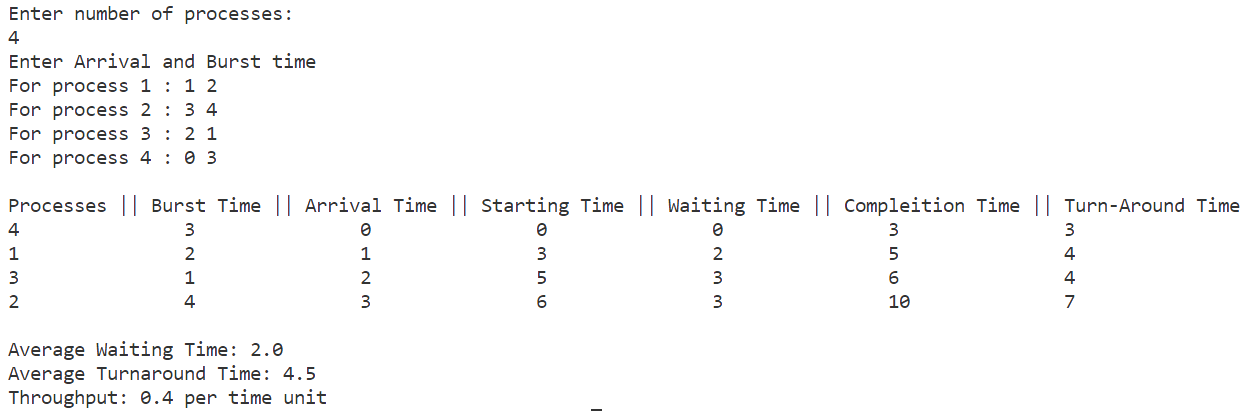
}

}

**FORMATTED INPUT:**

|  |  |  |
| --- | --- | --- |
| **PROCESS ID** | **ARRIVAL TIME** | **BURST TIME** |
| 1 | 1 | 2 |
| 2 | 3 | 4 |
| 3 | 2 | 1 |
| 4 | 0 | 3 |

**OUTPUT:**

****

**RESULT:**

The program has been executed successfully and the output is noted.

**PROGRAM CODE:**

import java.util.\*;

class Process {

    int id;

    int arrivalTime;

    int burstTime;

    int startTime;

    int completionTime;

    int waitingTime;

    int turnAroundTime;

}

public class sjfsfinal {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

        // Create an array of processes

        Process[] processes = new Process[n];

        System.out.println("Enter Arrival and Burst time :");

        // Get arrival time and burst time for each process

        for (int i = 0; i < n; i++) {

            processes[i] = new Process();

            processes[i].id = i + 1;

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

            processes[i].arrivalTime = scanner.nextInt();

            processes[i].burstTime = scanner.nextInt();

        }

        // Sort the processes based on arrival time and burst time

        Arrays.sort(processes, Comparator.comparingInt((Process p) -> p.arrivalTime)

                .thenComparingInt(p -> p.burstTime));

        int currentTime = 0; // Current time

        float totalTurnAroundTime = 0;

        float totalWaitingTime = 0;

        System.out.println("\nProcess\tArrival Time\tBurst Time\tStart Time\tCompletion Time\tWaiting Time\tTurnaround Time");

        // Calculate scheduling metrics for each process

        for (int i = 0; i < n; i++) {

            Process currentProcess = processes[i];

            // Calculate waiting time

            currentProcess.waitingTime = currentTime - currentProcess.arrivalTime;

            if (currentProcess.waitingTime < 0) {

                currentProcess.waitingTime = 0;

                currentTime = currentProcess.arrivalTime;

            }

            // Calculate starting time and completion time

            currentProcess.startTime = currentTime;

            currentProcess.completionTime = currentTime + currentProcess.burstTime;

            // Calculate turnaround time

            currentProcess.turnAroundTime = currentProcess.completionTime - currentProcess.arrivalTime;

            // Update current time

            currentTime = currentProcess.completionTime;

            // Calculate total turnaround time and total completion time

            totalTurnAroundTime += currentProcess.turnAroundTime;

            totalWaitingTime += currentProcess.waitingTime;

            // Print process details

            System.out.printf("%-8d%-16d%-16d%-16d%-20d%-16d%-16d\n", currentProcess.id, currentProcess.arrivalTime,

                    currentProcess.burstTime, currentProcess.startTime, currentProcess.completionTime,

                    currentProcess.waitingTime, currentProcess.turnAroundTime);

        }

        // Calculate average turnaround time, average waiting time, and throughput

        float avgTurnAroundTime = totalTurnAroundTime / processes.length;

        float avgWaitingTime = totalWaitingTime / processes.length;

        float throughput = (float) processes.length / currentTime;

        System.out.println("\nAverage Turnaround Time: " + avgTurnAroundTime);

        System.out.println("Average Waiting Time: " + avgWaitingTime);

        System.out.println("Throughput: " + throughput +" per time unit");

        scanner.close();

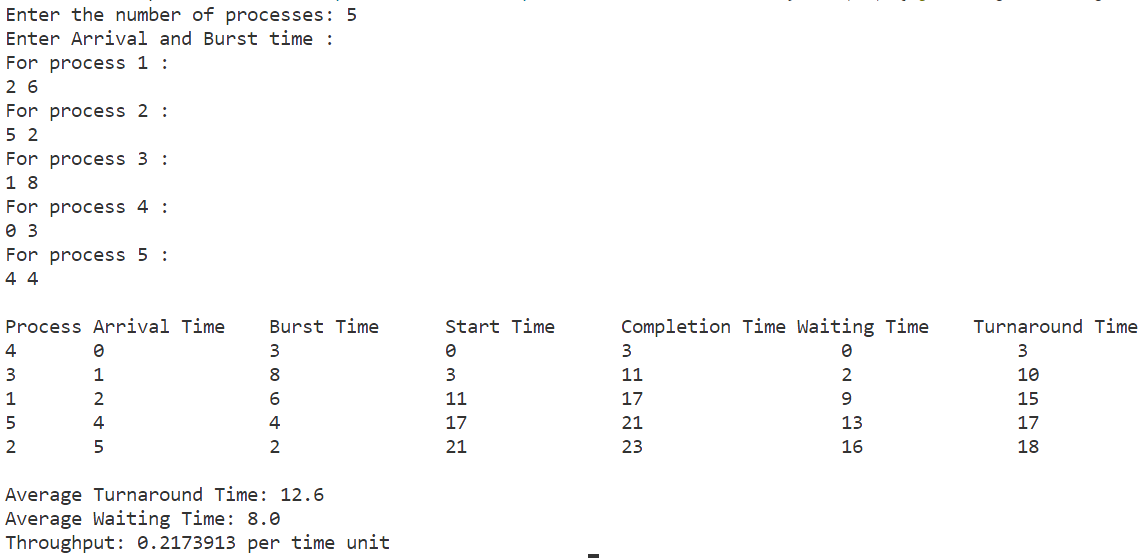
    }

}

**FORMATTED INPUT:**

|  |  |  |
| --- | --- | --- |
| **PROCESS ID** | **ARRIVAL TIME** | **BURST TIME** |
| 1 | 2 | 6 |
| 2 | 5 | 2 |
| 3 | 1 | 8 |
| 4 | 0 | 3 |
| 5 | 4 | 4 |

**OUTPUT:**



**RESULT:**

The program has been executed successfully and the output is noted.

**PROGRAM CODE:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

import java.util.Scanner;

class Process3 {

    String name;

    int arrivalTime;

    int burstTime;

    int age = 2;

    public Process3(String name, int arrivalTime, int burstTime) {

        this.name = name;

        this.arrivalTime = arrivalTime;

        this.burstTime = burstTime;

    }

}

// Sort based on arrival time

class ArrivalComparator1 implements Comparator<Process3> {

    public int compare(Process3 p1, Process3 p2) {

        if (p1.arrivalTime < p2.arrivalTime)

            return -1;

        else if (p1.arrivalTime > p2.arrivalTime)

            return 1;

        else

            return 0;

    }

}

// Sort based on burst time

class BurstComparator1 implements Comparator<Process3> {

public int compare(Process3 p1, Process3 p2) {

        if (p1.burstTime < p2.burstTime)

            return -1;

        else if (p1.burstTime > p2.burstTime)

            return 1;

        else

            return 0;

    }

}

// Main class

public class sjfa {

    static float AverageWaitingTime = 0f;

    static float AverageTurnAroundTime = 0f;

    static float throughput = 0f;

    static ArrayList<Process3> processes = new ArrayList<>();

    static ArrayList<Process3> temp = new ArrayList<>();

    static int currentTime = 0;

    public static void calculate(String name, int at, int bt) {

        int st = currentTime;

        int ct = st + bt;

        currentTime = currentTime + bt;

        int tat = ct - at;

        int wt = tat - bt;

        AverageWaitingTime += wt;

        AverageTurnAroundTime += tat;

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

    }

    public static void main(String[] args) {

        // TODO Auto-generated method stub

        Scanner sc = new Scanner(System.in);

        Boolean flag = false;

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

        int n = sc.nextInt();

        System.out.print("Enter process Arrival & Burst time :\n");

        for (int i = 0; i < n; i++) {

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

            int arrivalTime = sc.nextInt();

            int burstTime = sc.nextInt();

            processes.add(new Process3("P" + (i + 1), arrivalTime, burstTime));

        }

        // Sort based on arrival time

        Collections.sort(processes, new ArrivalComparator1());

        System.out.println("PID\t\t" + "Arrival\t\t" + "Burst\t\t" + "Starting\t\t" + "Compleition\t\t" + "TurnAround\t\t" + "Waiting");

        // Calculate 1st process

        calculate(processes.get(0).name, processes.get(0).arrivalTime, processes.get(0).burstTime);

        processes.remove(0);

        while (!processes.isEmpty() || !temp.isEmpty()) {

            // Add processes that have arrived into the temporary list

            if (!processes.isEmpty()) {

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

                    if (processes.get(i).arrivalTime <= currentTime) {

                        temp.add(processes.get(i));

                        processes.remove(processes.get(i));

                    }

                }

            }

            if (!processes.isEmpty()) {

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

                    if (processes.get(i).arrivalTime <= currentTime) {

                        temp.add(processes.get(i));

                        processes.remove(processes.get(i));

                    }

                }

            }

            // Sort based on burst time

            Collections.sort(temp, new BurstComparator1());

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

                if (temp.get(i).age == 0) {

                    flag = true;

                    calculate(temp.get(i).name, temp.get(i).arrivalTime, temp.get(i).burstTime);

                    temp.remove(temp.get(i));

                } else {

                    flag = false;

                }

            }

            if (flag == false) {

                calculate(temp.get(0).name, temp.get(0).arrivalTime, temp.get(0).burstTime);

                for (Process3 p : temp) {

                    p.age -= 1;

                }

                temp.remove(temp.get(0));

                flag = false;

            }

        }

        System.out.println("Average Waiting Time: " + AverageWaitingTime / n);

        System.out.println("Average Turn Around Time: " + AverageTurnAroundTime / n);

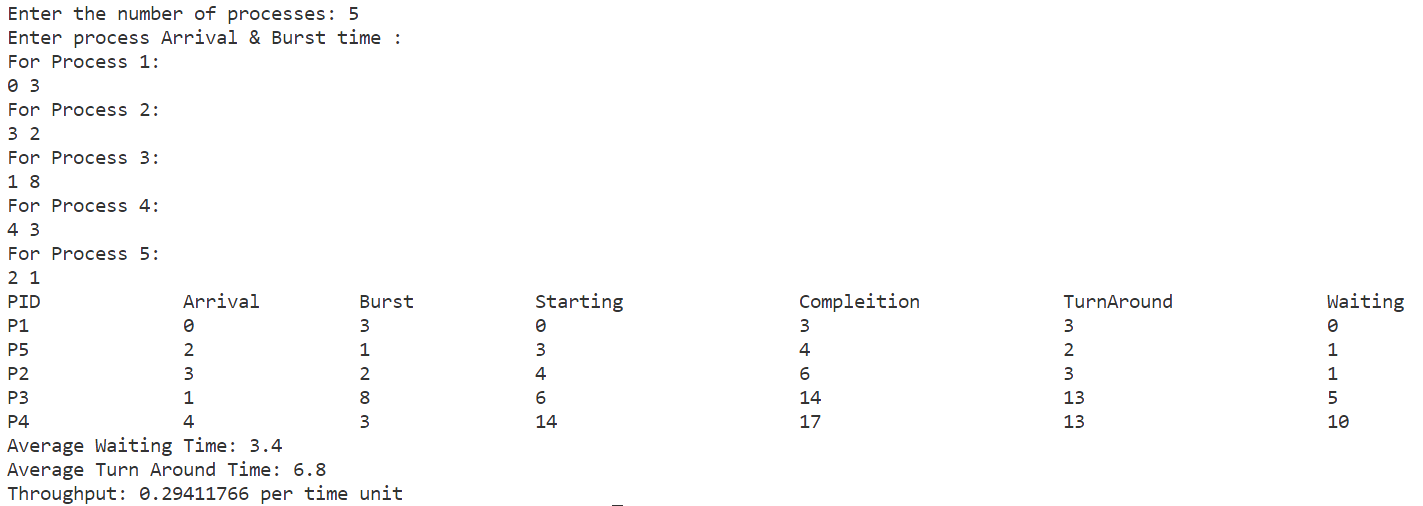
        throughput = (float) n / currentTime;

        System.out.println("Throughput: " + throughput+" per time unit");

    }}

**FORMATTED INPUT:**

|  |  |  |
| --- | --- | --- |
| **PROCESS ID** | **ARRIVAL TIME** | **BURST TIME** |
| 1 | 0 | 3 |
| 2 | 3 | 2 |
| 3 | 1 | 8 |
| 4 | 4 | 3 |
| 5 | 2 | 1 |

**OUTPUT:**

**RESULT:**

The program has been executed successfully and the output is noted.

**PROGRAM CODE:**

import java.util.\*;

class Process {

    int id;

    int arrivalTime;

    int tempBT;

    int burstTime;

    int priority;

    int startTime;

    int completionTime;

    int waitingTime;

    int turnaroundTime;

    boolean executed;

    public Process(int id, int arrivalTime, int burstTime, int priority) {

        this.id = id;

        this.arrivalTime = arrivalTime;

        this.burstTime = burstTime;

        this.tempBT = burstTime;

        this.priority = priority;

        this.startTime = 0;

        this.completionTime = 0;

        this.waitingTime = 0;

        this.turnaroundTime = 0;

        this.executed = false;

    }

}

public class preemptive {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

        List<Process> processes = new ArrayList<>();

        System.out.println("Enter Arrival, Burst, Priority: ");

        for (int i = 0; i < n; i++) {

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

            int arrivalTime = scanner.nextInt();

            int burstTime = scanner.nextInt();

            int priority = scanner.nextInt();

            Process process = new Process(i + 1, arrivalTime, burstTime, priority);

            processes.add(process);

        }

        // Sort processes based on arrival time

        processes.sort(Comparator.comparingInt(p -> p.arrivalTime));

        int currentTime = 0;

        int completedProcesses = 0;

        while (completedProcesses < n) {

            int highestPriority = Integer.MAX\_VALUE;

            int selectedProcessIndex = -1;

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

                Process process = processes.get(i);

                if (!process.executed && process.arrivalTime <= currentTime && process.priority < highestPriority) {

                    highestPriority = process.priority;

                    selectedProcessIndex = i;

                }

            }

            if (selectedProcessIndex == -1) {

                currentTime++;

                continue;

            }

            Process selectedProcess = processes.get(selectedProcessIndex);

            selectedProcess.executed = true;

            selectedProcess.startTime = currentTime;

            selectedProcess.completionTime = currentTime + 1;

            selectedProcess.turnaroundTime = selectedProcess.completionTime - selectedProcess.arrivalTime;

            selectedProcess.waitingTime = selectedProcess.turnaroundTime - selectedProcess.tempBT;

            currentTime++;

            if (selectedProcess.burstTime > 1) {

                selectedProcess.burstTime--;

                selectedProcess.executed = false;

            } else {

                completedProcesses++;

            }

        }

        int averageWaitingTime = 0, averageTurnaroundTime = 0;

        for (Process p : processes) {

            averageTurnaroundTime += p.turnaroundTime;

            averageWaitingTime += p.waitingTime;

        }

        float throughput = (float) n / currentTime;

        System.out.println("\nProcess\t\tArrival\t\tBurst\t\tCompletion\t\tWaiting\t\tTurnaround");

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

        for (Process process : processes) {

            System.out.println(process.id + "\t\t" + process.arrivalTime + "\t\t" + process.tempBT + "\t\t" + process.completionTime + "\t\t\t" + process.waitingTime + "\t\t\t" + process.turnaroundTime);

        }

        System.out.println("Average Waiting Time: " + (averageWaitingTime / n));

        System.out.println("Average Turnaround Time: " + (averageTurnaroundTime / n));

        System.out.println("Throughput: " + throughput+" per time unit");

        scanner.close();

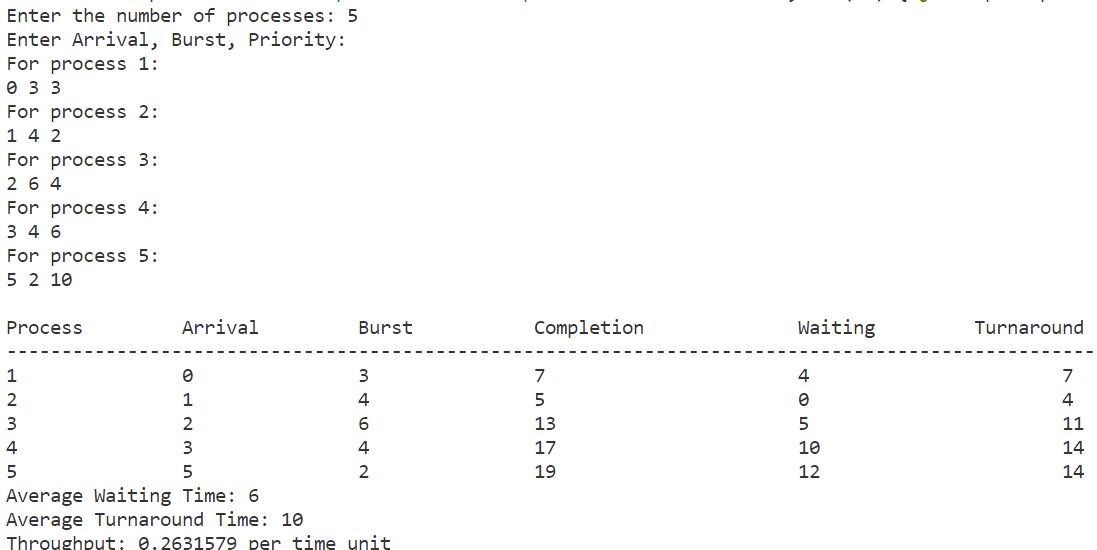
    }

}

**FORMATTED INPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| **PROCESS ID** | **ARRIVAL TIME** | **BURST TIME** | **PRIORITY** |
| 1 | 0 | 3 | 3 |
| 2 | 1 | 4 | 2 |
| 3 | 2 | 6 | 4 |
| 4 | 3 | 4 | 6 |
| 5 | 5 | 2 | 10 |

**OUTPUT:**



**RESULT:**

The program has been executed successfully and the output is noted.

**PROGRAM CODE:**

import java.util.\*;

class Process1 {

    int id;

    int arrivalTime;

    int burstTime;

    int tempBT;

    int priority;

    int startTime;

    int completionTime;

    int waitingTime;

    int turnaroundTime;

    boolean completed;

    public Process1(int id, int arrivalTime, int burstTime, int priority) {

        this.id = id;

        this.arrivalTime = arrivalTime;

        this.burstTime = burstTime;

        this.tempBT = burstTime;

        this.priority = priority;

        this.startTime = 0;

        this.completionTime = 0;

        this.waitingTime = 0;

        this.turnaroundTime = 0;

        this.completed = false;

    }

}

public class nonpreemptive {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

        List<Process1> processes = new ArrayList<>();

        System.out.println("Enter Arrival, Burst, Priority:");

        for (int i = 0; i < n; i++) {

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

            int arrivalTime = scanner.nextInt();

            int burstTime = scanner.nextInt();

            int priority = scanner.nextInt();

            Process1 process = new Process1(i + 1, arrivalTime, burstTime, priority);

            processes.add(process);

        }

        // Sort processes based on arrival time and priority

        int currentTime = 0;

        int completedProcesses = 0;

        while (completedProcesses < n) {

            Process1 selectedProcess = null;

            int highestPriority = Integer.MAX\_VALUE;

            for (Process1 process : processes) {

                if (process.arrivalTime <= currentTime && !process.completed && process.priority < highestPriority) {

                    highestPriority = process.priority;

                    selectedProcess = process;

                }

            }

            if (selectedProcess == null) {

                currentTime++;

                continue;

            }

            selectedProcess.startTime = currentTime;

            selectedProcess.completionTime = selectedProcess.startTime + selectedProcess.burstTime;

            currentTime += selectedProcess.burstTime;

            selectedProcess.turnaroundTime = selectedProcess.completionTime - selectedProcess.arrivalTime;

            selectedProcess.waitingTime = selectedProcess.turnaroundTime - selectedProcess.burstTime;

            selectedProcess.completed = true;

            completedProcesses++;

        }

        float averageWaitingTime = 0f, averageTurnaroundTime = 0f;

        System.out.println("\nProcess\t\tArrival\t\tBurst\t\tPriority\t\tStart\t\tCompletion\t\tWaiting\t\tTurnaround");

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

        for (Process1 process : processes) {

            averageTurnaroundTime += process.turnaroundTime;

            averageWaitingTime += process.waitingTime;

            System.out.println(process.id + "\t\t" + process.arrivalTime + "\t\t" + process.tempBT + "\t\t" + process.priority + "\t\t\t" +

                    process.startTime + "\t\t" + process.completionTime + "\t\t\t" + process.waitingTime +

                    "\t\t\t" + process.turnaroundTime);

        }

        System.out.println("Average Waiting Time: " + (averageWaitingTime / n));

        System.out.println("Average Turnaround Time: " + (averageTurnaroundTime / n));

        float throughput = (float) n / currentTime;

        System.out.println("Throughput: " + throughput+" per time unit");

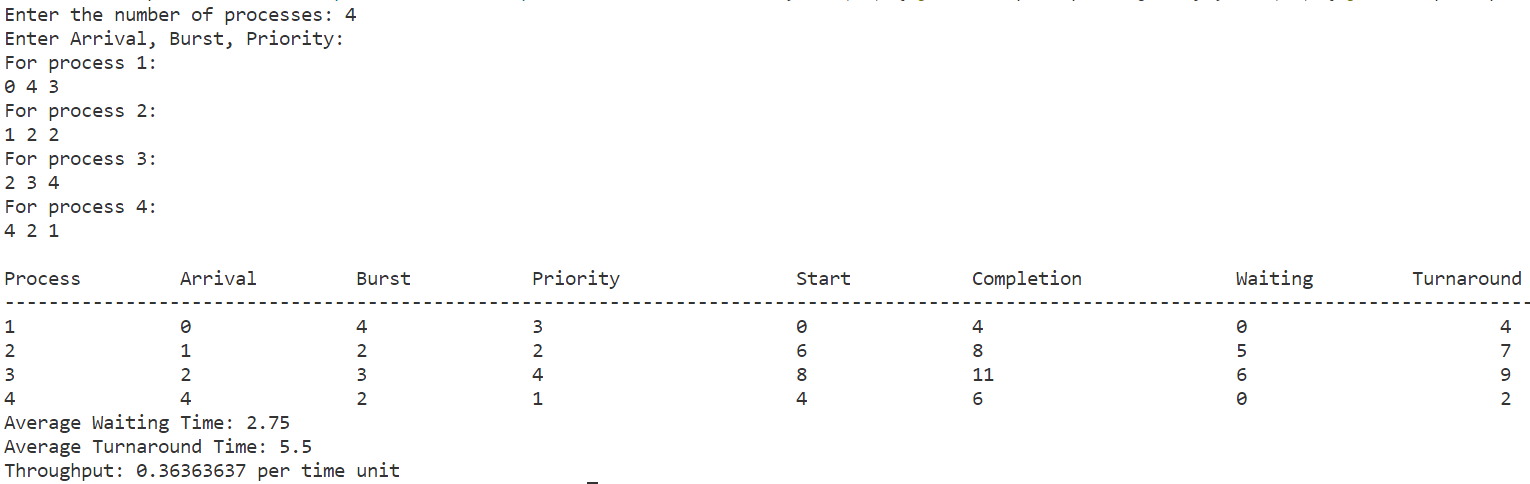
        scanner.close();

    }

}

**FORMATTED INPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| **PROCESS ID** | **ARRIVAL TIME** | **BURST TIME** | **PRIORITY** |
| 1 | 0 | 4 | 3 |
| 2 | 1 | 2 | 2 |
| 3 | 2 | 3 | 4 |
| 4 | 4 | 2 | 1 |

**OUTPUT:**

**RESULT:**

The program has been executed successfully and the output is noted.

**PROGRAM CODE:**

import java.util.\*;

class Process2 {

    int id;

    int arrivalTime;

    int burstTime;

    int priority;

    int tempPriority;

    int startTime;

    int completionTime;

    int waitingTime;

    int turnaroundTime;

    boolean completed;

    public Process2(int id, int arrivalTime, int burstTime, int priority) {

        this.id = id;

        this.arrivalTime = arrivalTime;

        this.burstTime = burstTime;

        this.priority = priority;

        this.tempPriority = priority;

        this.startTime = 0;

        this.completionTime = 0;

        this.waitingTime = 0;

        this.turnaroundTime = 0;

        this.completed = false;

    }

}

public class prioritystarvation {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

        List<Process2> processes = new ArrayList<>();

        System.out.println("Enter Arrival, Burst, Priority:");

        for (int i = 0; i < n; i++) {

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

            int arrivalTime = scanner.nextInt();

            int burstTime = scanner.nextInt();

            int priority = scanner.nextInt();

            Process2 process = new Process2(i + 1, arrivalTime, burstTime, priority);

            processes.add(process);

        }

        // Sort processes based on arrival time

        processes.sort(Comparator.comparingInt((Process2 p) -> p.arrivalTime));

        int currentTime = 0;

        int completedProcesses = 0;

        while (completedProcesses < n) {

            Process2 selectedProcess = null;

            int highestPriority = Integer.MAX\_VALUE;

            for (Process2 process : processes) {

                if (process.arrivalTime <= currentTime && process.priority < highestPriority && !process.completed) {

                    highestPriority = process.priority;

                    selectedProcess = process;

                }

            }

            if (selectedProcess == null) {

                currentTime++;

                continue;

            }

            selectedProcess.startTime = currentTime;

            selectedProcess.completionTime = selectedProcess.startTime + selectedProcess.burstTime;

            currentTime += selectedProcess.burstTime;

            selectedProcess.turnaroundTime = selectedProcess.completionTime - selectedProcess.arrivalTime;

            selectedProcess.waitingTime = selectedProcess.turnaroundTime - selectedProcess.burstTime;

            selectedProcess.completed = true;

            completedProcesses++;

            // Aging: Increase the priority of waiting processes

            for (Process2 process : processes) {

                if (process.arrivalTime <= currentTime && process != selectedProcess) {

                    process.priority--;

                }

            }

        }

        float averageWaitingTime = 0f, averageTurnaroundTime = 0f;

        System.out.println("\nProcess\t\tArrival\t\tBurst\t\tPriority\tCompletion\t\tWaiting\t\tTurnaround");

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

        for (Process2 process : processes) {

            averageTurnaroundTime += process.turnaroundTime;

            averageWaitingTime += process.waitingTime;

            System.out.println(process.id + "\t\t" + process.arrivalTime + "\t\t" + process.burstTime + "\t\t" + process.tempPriority + "\t\t" +

                    +process.completionTime + "\t\t" + process.waitingTime +

                    "\t\t" + process.turnaroundTime);

        }

        System.out.println("Average Waiting Time: " + averageWaitingTime / n);

        System.out.println("Average Turnaround Time: " + averageTurnaroundTime / n);

        float throughput = (float) n / currentTime;

        System.out.println("Throughput: " + throughput+" per time unit");

        scanner.close();

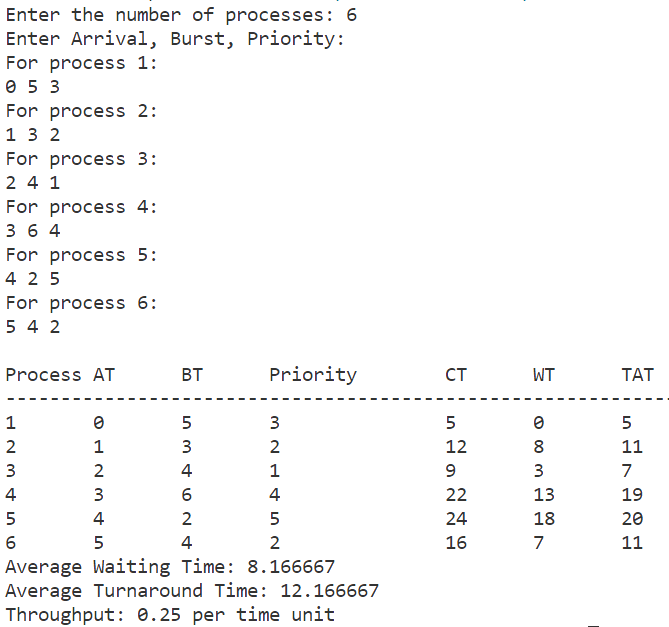
    }

}

**FORMATTED INPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| **PROCESS ID** | **ARRIVAL TIME** | **BURST TIME** | **PRIORITY** |
| 1 | 0 | 5 | 3 |
| 2 | 1 | 3 | 2 |
| 3 | 2 | 4 | 1 |
| 4 | 3 | 6 | 4 |
| 5 | 4 | 2 | 5 |
| 6 | 5 | 4 | 2 |

**OUTPUT:**



**RESULT:**

The program has been executed successfully and the output is noted.

**PROGRAM CODE:**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class Process1 {

    int pid; // Process ID

    int bt; // Burst Time

    int art; // Arrival Time

    public Process1(int pid, int bt, int art) {

        this.pid = pid;

        this.bt = bt;

        this.art = art;

    }

}

public class srtf {

    // Method to find the waiting time for all processes

    static void findWaitingTime(List<Process1> processes, int n, int wt[], int ct[]) {

        int rt[] = new int[n];

        // Copy the burst time into rt[]

        for (int i = 0; i < n; i++)

            rt[i] = processes.get(i).bt;

        int complete = 0, t = 0, minm = Integer.MAX\_VALUE;

        int shortest = -1, finishTime;

        boolean check = false;

        // Process until all processes get completed

        while (complete != n) {

            // Find the process with the minimum remaining time among the processes that have arrived till the current time

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

                if (processes.get(j).art <= t && rt[j] < minm && rt[j] > 0) {

                    minm = rt[j];

                    shortest = j;

                    check = true;

                }

            }

            if (!check) {

                t++;

                continue;

            }

            // Reduce the remaining time by one

            rt[shortest]--;

            // Update the minimum remaining time

            minm = rt[shortest];

            if (minm == 0)

                minm = Integer.MAX\_VALUE;

            // If a process gets completely executed

            if (rt[shortest] == 0) {

                complete++;

                check = false;

                // Find the finish time of the current process

                finishTime = t + 1;

                // Calculate completion time

                ct[shortest] = finishTime;

                // Calculate waiting time

                wt[shortest] = finishTime - processes.get(shortest).bt - processes.get(shortest).art;

                if (wt[shortest] < 0)

                    wt[shortest] = 0;

            }

            // Increment time

            t++;

        }

    }

    // Method to calculate turnaround time

    static void findTurnaroundTime(List<Process1> processes, int n, int wt[], int tat[]) {

        // Calculate turnaround time by adding burst time and waiting time

        for (int i = 0; i < n; i++)

            tat[i] = processes.get(i).bt + wt[i];

    }

    // Method to calculate average time

    static void findAverageTime(List<Process1> processes, int n) {

        int wt[] = new int[n], tat[] = new int[n];

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

        int ct[] = new int[n];

        // Find waiting time of all processes

        findWaitingTime(processes, n, wt, ct);

        // Find turnaround time for all processes

        findTurnaroundTime(processes, n, wt, tat);

        // Display process details

        System.out.println("Process\tArrival\tBurst\tCompletion\tWaiting\tTurnaround");

        for (int i = 0; i < n; i++) {

            System.out.println(

                    processes.get(i).pid + "\t" +

                            processes.get(i).art + "\t" +

                            processes.get(i).bt + "\t" +

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

                            wt[i] + "\t\t" +

                            tat[i]

            );

            total\_wt += wt[i];

            total\_tat += tat[i];

        }

        // Calculate average waiting time and average turnaround time

        float avg\_wt = (float) total\_wt / n;

        float avg\_tat = (float) total\_tat / n;

        System.out.println("\nAverage Waiting Time: " + avg\_wt);

        System.out.println("Average Turnaround Time: " + avg\_tat);

        // Calculate throughput

        float throughput = (float) n / ct[n - 1];

        System.out.println("Throughput: " + throughput+" per time unit");

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

        List<Process1> processes = new ArrayList<>();

        System.out.println("Enter Arrival Time and Burst Time:");

        for (int i = 0; i < n; i++) {

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

            int arrivalTime = scanner.nextInt();

            int burstTime = scanner.nextInt();

            Process1 process = new Process1(i + 1, burstTime, arrivalTime);

            processes.add(process);

        }

        findAverageTime(processes, processes.size());

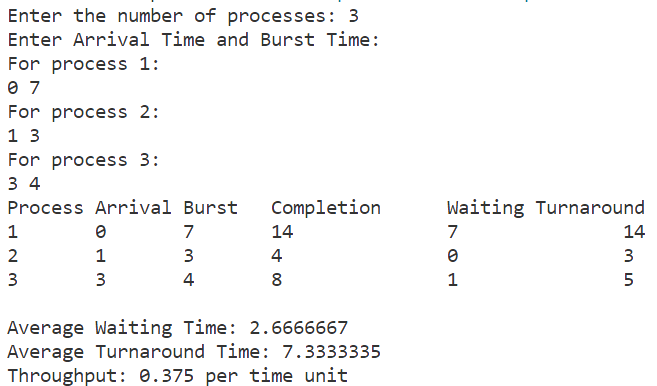
    }

}

**FORMATTED INPUT:**

|  |  |  |
| --- | --- | --- |
| **PROCESS ID** | **ARRIVAL TIME** | **BURST TIME** |
| 1 | 0 | 7 |
| 2 | 1 | 3 |
| 3 | 3 | 4 |

**OUTPUT:**



**RESULT:**

The program has been executed successfully and the output is noted.

**PROGRAM CODE:**

import java.util.Scanner;

class RoundRobinScheduler {

    private Process3[] processes;

    private int timeQuantum;

    public RoundRobinScheduler(int numProcesses, int timeQuantum) {

        processes = new Process3[numProcesses];

        this.timeQuantum = timeQuantum;

    }

    public void inputProcessDetails(Scanner scanner) {

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

            System.out.println("Enter the arrival time and burst time for Process " + (i + 1) + ":");

            int arrivalTime = scanner.nextInt();

            int burstTime = scanner.nextInt();

            processes[i] = new Process3(i + 1, arrivalTime, burstTime);

        }

    }

    public void runScheduler() {

        int[] remainingTime = new int[processes.length];

        int currentTime = 0;

        boolean allProcessesCompleted = false;

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

            remainingTime[i] = processes[i].getBurstTime();

        }

        while (!allProcessesCompleted) {

            allProcessesCompleted = true;

            for (Process3 process : processes) {

                int index = process.getId() - 1;

                if (remainingTime[index] > 0) {

                    allProcessesCompleted = false;

                    if (remainingTime[index] <= timeQuantum) {

                        currentTime += remainingTime[index];

                        remainingTime[index] = 0;

                        process.setCompletionTime(currentTime);

                    } else {

                        currentTime += timeQuantum;

                        remainingTime[index] -= timeQuantum;

                    }

                }

            }

        }

    }

    public void calculateTimes() {

        for (Process3 process : processes) {

            int waitingTime = process.getCompletionTime() - process.getArrivalTime() - process.getBurstTime();

            process.setWaitingTime(waitingTime);

            int turnAroundTime = process.getCompletionTime() - process.getArrivalTime();

            process.setTurnAroundTime(turnAroundTime);

        }

    }

    public void printProcessDetails() {

        System.out.println("\nProcess\tArrival Time\tBurst Time\tCompletion Time\t\tWaiting Time\tTurnaround Time");

        for (Process3 process : processes) {

            System.out.println(process.getId() + "\t\t" + process.getArrivalTime() + "\t\t" + process.getBurstTime()

                    + "\t\t" + process.getCompletionTime() + "\t\t\t" + process.getWaitingTime()

                    + "\t\t" + process.getTurnAroundTime());

        }

    }

    public void printAverageTimes() {

        int totalWaitingTime = 0;

        int totalTurnAroundTime = 0;

        for (Process3 process : processes) {

            totalWaitingTime += process.getWaitingTime();

            totalTurnAroundTime += process.getTurnAroundTime();

        }

        double averageWaitingTime = (double) totalWaitingTime / processes.length;

        double averageTurnAroundTime = (double) totalTurnAroundTime / processes.length;

        System.out.println("\nAverage Waiting Time: " + averageWaitingTime);

        System.out.println("Average Turnaround Time: " + averageTurnAroundTime);

        // Calculate throughput

        double throughput = (double) processes.length / processes[processes.length - 1].getCompletionTime();

        System.out.println("Throughput: " + throughput+" per time unit");

    }

}

class Process3 {

    private int id;

    private int arrivalTime;

    private int burstTime;

    private int completionTime;

    private int waitingTime;

    private int turnAroundTime;

    public Process3(int id, int arrivalTime, int burstTime) {

        this.id = id;

        this.arrivalTime = arrivalTime;

        this.burstTime = burstTime;

        this.completionTime = 0;

        this.waitingTime = 0;

        this.turnAroundTime = 0;

    }

    public int getId() {

        return id;

    }

    public int getArrivalTime() {

        return arrivalTime;

    }

    public int getBurstTime() {

        return burstTime;

    }

    public int getCompletionTime() {

        return completionTime;

    }

    public void setCompletionTime(int completionTime) {

        this.completionTime = completionTime;

    }

    public int getWaitingTime() {

        return waitingTime;

    }

    public void setWaitingTime(int waitingTime) {

        this.waitingTime = waitingTime;

    }

    public int getTurnAroundTime() {

        return turnAroundTime;

    }

    public void setTurnAroundTime(int turnAroundTime) {

        this.turnAroundTime = turnAroundTime;

    }

}

public class roundrobin {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int numProcesses = scanner.nextInt();

        System.out.print("Enter the time quantum: ");

        int timeQuantum = scanner.nextInt();

        RoundRobinScheduler scheduler = new RoundRobinScheduler(numProcesses, timeQuantum);

        scheduler.inputProcessDetails(scanner);

        scheduler.runScheduler();

        scheduler.calculateTimes();

        scheduler.printProcessDetails();

        scheduler.printAverageTimes();

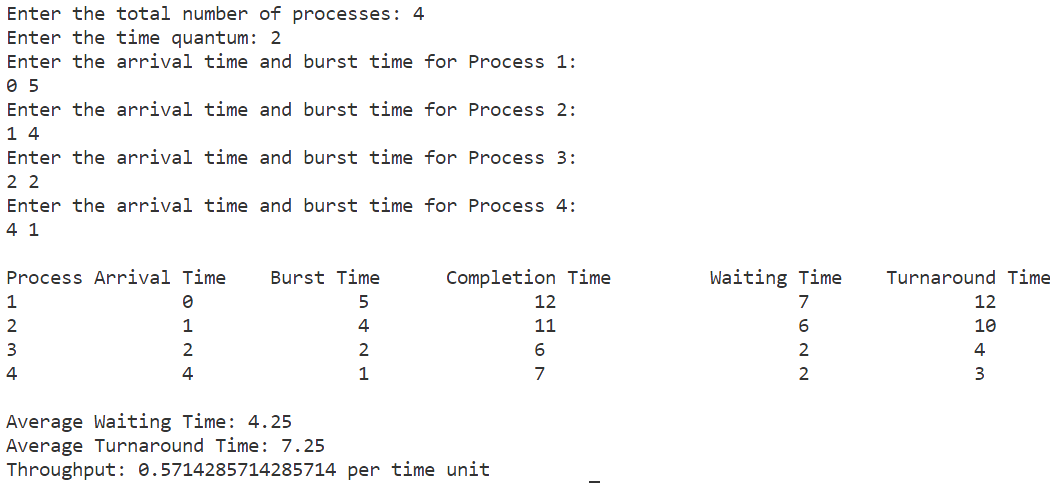
    }

}

**FORMATTED INPUT:**

|  |  |  |
| --- | --- | --- |
| **PROCESS ID** | **ARRIVAL TIME** | **BURST TIME** |
| 1 | 0 | 5 |
| 2 | 1 | 4 |
| 3 | 2 | 3 |
| 4 | 4 | 1 |

**OUTPUT:**



**RESULT:**

The program has been executed successfully and the output is noted.