**NAME: AKSH NISHAR**

**SAP ID: 60004220123**

**DIV: B3**

**EXPERIMENT 3: CPU scheduling algorithms like FCFS, SJF, Round Robin etc.**

**OPERATING SYSTEMS – EXPERIMENT 4**

**AIM**: CPU scheduling algorithms like FCFS, SJF, Round Robin etc.

**PROBLEM STATEMENT**:

1. Perform comparative assessment of various Scheduling Policies like FCFS, SJF (preemptive and non-preemptive), Priority (preemptive and non-preemptive) and Round Robin.
2. Take the input processes, their arrival time, burst time, priority, quantum from user.

**THEORY:**

Scheduling algorithms are used when more than one process is executable and the OS has to decide which one to run first. Terms used

1. Arrival time: The process at which the process is given to CPU.

2. Burst time: The amount of time each process takes for execution.

3. Completion time: The time at which the process has completely executed.

4. Response time: The difference between the time when the process starts execution and the arrival time.

5. Turnaround time: The difference between the time when the process completes execution and the arrival time.

6. Waiting Time: The difference between the turnaround time and burst time.

First Come First Serve (FCFS) The processes are executed in the order in which they have been submitted.

Shortest Job First (SJF) The processes are checked at each arrival time and the process which have the shortest remaining burst time at that moment gets executed first. This is non-preemptive algorithm.

Shortest Remaining Time Next (SRTN) This is preemptive version of SJF. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Similar to SJF, if a new process that has arrived has shorter burst time than the current running process, the previous process is pre-empted and new process is allocated CPU.

Priority Scheduling Each process is assigned a priority and executable process with highest priority is allowed to run.

Round Robin Each process is assigned a time interval called its quantum (time slice) If the process is still running at the end of the quantum the CPU is preempted and given to another process, and this continues in circular fashion, till all the processes are completely executed.

FCFS (First Come First Serve)

//Aksh Nishar 60004220123

import java.util.Scanner;

class FCFSpdf {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        int n;

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

        n = sc.nextInt();

        int p[] = new int[n]; // Process id array

        int at[] = new int[n]; // Arrival Time array

        int bt[] = new int[n]; // Burst Time array

        int i, j, min1, min2, min3;

        for (i = 0; i < n; i++) {

            p[i] = i + 1;

            System.out.print("Enter arrival time: ");

            at[i] = sc.nextInt();

            System.out.print("Enter burst time: ");

            bt[i] = sc.nextInt();

        }

        for (i = 0; i < n; i++) {

            min1 = at[i];

            min2 = bt[i];

            min3 = p[i];

            j = i - 1;

            while (j >= 0 && at[j] > min1) {

                at[j + 1] = at[j];

                bt[j + 1] = bt[j];

                p[j + 1] = p[j];

                j--;

            }

            at[j + 1] = min1;

            bt[j + 1] = min2;

            p[j + 1] = min3;

        }

        int ct[] = new int[n]; // Completion time array

        int cur\_t = 0; // Current Time

        int st[] = new int[n]; // Start Time array

        for (i = 0; i < n; i++) {

            ct[i] = cur\_t + bt[i];

            st[i] = cur\_t;

            cur\_t = ct[i];

        }

        int tat[] = new int[n]; // Turnaround Time array

        int wt[] = new int[n]; // Waiting Time array

        float sum1 = 0, sum2 = 0; // sum1 for tat and sum2 for wt

        System.out.println("\nProcess No.\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.");

        for (i = 0; i < n; i++) {

            tat[i] = ct[i] - at[i];

            wt[i] = tat[i] - bt[i];

            System.out.println(

                    "Process" + p[i] + "\t" + at[i] + "\t" + bt[i] + "\t" + ct[i] + "\t" + tat[i] + "\t" + wt[i]);

            sum1 += tat[i];

            sum2 += wt[i];

        }

        System.out.println("Average Turn Around Time: " + (sum1 / n));

        System.out.println("Average Waiting Time: " + (sum2 / n));

        System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion Time");

        for (i = 0; i < n; i++) {

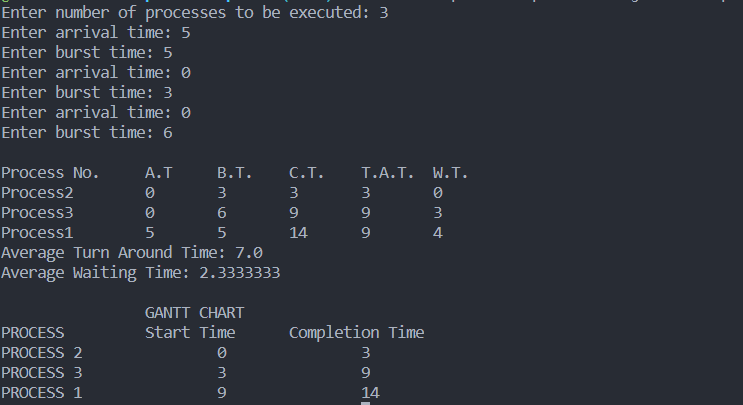
            System.out.println("PROCESS " + p[i] + "\t\t" + st[i] + "\t\t" + ct[i]);

        }

    }

}

Output:



SRTF (Shortest Remaining Time first)

//Aksh Nishar 60004220123

import java.util.Scanner;

class SRTF {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        int n;

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

        n = sc.nextInt();

        int p[] = new int[n]; // Process id array

        int at[] = new int[n]; // Arrival time array

        int bt[] = new int[n]; // Burst time array

        int flag[] = new int[n]; // To check if process is completed

        int i, j, min1, min2, min3;

        for (i = 0; i < n; i++) {

            p[i] = i + 1;

            System.out.print("Enter arrival time: ");

            at[i] = sc.nextInt();

            System.out.print("Enter burst time: ");

            bt[i] = sc.nextInt();

            flag[i] = 0;

        }

        for (i = 0; i < n; i++) {

            min1 = at[i];

            min2 = bt[i];

            min3 = p[i];

            j = i - 1;

            while (j >= 0 && at[j] > min1) {

                at[j + 1] = at[j];

                bt[j + 1] = bt[j];

                p[j + 1] = p[j];

                j--;

            }

            at[j + 1] = min1;

            bt[j + 1] = min2;

            p[j + 1] = min3;

        }

        int cur\_t = 0; // Current time

        int st[] = new int[n]; // Starting time of each process

        int ct[] = new int[n]; // completion time array

        int tot = 0; // To count number of processes completed

        int minbt = 1000; // To store the shortest bt

        int c = 0; // To track id of process to be scheduled next

        while (tot < n) {

            for (i = 0; i < n; i++) {

                if ((at[i] <= cur\_t) && (flag[i] == 0) && (bt[i] <= minbt)) {

                    minbt = bt[i];

                    c = i;

                }

            }

            ct[c] = cur\_t + minbt;

            st[c] = cur\_t;

            flag[c] = 1;

            cur\_t = ct[c];

            tot++;

            minbt = 1000; // reset so that bt values of remaining processes are compared

        }

        int tat[] = new int[n]; // Turnaround Time array

        int wt[] = new int[n]; // Waiting Time array

        float sum1 = 0, sum2 = 0; // sum1 for tat and sum2 for wt

        System.out.println("\nProcess No.\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.");

        for (i = 0; i < n; i++) {

            tat[i] = ct[i] - at[i];

            wt[i] = tat[i] - bt[i];

            System.out.println(

                    "Process" + p[i] + "\t" + at[i] + "\t" + bt[i] + "\t" + ct[i] + "\t" + tat[i] + "\t" + wt[i]);

            sum1 += tat[i];

            sum2 += wt[i];

        }

        System.out.println("Average Turn Around Time: " + (sum1 / n));

        System.out.println("Average Waiting Time: " + (sum2 / n));

        // To prepare Gantt Chart processes are soted according to start time

        for (i = 0; i < n; i++) {

            min1 = st[i];

            min2 = ct[i];

            min3 = p[i];

            j = i - 1;

            while (j >= 0 && st[j] > min1) {

                st[j + 1] = st[j];

                ct[j + 1] = ct[j];

                p[j + 1] = p[j];

                j--;

            }

            st[j + 1] = min1;

            ct[j + 1] = min2;

            p[j + 1] = min3;

        }

        System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion Time");

        for (i = 0; i < n; i++) {

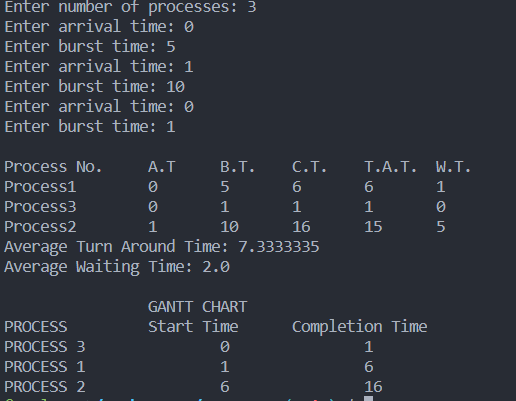
            System.out.println("PROCESS " + p[i] + "\t\t" + st[i] + "\t\t" + ct[i]);

        }

    }

}

Output :

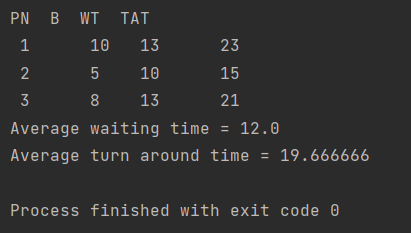


RR (Round Robin)

//Aksh Nishar - 60004220123

public class RR  
{  
 // Method to find the waiting time for all  
 // processes  
 static void findWaitingTime(int processes[], int n,  
 int bt[], int wt[], int quantum)  
 {  
 // Make a copy of burst times bt[] to store remaining  
 // burst times.  
 int rem\_bt[] = new int[n];  
 for (int i = 0 ; i < n ; i++)  
 rem\_bt[i] = bt[i];  
  
 int t = 0; // Current time  
  
 // Keep traversing processes in round robin manner  
 // until all of them are not done.  
 while(true)  
 {  
 boolean done = true;  
  
 // Traverse all processes one by one repeatedly  
 for (int i = 0 ; i < n; i++)  
 {  
 // If burst time of a process is greater than 0  
 // then only need to process further  
 if (rem\_bt[i] > 0)  
 {  
 done = false; // There is a pending process  
  
 if (rem\_bt[i] > quantum)  
 {  
 // Increase the value of t i.e. shows  
 // how much time a process has been processed  
 t += quantum;  
  
 // Decrease the burst\_time of current process  
 // by quantum  
 rem\_bt[i] -= quantum;  
 }  
  
 // If burst time is smaller than or equal to  
 // quantum. Last cycle for this process  
 else  
 {  
 // Increase the value of t i.e. shows  
 // how much time a process has been processed  
 t = t + rem\_bt[i];  
  
 // Waiting time is current time minus time  
 // used by this process  
 wt[i] = t - bt[i];  
  
 // As the process gets fully executed  
 // make its remaining burst time = 0  
 rem\_bt[i] = 0;  
 }  
 }  
 }  
  
 // If all processes are done  
 if (done == true)  
 break;  
 }  
 }  
  
 // Method to calculate turn around time  
 static void findTurnAroundTime(int processes[], int n,  
 int bt[], int wt[], int tat[])  
 {  
 // calculating turnaround time by adding  
 // bt[i] + wt[i]  
 for (int i = 0; i < n ; i++)  
 tat[i] = bt[i] + wt[i];  
 }  
  
 // Method to calculate average time  
 static void findavgTime(int processes[], int n, int bt[],  
 int quantum)  
 {  
 int wt[] = new int[n], tat[] = new int[n];  
 int total\_wt = 0, total\_tat = 0;  
  
 // Function to find waiting time of all processes  
 *findWaitingTime*(processes, n, bt, wt, quantum);  
  
 // Function to find turn around time for all processes  
 *findTurnAroundTime*(processes, n, bt, wt, tat);  
  
 // Display processes along with all details  
 System.*out*.println("PN " + " B " +  
 " WT " + " TAT");  
  
 // Calculate total waiting time and total turn  
 // around time  
 for (int i=0; i<n; i++)  
 {  
 total\_wt = total\_wt + wt[i];  
 total\_tat = total\_tat + tat[i];  
 System.*out*.println(" " + (i+1) + "\t\t" + bt[i] +"\t " +  
 wt[i] +"\t\t " + tat[i]);  
 }  
  
 System.*out*.println("Average waiting time = " +  
 (float)total\_wt / (float)n);  
 System.*out*.println("Average turn around time = " +  
 (float)total\_tat / (float)n);  
 }  
  
 // Driver Method  
 public static void main(String[] args)  
 {  
 // process id's  
 int processes[] = { 1, 2, 3};  
 int n = processes.length;  
  
 // Burst time of all processes  
 int burst\_time[] = {10, 5, 8};  
  
 // Time quantum  
 int quantum = 2;  
 *findavgTime*(processes, n, burst\_time, quantum);  
 }  
}

Output :



Priority

//Aksh Nishar 60004220123

import java.util.Scanner;

class Priority {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        int n;

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

        n = sc.nextInt();

        int p[] = new int[n]; // Process id array

        int at[] = new int[n]; // Arrival time array

        int bt[] = new int[n]; // Burst time array

        int flag[] = new int[n]; // To check if process is completed

        int priority[] = new int[n];

        int i, j, min1, min2, min3, min4;

        int cur\_t = 0; // Current time

        int st[] = new int[n]; // Starting time of each process

        int ct[] = new int[n]; // completion time array

        int tot = 0; // To count number of processes completed

        int minpri = 1000; // To store the highest priority

        int c = 0;

        int b[] = new int[n]; // To store a copy of burst time array

        for (i = 0; i < n; i++) {

            p[i] = i + 1;

            System.out.print("Enter arrival time: ");

            at[i] = sc.nextInt();

            System.out.print("Enter burst time: ");

            bt[i] = sc.nextInt();

            System.out.print("Enter priority [smallest number is high]: ");

            priority[i] = sc.nextInt();

            flag[i] = 0;

            st[i] = -1;

        }

        for (i = 0; i < n; i++) {

            min1 = at[i];

            min2 = bt[i];

            min3 = p[i];

            min4 = priority[i];

            j = i - 1;

            while (j >= 0 && at[j] > min1) {

                at[j + 1] = at[j];

                bt[j + 1] = bt[j];

                p[j + 1] = p[j];

                priority[j + 1] = priority[j];

                j--;

            }

            at[j + 1] = min1;

            bt[j + 1] = min2;

            p[j + 1] = min3;

            priority[j + 1] = min4;

        }

        for (i = 0; i < n; i++)

            b[i] = bt[i];

        while (tot < n) {

            for (i = 0; i < n; i++) {

                if ((priority[i] <= minpri) && (at[i] <= cur\_t) && (flag[i] == 0)) {

                    minpri = priority[i];

                    c = i;

                }

            }

            if (st[c] == -1) {

                st[c] = cur\_t;

                ct[c] = cur\_t;

            }

            b[c]--;

            cur\_t++;

            ct[c]++;

            for (i = 0; i < n; i++) {

                if ((c != i) && (st[i] != -1) && (flag[i] == 0))

                    ct[i]++;

            }

            if (b[c] == 0) {

                flag[c] = 1;

                tot++;

            }

            minpri = 1000;

        }

        int tat = 0; // Turnaround Time

        int wt = 0; // Waiting Time array

        float sum1 = 0, sum2 = 0, rt = 0; // sum1 for tat and sum2 for wt and response time

        System.out.println("\nProcess No.\tPriority\tA.T\tB.T.\tC.T.\tT.A.T.\tW.T.\tR.T.");

        for (i = 0; i < n; i++) {

            tat = ct[i] - at[i];

            wt = tat - bt[i];

            rt = st[i] - at[i];

            System.out.println("Process" + p[i] + "\t" + priority[i] + "\t\t" + at[i] + "\t" + bt[i] + "\t" + ct[i]

                    + "\t" + tat + "\t" + wt + "\t" + rt);

            sum1 += tat;

            sum2 += wt;

        }

        System.out.println("Average Turn Around Time: " + (sum1 / n));

        System.out.println("Average Waiting Time: " + (sum2 / n));

        // To prepare Gantt Chart processes are soted according to start time

        for (i = 0; i < n; i++) {

            min1 = st[i];

            min2 = ct[i];

            min3 = p[i];

            j = i - 1;

            while (j >= 0 && st[j] > min1) {

                st[j + 1] = st[j];

                ct[j + 1] = ct[j];

                p[j + 1] = p[j];

                j--;

            }

            st[j + 1] = min1;

            ct[j + 1] = min2;

            p[j + 1] = min3;

        }

        System.out.println("\n\t\tGANTT CHART\nPROCESS \tStart Time\tCompletion Time");

        for (i = 0; i < n; i++) {

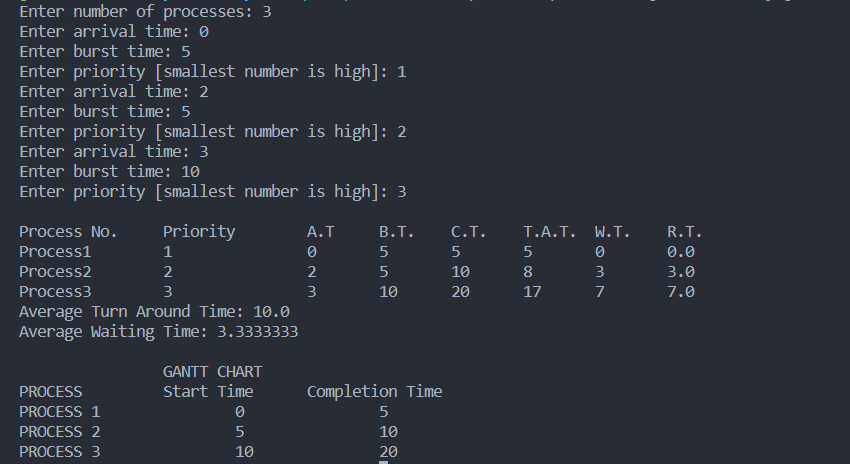
            System.out.println("PROCESS " + p[i] + "\t\t" + st[i] + "\t\t" + ct[i]);

        }

    }

}

Output :



**Conclusion :** Hereby, we have implemented the above CPU scheduling algorithms.