Fokane Sakshi Anil
TE-A-42
Assignment No: 8

job Scheduling Algorithm.

- Aim: Implement job Scheduling algorithm
  1) FCFS 3) Priority
  2) Shortest Job first 4) Round Robin

- Problem Statement: write a java program (using oop features) to implement following Scheduling algorithms FCFS, SJF (preemptive), priority (Non-premptive) Round Robin (preemptive)

- Theory :-

① FCFS : (First Come First Serve):
This is the simplest CPU Scheduling algorithm the process that request the CPU first is the one to which it is allocated first.

- Implementation :-

1) Input the process along with their burst time (bt)
2) Find waiting time (wt) for all process
3) As first process that comes need not to wait so waiting time for process i will be 0 i.e wt[i].
4) find waiting time for all other process i.e for process i -> wt[i] = bt[i-1] + wt[i-1]
5) Find ~~turn~~ turnaround time = waiting time + burst time for all process
6) Find average waiting time =

Scanned by TapScanner

total waiting time / no. of processes

7) Similarly, find average turnaround time = total - turn around time / no. of proce

② Shortest Job first :-

This algorithm associates with it the length of the next CPU Burst when the CPU is available it is assigned to that job with the smallest CPU Burst.

Algorithm : 1) Sort all the process in increasing order according to burst time.

2) Then simply apply FCFS

Shortest remaining time :-

- Shortest remaining time (SRt) is the preemptive version of the SJN algo.
- The processor is allocated to the job to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive System where required CPU time is not known.

③ Priority based Scheduling :-

priority Scheduling is non-preemptive algo and one of the most common and soon

Implementation

i) first i/p the processes with their burst time and priority :

ii) Sort the processes burst time & priority according to the priority.

iii) Now simply apply FCFS algorithm.

④ Round Robin Scheduling :-

Round Robin is CPU Scheduling algorithm where each process is assigned a fixed time slot in cyclic way.

java program for implementat^n FCFS.

```
import java.text.parse.Exception;
class GFG{
Static void find vatingtime (int process[],
  int n, int b[], int wt[]){
    wt[0] = 0;
    for(int i=1; i<n; i++){
        wt[i] = bt[i-1] + wt[i-1];
   }
}
Static void find TurnAroundTime(int proce
  ss[], int n; int bt[], int wt[], int tat[]){
 for(int i=0; i<n; i++){
      tot[i] = bt[i] + wt[i]);
   }
}
static void findAvgTime [int process[],
int n; int bt[] ){
  int wt[] = new int[n], tat[] = New int[n];
  int total-wt = 0, total-tot = 0;
findwaitingTime (process , n, bt , wt);
findturnaroundtime (process, n , bt, wt, tat);
System.out.printf(" process Burst time
waiting ", time turn aroundtime );
}
floats =(float) tot_wt / (float) n;
int t = total_tot /n ;
System.out.printf(" Average waiting
time ' = %.f'', s);
System.out.printf("\n");
System.out.printf(" Average waiting
time = %d", t);
}
Public Static void main(String[] args)
throws parseException{
 int processes[] = {1,2,3};
 int n= processes.length;
 int bursttime() = {10,5,8}
  findavegTime (process.n, burst_time);
}}
```

Java Program for SJF Scheduling:

```
Import java.util.*;
class SJF {
 Public Static Void main(String arg[]){
Scanner sc = new Scanner (system.in);
int n, BT[], WT[], TAT[];
System.out.println ("Enter no. of process:");
n = sc.nextINT()
BT= new int(n+1);
```

Scanned by TapScanner

```
WT = new int(n+1);
TAT = new int(n+1);
float AWT = 0;
System.out.println("Enter Burst time for each");
for( int i=0; i<n; i++){
System.out.println("Enter BT for Process"
        + (i+1));
BT[i] = Sc.nextINT(); }
For(int i=0; i<n; i++){
   WT [i] = 0;    TAT [i] = 0; }
int temp;
for(int i=0; i<n; i++){
 for (int j=0; j<n-1 ; j++){
   if(BT[j] > BT[j+1]){
     temp = BT[j];
     BT[j] = BT[j+1];
     BT[ j+1] = temp;
     temp = WT [j];    WT[j] = WT[j+1];
   WT[j+1] = temp; }
 }
}
 for (int i=0; i<n ; i++)}
   TAT[i] = BT[i] = WT[i];
   WT[i+1] = TAT[i]; } } }
For(int i=0; i<n ; i++){
   TAT[i] = BT[i] + WT [i];
   WT [i+1] = TAT[i];
}
 TAT[n] = WT[n]+ BT[n];
System.out.println("process BT WT TAT");
  for (int i=0; i<n; i++)
  system.out.println(" "+ i + " " +BT[i]+" "+ WT[i]+" "+TAT)
~~for(int j=0; j<n-j; j++)~~
    AWT = WT[i];          AWT = AWT/n;
System.out.println(" *** Avg waiting time = "+AWT);
```

Scanned by TapScanner

```
*******************FCFS**************************
//Name: Fokane Sakshi Anil
// TE-A 42
//  ASSINGNMENT:GROUP_C_1
//Java program for implementation of FCFS
// scheduling
import java.util.*;

public class srtf_c1 {
        public static void main (String args[])
        {
                Scanner sc=new Scanner(System.in);
                System.out.println ("enter no of process:");
                int n= sc.nextInt();
                int pid[] = new int[n]; // it takes pid of process
                int at[] = new int[n]; // at means arrival time
                int bt[] = new int[n]; // bt means burst time
                int ct[] = new int[n]; // ct means complete time
                int ta[] = new int[n];// ta means turn around time
                int wt[] = new int[n];  // wt means waiting time
                int f[] = new int[n]; // f means it is flag it checks process is completed or not
                int k[]= new int[n];  // it is also stores brust time
          int i, st=0, tot=0;
          float avgwt=0, avgta=0;

          for (i=0;i<n;i++)
          {
                pid[i]= i+1;
                System.out.println ("enter process " +(i+1)+ " arrival time:");
                at[i]= sc.nextInt();
                System.out.println("enter process " +(i+1)+ " burst time:");
                bt[i]= sc.nextInt();
                k[i]= bt[i];
                f[i]= 0;
          }

          while(true){
                int min=99,c=n;
                if (tot==n)
                        break;

                for ( i=0;i<n;i++)
                {
                        if ((at[i]<=st) && (f[i]==0) && (bt[i]<min))
                        {
                                min=bt[i];
                                c=i;
                        }
                }

                if (c==n)
                st++;
            else
            {
                bt[c]--;
                st++;
                if (bt[c]==0)
                {
                    ct[c]= st;
                    f[c]=1;
                    tot++;
                }
            }
        }

        for(i=0;i<n;i++)
        {
            ta[i] = ct[i] - at[i];
            wt[i] = ta[i] - k[i];
            avgwt+= wt[i];
            avgta+= ta[i];
        }

        System.out.println("pid  arrival  burst  complete turn waiting");
        for(i=0;i<n;i++)
        {
            System.out.println(pid[i] +"\t"+ at[i]+"\t"+ k[i] +"\t"+ ct[i] +"\t"+ ta[i] +"\t"+ wt[i]);
        }

        System.out.println("\naverage tat is "+ (float)(avgta/n));
        System.out.println("average wt is "+ (float)(avgwt/n));
        sc.close();
      }
}
```

```
_______________OUTPUT_________________
Processes      Burst time     Waiting time Turn around time
 1              7             0               7
 2              3             7               10
 3              6             10              16
 4              4             16              20
 5              2             20              22
Average waiting time = 10.600000
Average turn around time = 15
```

**********************SRTF******************************

```
import java.util.*;
public class srtf_c1 {
        public static void main (String args[])
        {Scanner sc=new Scanner(System.in);
                System.out.println ("enter no of process:");
                int n= sc.nextInt();
                int pid[] = new int[n]; // it takes pid of process
                int at[] = new int[n]; // at means arrival time
                int bt[] = new int[n]; // bt means burst time
                int ct[] = new int[n]; // ct means complete time
                int ta[] = new int[n];// ta means turn around time
                int wt[] = new int[n];  // wt means waiting time
                int f[] = new int[n];  // f means it is flag it checks process is completed or not
                int k[]= new int[n]; // it is also stores brust time
          int i, st=0, tot=0;
          float avgwt=0, avgta=0;
          for (i=0;i<n;i++)  {
                pid[i]= i+1;
                System.out.println ("enter process " +(i+1)+ " arrival time:");
                at[i]= sc.nextInt();
                System.out.println("enter process " +(i+1)+ " burst time:");
                bt[i]= sc.nextInt();
                k[i]= bt[i];
                f[i]= 0;    }
          while(true){
                int min=99,c=n;
                if (tot==n)
                        break;
                for ( i=0;i<n;i++){
                        if ((at[i]<=st) && (f[i]==0) && (bt[i]<min)){
                                min=bt[i];
                                c=i;}}
                if (c==n)
                        st++;
                else{
                        bt[c]--;
                        st++;
                        if (bt[c]==0){
                                ct[c]= st;
                                f[c]=1;
                                tot++;} }
          for(i=0;i<n;i++)   {
                ta[i] = ct[i] - at[i];
                wt[i] = ta[i] - k[i];
                avgwt+= wt[i];
                avgta+= ta[i];    }
          System.out.println("pid  arrival  burst  complete turn waiting");
          for(i=0;i<n;i++) {
                System.out.println(pid[i] +"\t"+ at[i]+"\t"+ k[i] +"\t"+ ct[i] +"\t"+ ta[i] +"\t"+ wt[i]); }
          System.out.println("\naverage tat is "+ (float)(avgta/n));
          System.out.println("average wt is "+ (float)(avgwt/n));
          sc.close();}}
```

_______________OUTPUT_________________________

```
enter no of process:
3
```

```
enter process 1 arrival time:
2
enter process 1 burst time:
4
enter process 2 arrival time:
5
enter process 2 burst time:
2
enter process 3 arrival time:
6
enter process 3 burst time:
3
pid  arrival  burst  complete turn waiting
1       2       4       6       4       0
2       5       2       8       3       1
3       6       3       11      5       2

average tat is 4.0
average wt is 1.0
```

**********************PRIORITY*****************************

```
import java.util.*;

class Process
{
int pid; // Process ID
int bt; // CPU Burst time required
int priority; // Priority of this process
Process(int pid, int bt, int priority)
{
this.pid = pid;
this.bt = bt;
this.priority = priority;
}
public int prior() {
return priority;
}
}


public class GFG
{

// Function to find the waiting time for all
// processes
public void findWaitingTime(Process proc[], int n,
int wt[])
{

// waiting time for first process is 0
```

```
wt[0] = 0;

// calculating waiting time
for (int i = 1; i < n ; i++ )
wt[i] = proc[i - 1].bt + wt[i - 1] ;
}

// Function to calculate turn around time
public void findTurnAroundTime( Process proc[], int n,
int wt[], int tat[])
{
// calculating turnaround time by adding
// bt[i] + wt[i]
for (int i = 0; i < n ; i++)
tat[i] = proc[i].bt + wt[i];
}

// Function to calculate average time
public void findavgTime(Process proc[], int n)
{
int wt[] = new int[n], tat[] = new int[n], total_wt = 0, total_tat = 0;

// Function to find waiting time of all processes
findWaitingTime(proc, n, wt);

// Function to find turn around time for all processes
findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all details
System.out.print("\nProcesses  Burst time   Waiting time   Turn around time\n");

// 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.print(" " + proc[i].pid + "\t\t" + proc[i].bt + "\t " + wt[i] + "\t\t " + tat[i] + "\n");
}

System.out.print("\nAverage waiting time = "
+(float)total_wt / (float)n);
System.out.print("\nAverage turn around time = "+(float)total_tat / (float)n);
}

public void priorityScheduling(Process proc[], int n)
{

// Sort processes by priority
Arrays.sort(proc, new Comparator<Process>() {
@Override
public int compare(Process a, Process b) {
return b.prior() - a.prior();
}
});
System.out.print("Order in which processes gets executed \n");
for (int i = 0 ; i < n; i++)
System.out.print(proc[i].pid + " ") ;

findavgTime(proc, n);
}

// Driver code
public static void main(String[] args)
{
GFG ob=new GFG();
int n = 3;
Process proc[] = new Process[n];
proc[0] = new Process(1, 10, 2);
proc[1] = new Process(2, 5, 0);
proc[2] = new Process(3, 8, 1);
ob.priorityScheduling(proc, n);
}
}
```

______________OUTPUT________________________

```
Order in which processes gets executed
1 3 2
Processes  Burst time  Waiting time  Turn around time
 1               10      0                10
 3               8       10               18
 2               5       18               23

Average waiting time = 9.33333
Average turn around time = 17
```

*******************ROUND ROBIN**************************

```
public class GFG
{
   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
```

```
    while(true)
    {
        boolean done = true;
        for (int i = 0 ; i < n; i++)
        {
            if (rem_bt[i] > 0)
            {
                done = false; // There is a pending process

                if (rem_bt[i] > quantum)
                {
                    t += quantum;
                   rem_bt[i] -= quantum;
                }
                else
                {
                    t = t + rem_bt[i];
                   wt[i] = t - bt[i];
                   rem_bt[i] = 0;
                }
            }
        }
        if (done == true)
          break;
    }
}

static void findTurnAroundTime(int processes[], int n,
                        int bt[], int wt[], int tat[])
{

    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}


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;

    findWaitingTime(processes, n, bt, wt, quantum);
   findTurnAroundTime(processes, n, bt, wt, tat);
   System.out.println("Processes " + " Burst time " +
                  " Waiting time " + " 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);
    }

    public static void main(String[] args)
    {

        int processes[] = { 1, 2, 3};
        int n = processes.length;

        int burst_time[] = {10, 5, 8};

        int quantum = 2;
        findavgTime(processes, n, burst_time, quantum);
    }
}
```

______________OUTPUT________________________

```
Processes Burst time  Waiting time  Turn around time
 1       10     13        23
 2        5     10        15
 3        8     13        21
Average waiting time = 12
Average turn around time = 19.6667
```