CLOUD COMPUTING LAB-02
NAME: DRISHTI
ROLL NO.: 1928228
DATE: 05/01/2022

Q. Write a program to implement Round Robin Scheduling, SJF(preemptive), SJF(non preemptive) and FCFS algorithm in C.

//Round Robin

```
#include<stdio.h>
struct times
{
int p,art,but,wtt,tat,rnt;
};


void sortart(struct times a[],int pro)
{
int i,j;
struct times temp;
for(i=0;i<pro;i++)
{
for(j=i+1;j<pro;j++)
{
if(a[i].art > a[j].art)
{
temp = a[i];
a[i] = a[j];
a[j] = temp;
}
}
}
return;
}

int main()
{
int i,j,pro,time,remain,flag=0,ts;
struct times a[100];
float avgwt=0,avgtt=0;
printf("Round Robin Scheduling Algorithm\n");
printf("Enter Number Of Processes : ");
scanf("%d",&pro);
remain=pro;
for(i=0;i<pro;i++)
{
printf("Enter arrival time and Burst time for Process P%d : ",i);
scanf("%d%d",&a[i].art,&a[i].but);
a[i].p = i;
a[i].rnt = a[i].but;
}
sortart(a,pro);
printf("Enter Time Slice OR Quantum Number : ");
scanf("%d",&ts);
printf("\n**************************************\n");
printf("Gantt Chart\n");
printf("0");
for(time=0,i=0;remain!=0;)
{
if(a[i].rnt<=ts && a[i].rnt>0)
{
time = time + a[i].rnt;
printf(" -> [P%d] <- %d",a[i].p,time);
a[i].rnt=0;
flag=1;
}
else if(a[i].rnt > 0)
{
a[i].rnt = a[i].rnt - ts;
time = time + ts;
printf(" -> [P%d] <- %d",a[i].p,time);
}
if(a[i].rnt==0 && flag==1)
{
remain--;
a[i].tat = time-a[i].art;
a[i].wtt = time-a[i].art-a[i].but;
avgwt = avgwt + time-a[i].art-a[i].but;
avgtt = avgtt + time-a[i].art;
flag=0;
}
if(i==pro-1)
i=0;
else if(a[i+1].art <= time)
i++;
else
i=0;
}
printf("\n\n");
printf("**************************************\n
");
printf("Pro\tAT\tBT\tTT\tWT\n");
printf("**************************************\n
");
for(i=0;i<pro;i++)
{
printf("P%d\t%d\t%d\t%d\t%d\n",a[i].p,a[i].art,a[i].but,a
[i].tat,a[i].wtt);
}
printf("**************************************\n
");
avgwt = avgwt/pro;
avgtt = avgtt/pro;
printf("Average Waiting Time : %.2f\n",avgwt);
printf("Average Turnaround Time : %.2f\n",avgtt);
return 0;
}
```

OUTPUT:

```
Round Robin Scheduling Algorithm
Enter Number Of Processes : 10
Enter arrival time and Burst time for Process P0 : 0
5
Enter arrival time and Burst time for Process P1 : 1
3
Enter arrival time and Burst time for Process P2 : 2
1
Enter arrival time and Burst time for Process P3 : 3
2
Enter arrival time and Burst time for Process P4 : 4
3
Enter arrival time and Burst time for Process P5 : 5
4
Enter arrival time and Burst time for Process P6 : 6
5
Enter arrival time and Burst time for Process P7 : 7
1
Enter arrival time and Burst time for Process P8 : 8
3
Enter arrival time and Burst time for Process P9 : 9
5
Enter Time Slice OR Quantum Number : 2
```

```
**************************************
Pro     AT      BT      TT      WT
**************************************
P0      0       5       30      25
P1      1       3       20      17
P2      2       1       3       2
P3      3       2       4       2
P4      4       3       18      15
P5      5       4       19      15
P6      6       5       25      20
P7      7       1       7       6
P8      8       3       19      16
P9      9       5       23      18
**************************************
Average Waiting Time : 13.60
Average Turnaround Time : 16.80
```

//Shortest Job First(PREEMEPTIVE) OR Shortest Remaining Time First

```
#include<stdio.h>
#include<stdbool.h>
typedef struct
{
int pid;
float at, wt, bt, ta, st;
bool isComplete;
}process;
void procdetail(int i, process p[])
{
printf("Process id: ");
scanf("%d", &p[i].pid);
printf("Arrival Time: ");
scanf("%f", &p[i].at);
printf("Burst Time: ");
scanf("%f", &p[i].bt);
p[i].isComplete = false;
}//procdetail
void sort(process p[], int i, int start)
{
int k = 0, j;
process temp;
for (k = start; k<i; k++){
for (j = k+1; j<i; j++){
if(p[k].bt < p[j].bt)
continue;
else{
temp = p[k];
p[k] = p[j];
p[j] = temp;}}}}
//sort
void main()
{ int n, i, k = 0, j = 0;
float avgwt = 0.0, avgta = 0.0, tst = 0.0;
printf("Enter number of processes: ");
scanf("%d",&n);
process p[n];
for (i = 0; i<n; i++)
{
printf("\nEnter process %d's details: ",i);
procdetail(i,p);
}
for (i = 0; i<n; i++)
{
if (p[i].isComplete == true)
continue;
else
{
k = i;
while (p[i].at<=tst && i<n)
i++;
sort (p,i,k);
i = k;
if(p[i].at<=tst)
p[i].st = tst;
else
p[i].st = p[i].at;
p[i].st = tst;
p[i].isComplete = true;
tst += p[i].bt;
p[i].wt = p[i].st - p[i].at;
p[i].ta = p[i].bt + p[i].wt;
avgwt += p[i].wt;
avgta += p[i].ta;
}
}
avgwt /= n;
avgta /= n;
printf("Process Schedule Table: \n");
printf("\tProcess ID\tArrival Time\tBurst Time\tWait
Time\tTurnaround Time\n");
for (i = 0; i<n; i++)
printf("\t%d\t\t%f\t%f\t%f\t%f\n", p[i].pid,p[i].at, p[i].bt,
p[i].wt, p[i].ta);
printf("\nAverage wait time: %f", avgwt);
printf("\nAverage turnaround time: %f\n", avgta);
}//main
```

OUTPUT:

```
Enter process 0's details: Process id: 1
Arrival Time: 0
Burst Time: 5

Enter process 1's details: Process id: 2
Arrival Time: 1
Burst Time: 3

Enter process 2's details: Process id: 3
Arrival Time: 2
Burst Time: 1

Enter process 3's details: Process id: 4
Arrival Time: 3
Burst Time: 2

Enter process 4's details: Process id: 5
Arrival Time: 4
Burst Time: 3

Enter process 5's details: Process id: 6
Arrival Time: 5
Burst Time: 4

Enter process 6's details: Process id: 7
Arrival Time: 6
Burst Time: 5

Enter process 7's details: Process id: 8
Arrival Time: 7
Burst Time: 1

Enter process 8's details: Process id: 9
Arrival Time: 8
Burst Time: 3

Enter process 9's details: Process id: 10
Arrival Time: 9
Burst Time: 5
```

```
Process Schedule Table:
        Process ID      Arrival Time    Burst Time      Wait Time       Turnaround Time
        1               0.000000        5.000000        0.000000        5.000000
        3               2.000000        1.000000        3.000000        4.000000
        4               3.000000        2.000000        3.000000        5.000000
        8               7.000000        1.000000        1.000000        2.000000
        2               1.000000        3.000000        8.000000        11.000000
        9               8.000000        3.000000        4.000000        7.000000
        5               4.000000        3.000000        11.000000       14.000000
        6               5.000000        4.000000        13.000000       17.000000
        10              9.000000        5.000000        13.000000       18.000000
        7               6.000000        5.000000        21.000000       26.000000

Average wait time: 7.700000
Average turnaround time: 10.900000
```

//First Come First Serve

```
#include<stdio.h>
void findWaitingTime(int processes[], int n,int bt[], int wt[])
{
wt[0] = 0;
for (int i = 1; i < n ; i++ )
wt[i] = bt[i-1] + wt[i-1] ;
}
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];
}
void findavgTime( int processes[], int n, int bt[])
{
int wt[n], tat[n], total_wt = 0, total_tat = 0;
findWaitingTime(processes, n, bt, wt);
findTurnAroundTime(processes, n, bt, wt, tat);
printf("For FCFS\n");
printf("Processes Burst time Waiting time Turn around time\n");
for (int i=0; i<n; i++)
{
total_wt = total_wt + wt[i];
total_tat = total_tat + tat[i];
printf(" %d ",(i+1));
printf("        %d ", bt[i] );
printf("       %d",wt[i] );
printf("       %d\n",tat[i] );
}
int s=(float)total_wt / (float)n;
int t=(float)total_tat / (float)n;
printf("Average waiting time = %d",s);
printf("\n");
printf("Average turn around time = %d ",t);
}
int main()
{
int processes[] = { 1, 2, 3,4,5,6,7,8,9,10};
int n = sizeof processes / sizeof processes[0];
int burst_time[] = {5,3,1,2,3,4,5,1,3,5};
findavgTime(processes, n, burst_time);
return 0;
}
```

OUTPUT

```
For FCFSProcesses Burst time Waiting time Turn around time
 1              5              0       5
 2              3              5       8
 3              1              8       9
 4              2              9       11
 5              3              11      14
 6              4              14      18
 7              5              18      23
 8              1              23      24
 9              3              24      27
 10             5              27      32
Average waiting time = 13
Average turn around time = 17
```

//Shortest Job First (NON PREEMPTIVE)

```
#include<stdio.h>

int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d:",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;
   }
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(bt[j]<bt[pos])
            pos=j;
      }

      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;

      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }

   wt[0]=0;
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];

      total+=wt[i];
   }

   avg_wt=(float)total/n;
   total=0;

   printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      total+=tat[i];
      printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
   }

   avg_tat=(float)total/n;
   printf("\n\nAverage Waiting Time=%f",avg_wt);
   printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

OUTPUT

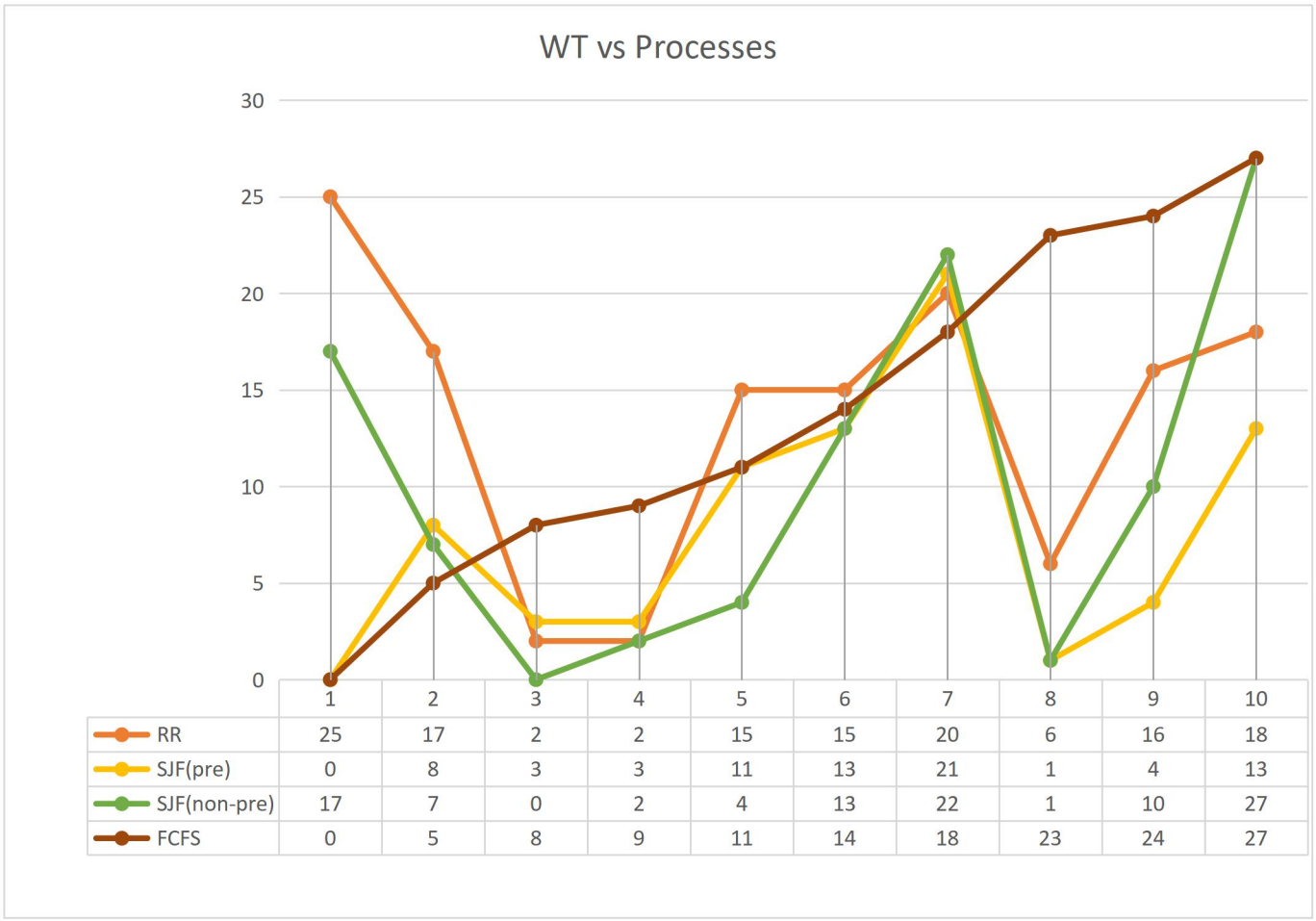
```
For SJF
Enter number of process:10

Enter Burst Time:
p1:5
p2:3
p3:1
p4:2
p5:3
p6:4
p7:5
p8:1
p9:3
p10:5
```

```
Process      Burst Time          Waiting Time     Turnaround Time
 p3                1                  0                1
 p8                1                  1                2
 p4                2                  2                4
 p5                3                  4                7
 p2                3                  7                10
 p9                3                  10               13
 p6                4                  13               17
 p1                5                  17               22
 p7                5                  22               27
 p10               5                  27               32

Average Waiting Time=10.300000
Average Turnaround Time=13.500000
```

Q. Compare all the scheduling algos' average waiting time(s) and turn around time(s) and waiting time at each process.



	1	2	3	4	5	6	7	8	9	10
RR	25	17	2	2	15	15	20	6	16	18
SJF(pre)	0	8	3	3	11	13	21	1	4	13
SJF(non-pre)	17	7	0	2	4	13	22	1	10	27
FCFS	0	5	8	9	11	14	18	23	24	27

	RR	SJF(pre)	SJF(non-pre)	FCFS
Avg WT	13.60	7.70	10.30	13.00
Avg TAT	16.80	10.90	13.50	17.00

Assumptions:

- The arrival time of all the processes in Round Robin Scheduling and Shorest Job First (Preemptive) is taken as [0,1,2,3,4,5,6,7,8,9,10], for First Come First Serve we have taken the processes in the [1,2,3,4,5,6,7,8,9,10] and for Shortest Job Job First(Non Preemptive) we have not considered the arriving time.
- We have considered 10 processes, with burst time [5,3,1,2,3,4,5,1,3,5] for each of the scheduling algorithms.
- For Round Robin we have taken Time slice to be 2.

Conclusion:

We can infer from the above graph that waiting time is in the following fashion-

SRTF(preemptive SJF) < (non preemptive) SJF < FCFS < RR

Shortest Job First- preemptive having the least waiting time and Round Robin having the highest

So, the order of efficiency goes the other way around-

SRTF(preemptive SJF) > (non preemptive) SJF > FCFS > RR
