Practical No.3

Name: Jadhav Siddhesh Ramesh
Class: T.Y.B.Sc(Computer Science)
Roll no.: 32
Div: A   Batch: B
Subject: Operating System-I
Name Of Practicale: CPU Scheduling.
Performance Date: / /2025   Submission Date: / /2025

Set A

Q.1) Write the program to simulate FCFS CPU-scheduling. The arrival time and first CPU- burst for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void FCFS1(int n,char pro[],int at[]){
int wt[10],tat[10];
float tavg,tatavg;
wt[0]=wtavg=0;
tat[0]=tatavg=bt[0];
for(i=1;i<n;i++)
{
        wt[i]=wt[i-1]+bt[i-1];
        tat[i]=tat[i-1]+bt[i];
        wtavg=wtavg+wt[i];
        tatavg=tatavg+tat[i];
}
printf("Gantt Chart:");
printf("0-");
for(i=0;i<n;i++)
{
        printf("%c-%d",pro[i],(tat[i]+at[i]));
}
printf("\npn\tat\tbt\tct\ttat\twt\n");
for(i=0;i<n;i++)
{
        printf("%c\t%d\t%d\t%d\t%d\t%d\n",pro[i],at[i],bt[i],tat[i]+at[i],wt[i]);
        printf("Avg TAT=%0.2f\tAvg WT=%0.2f\n",tatavg/n,wtavg/n);
}
int main(){
        char pro[10];
        int at[10];
        int bt[10];
        int i,n;
        printf("Enter   no.ofprocess:");
        scanf("%d",&n);
        for(i=0;i<n;i++);
        {
                fflush(stdin);
                printf("Enter process name:");
                scanf("%s",&pro[i]);
                printf("Enter  arrival time:");
                scanf("%d",&at[i]);
                printf("Enetr friest CPU burst time:");
                scanr("%d",&bt[i]);
}
                FCFS(n,pro,bt,at);
                return 0;
}
```

Output:

```
Enter number of process:4
enter process name P1
enter arrival time 0
enter first cpu burst time 2
enter process name P2
enter arrival time 1
enter first cpu burst time 3
enter process name P3
enter arrival time 3
enter first cpu burst time 4
enter process name P4
enter arrival time 4
enter first cpu burst time 5

gantt chart:0_1-2-2-6-3-12-4-18-
pn      at      bt      ct      tat     wt
P1      0       2       2       2       0
P2      1       3       6       5       2
P3      3       4       12      9       5
P4      4       5       18      14      9
Avg TAT=7.50            Avg Wt=4.00
```

Set B

Q1) Write the program to simulate Preemptive Shortest Job First (SJF) -scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

Program:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct process
{
  char pname[10];
  int AT,BT,ST,FT,TT,WT,BT1;
}p[15];
struct process t;
int i,n,j,k,bt,tq;
char GC[200];
void get_data()
{
  printf("Enter number of processes:");
  scanf("%d",&n);
  printf("Enter process details for %d processes",n);
```

```
 for(i=0;i<n;i++)
 {
  printf("\nEnetr Process name,arival time,cpu burst time:");
  scanf("%s %d %d",&p[i].pname,&p[i].AT,&p[i].BT);
  p[i].BT1=p[i].BT;
 }
}
void put_data()
{
 printf("\nProcesses are as below");
 printf("\nProcess name\t arival time\t cpu burst time");
 for(i=0;i<n;i++)
 {
  printf("\n%s\t\t%d\t\t%d",p[i].pname,p[i].AT,p[i].BT);
 }
}
void arrivalsort()
{
 for(i=0;i<n;i++)
 {
  for(j=i+1;j<n;j++)
  {
   if(p[i].AT>p[j].AT)
   {
    t=p[i];
    p[i]=p[j];
    p[j]=t;
   }
  }
 }
}
void burst_sort()
{
  for(i=0;i<n;i++)
 {
  for(j=i+1;j<n;j++)
  {
   if(p[i].AT>p[j].AT)
   {
    t=p[i];
    p[i]=p[j];
    p[j]=t;
   }
  }
 }
}
void avgTTWT()
{
 float sumtt=0,sumwt=0;
 for(i=0;i<n;i++)
 {
  p[i].TT=p[i].FT-p[i].AT;
  p[i].WT=p[i].TT-p[i].AT;
  sumtt=sumtt+p[i].TT;
  sumwt=sumwt+p[i].WT;
 }
```

```
printf("\nProcess\tAT\tBT\tTT\tWT\n");
for(i=0;i<n;i++)
{
 printf("\n%s\t\t%d\t%d\t%d\t%d",p[i].pname,p[i].AT,p[i].BT1,p[i].TT,p[i].WT);
}
printf("\nAverage turn around time=%f/%d=%f",sumtt,n,sumtt/n);
printf("\nAverage wait time=%f/%d=%f",sumwt,n,sumwt/n);
}
void pre_sjf()
{
 char str[5];
 i=0;
 int time=0;
 tq=1;
        strcpy(GC,"0|");
        aaa:
        if(p[i].BT!=0)
        {
            if(p[i].AT>time)
            {
              for(j=i+1;j<n;j++)
              {
                if(p[j].AT<p[i].AT&&p[j].BT!=0)
                {
                  time=p[j].AT;
                  sprintf(str,"%d",time);
                  strcat(GC,str);
                  strcat(GC,"|");
                  p[j].ST=time;
                                 strcat(GC,p[j].pname);
                                 p[j].BT=p[j].BT-tq;
                                 strcat(GC," ");
                                 time=time+tq;
                                        sprintf(str,"%d",time);
                                        strcat(GC,str);
                                        p[j].FT=time;
                }
              }
            }
            /*else
            {
              strcat(GC,"CPUIDLE");
              time=p[i].AT;
              sprintf(str,"%d",time);
              strcat(GC,str);
              strcat(GC,"|");
            }*/
            p[i].ST=time;
            strcat(GC,p[i].pname);
            time=time+tq;
            strcat(GC," ");
            p[i].FT=time;
            sprintf(str,"%d",time);
            strcat(GC,str);
            strcat(GC,"|");
            p[i].BT=p[i].BT-tq;
```

```
          burst_sort();
        }
        for(i=0;i<n;i++)
        {
          if(p[i].BT!=0)
             goto aaa;
        }
          printf("\nGantt Chart\n");
          puts(GC);
          avgTTWT();
}
int main()
{
  get_data();
  arrivalsort();
  put_data();
  pre_sjf();
}
```

Output:

```
Enter number of processes:4
Enter process details for 4 processes
Enter Process name,arival time,cpu burst time:P1 0 5
Enter Process name,arival time,cpu burst time:P2 2 4
Enter Process name,arival time,cpu burst time:P3 1 6
Enter Process name,arival time,cpu burst time:P4 3 2
Processes are as below
Process name  arival time      cpu burst time
P1            0                5
P3            1                6
P2            2                4
P4            3                2
Gantt Chart
0|P1 1|P1 2|P1 3|P1 4|P1 5|P3 6|P3 7|P3 8|P3 9|P3 10|P3 11|P2 12|P2 13|P2 14|P2 15|P4 16|P4 17|
Process       AT     BT     TT     WT

P1            0      5      5      5
P3            1      6      10     9
P2            2      4      13     11
P4            3      2      14     11
Average turn around time=42.000000/4=10.500000
Average wait time=36.000000/4=9.000000s
```

Q.2) Write the program to simulate Non-preemptive Priority scheduling. The arrival time and first CPU-burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

Program:

```
#include <stdio.h>
#include <limits.h>

struct process {
   char id;
   int arrival;
```

```
    int burst;
    int wait;
    int turn;
    int finish;
    int original_burst; // Store original burst time
};

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct process p[n];
    int i, j;

    for (i = 0; i < n; i++) {
        printf("Enter arrival time and burst time for process %c: ", 'A' + i);
        scanf("%d %d", &p[i].arrival, &p[i].burst);
        p[i].id = 'A' + i;
        p[i].original_burst = p[i].burst; // Store the original burst time
    }

    // Sort by arrival time
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            if (p[i].arrival > p[j].arrival) {
                struct process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }

    int time = 0;
    int completed = 0;
    float total_wait = 0, total_turn = 0;

    printf("\n\nGantt Chart:\n");
    int last_time = 0;

    while (completed < n) {
        int shortest = -1;
        int min_burst = INT_MAX; // Use INT_MAX from <limits.h> for better practice

        for (i = 0; i < n; i++) {
            if (p[i].arrival <= time && p[i].burst > 0 && p[i].burst < min_burst) {
                min_burst = p[i].burst;
                shortest = i;
            }
        }

        if (shortest == -1) {
            time++;
            continue;
        }
        printf("  %d  |  %c  ", time, p[shortest].id);

        time += p[shortest].burst;
        p[shortest].finish = time;
        p[shortest].turn = p[shortest].finish - p[shortest].arrival;
        p[shortest].wait = p[shortest].turn - p[shortest].original_burst;
        p[shortest].burst = 0; // Mark as completed

        completed++;
        total_wait += p[shortest].wait;
        total_turn += p[shortest].turn;
    }
    printf("  %d\n", time);

    printf("\n\nProcess\tArrival\tBurst\tWait\tTurnaround\n");
    for (i = 0; i < n; i++) {
        printf("%c\t%d\t%d\t%d\t%d\n", p[i].id, p[i].arrival, p[i].original_burst, p[i].wait, p[i].turn);
    }

    printf("\nAverage Wait Time: %.2f\n", total_wait / n);
    printf("Average Turnaround Time: %.2f\n", total_turn / n);

    return 0;
}
```

Output:

```
Enter number of processes: 4
Enter arrival time and burst time for process A: 0 5
Enter arrival time and burst time for process B: 1 4
Enter arrival time and burst time for process C: 2 6
Enter arrival time and burst time for process D: 4 3
Gantt Chart:
 0  |  A    5  |  D    8  |  B    12  |  C    18
Process         ArrivalBurst  Wait    Turnaround
A       0       5       0       5
B       1       4       7       11
C       2       6       10      16
D       4       3       1       4

Average Wait Time: 4.50
Average Turnaround Time: 9.00
```

Set C

Q.1) Write the program to simulate Preemptive Priority scheduling. The arrival time and first CPU-burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

Program:

```
#include <stdio.h>
struct process {
char id;
int arrival;
int burst;
int priority;
int wait;
int turn;
int finish;
};
int main() {
int n;
printf("Enter number of processes: ");
scanf("%d", &n);
struct process p[n];
int i, j;
for (i = 0; i < n; i++) {
printf("Enter arrival time, burst time, and priority for process %c: ", 'A' + i);
scanf("%d %d %d", &p[i].arrival, &p[i].burst, &p[i].priority);
p[i].id = 'A' + i;
}int time = 0;
int completed = 0;
float total_wait = 0, total_turn = 0;
printf("\nGantt Chart:\n");
while (completed < n) {
int highest_priority = -1;
int min_priority = 9999;
for (i = 0; i < n; i++) {
if (p[i].arrival <= time && p[i].burst > 0 && p[i].priority < min_priority) {
min_priority = p[i].priority;
highest_priority = i;
}
}
if (highest_priority == -1) {
time++;
continue;
}
printf("%d %c ", time, p[highest_priority].id);
time++;
p[highest_priority].burst--;
if (p[highest_priority].burst == 0) {p[highest_priority].finish = time;
p[highest_priority].turn = p[highest_priority].finish - p[highest_priority].arrival;
p[highest_priority].wait = p[highest_priority].turn - p[highest_priority].burst;
completed++;
total_wait += p[highest_priority].wait;
total_turn += p[highest_priority].turn;
}
}
printf("\n\nProcess\tArrival\tBurst\tPriority\tWait\tTurn\n");
for (i = 0; i < n; i++) {
printf("%c\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].arrival, p[i].burst + p[i].burst - p[i].burst,
p[i].priority, p[i].wait, p[i].turn);
}
printf("Average Wait: %.2f\n", total_wait / n);
printf("Average Turnaround: %.2f\n", total_turn / n);
return 0;
}
```

Output:

```
Enter number of processes: 4
Enter arrival time, burst time, and priority for process A: 0 1 2
Enter arrival time, burst time, and priority for process B: 2 5 4
Enter arrival time, burst time, and priority for process C: 3 2 3
Enter arrival time, burst time, and priority for process D: 1 6 1

Gantt Chart:
0 A 1 D 2 D 3 D 4 D 5 D 6 D 7 C 8 C 9 B 10 B 11 B 12 B 13 B

Process         ArrivalBurst    Priority        Wait    Turn
A       0       0       2       1       1
B       2       0       4       12      12
C       3       0       3       6       6
D       1       0       1       6       6
Average Wait: 6.25
Average Turnaround: 6.25
```