Lab # 08

AIM:

To write a C program for implementation of Priority scheduling algorithms.

ALGORITHM:

1. Input n (number of processes), burst times (bt[i]), and priorities (pri[i]).

2. Sort processes by pri[i] in ascending order.

3. Set waiting_time[0] = 0.

4. Loop from i = 1 to n:

a. waiting_time[i] = waiting_time[i-1] + bt[i-1].

b. turnaround_time[i] = waiting_time[i] + bt[i].

5. Calculate total waiting and turnaround times.

6. Print waiting and turnaround times for each process and averages.

PROGRAM:

```
#include<stdio.h>
 #include<stdio.h>
 #include<stdlib.h>
typedef struct
{
int pno;
 int pri;
int pri;
 int btime;
```

```
int wtime;
}sp;
int main()
{
int i,j,n;
int tbm=0,totwtime=0,totttime=0; sp *p,t;
printf("\n PRIORITY SCHEDULING.\n");
printf("\n enter the no of process.        \n");
scanf("%d",&n); p=(sp*)malloc(sizeof(sp));
printf("enter the burst time and priority:\n"); for(i=0;i<n;i++)
{
printf("process%d:",i+1); scanf("%d%d",&p[i].btime,&p[i].pri);
 p[i].pno=i+1; }
for(i=0;i<n-1;i++) for(j=i+1;j<n;j++)
{
if(p[i].pri>p[j].pri)
{
t=p[i];
p[i]=p[j];
 p[j]=t;
}
```

```
}
printf("\n process\tbursttime\twaiting time\tturnaround time\n"); for(i=0;i<n;i++)
{
totwtime+=p[i].wtime=tbm; tbm+=p[i].btime;
printf("\n%d\t\t%d",p[i].pno,p[i].btime);
printf("\t\t%d\t\t%d",p[i].wtime,p[i].wtime+p[i].btime);
}
totttime=tbm+totwtime;
printf("\n total waiting time:%d",totwtime);
printf("\n average waiting time:%f",(float)totwtime/n); printf("\n total turnaround time:%d",totttime); printf("\n avg turnaround time:%f",(float)totttime/n);
}
```

Lab #9

AIM:

To write a C program for implementation of Round Robin scheduling algorithm

ALGORITHM:

1. Input n (number of processes), burst times (bt[i]), and time quantum (tq).
2. Initialize remaining_time[i] = bt[i] for all i.
3. Initialize time = 0.
4. While all processes are not completed:

a. For each process i: i.

If remaining_time[i] > tq:

- time += tq

- remaining_time[i] -= tq. ii.

Else:

- time += remaining_time[i]

- remaining_time[i] = 0.

- waiting_time[i] = time - bt[i].

5. Calculate turnaround_time[i] = waiting_time[i] + bt[i].

6. Print waiting and turnaround times for each process and averages.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct rr
{
int pno,btime,sbtime,wtime,lst;
}
p[10];
int main()
{
int pp=-1,ts,flag,count,ptm=0,i,n,twt=0,totttime=0;
printf("\n round robin scheduling         ");
```

```
printf("enter no of processes:");
scanf("%d",&n);
printf("enter the time slice:");
scanf("%d",&ts); printf("enter the burst time");
for(i=0;i<n;i++)
{
printf("\n process%d\t",i+1);
 scanf("%d",&p[i].btime);
p[i].wtime=p[i].lst=0;
p[i].pno=i+1;
 p[i].sbtime=p[i].btime;
 }
printf("scheduling  \n");
do
{
flag=0; for(i=0;i<n;i++)
{
count=p[i].btime; if(count>0)
{
flag=-1; count=(count>=ts)?ts:count; printf("\n process %d",p[i].pno);
printf("from%d",ptm); ptm+=count; printf("to%d",ptm);
p[i].btime-=count;
if(pp!=i)
```

```
{
pp=i;
p[i].wtime+=ptm-p[i].lst-count; p[i].lst=ptm;
}
}
```

Lab no #10

AIM:

To write a C program for implementation of FCFS and SJF scheduling algorithms.

ALGORITHM:

1. Input n (number of processes) and burst times (bt[i]).
2. Set waiting_time[0] = 0.
3. Loop from i = 1 to n:

a. waiting_time[i] = waiting_time[i-1] + bt[i-1].

b. turnaround_time[i] = waiting_time[i] + bt[i].

4. Calculate total waiting and turnaround times.
5. Print waiting and turnaround times for each process and averages.

Step 7: Stop the program

PROGRAM:

```
#include<stdio.h>
 #include<stdlib.h>
 struct fcfs
{
int pid;

int btime;
 int wtime;
 int ttime;
}
```

```
 p[10];
int main()
{
int i,n;
int towtwtime=0,totttime=0;
 printf("\n fcfs scheduling...\n");
 printf("enter the no of process");
scanf("%d",&n); for(i=0;i<n;i++)
{
p[i].pid=1;
printf("\n burst time of the process”);
 scanf("%d",&p[i].btime);
 }
p[0].wtime=0;
 p[0].ttime=p[0].btime; totttime+=p[i].ttime; for(i=0;i<n;i++)
{
p[i].wtime=p[i-1].wtime+p[i-1].btim
p[i].ttime=p[i].wtime+p[i].btime; totttime+=p[i].ttime; towtwtime+=p[i].wtime;
}
for(i=0;i<n;i++)
{
{
```

```
printf("\n waiting time for process”);
 printf("\n turn around time for process”);
 printf("\n");
}}
printf("\n total waiting time :%d", totwtime );
printf("\n average waiting time :%f",(float)totwtime/n);
printf("\n total turn around time :%d",totttime);
printf("\n average turn around time: :%f",(float)totttime/n);
}
```

Lab #11

AIM:

To write a C program for implementation of SJF scheduling algorithms.

ALGORITHM:

1. Input n (number of processes) and burst times (bt[i]).

2. Sort processes based on bt[i] in ascending order.

3. Set waiting_time[0] = 0.

4. Loop from i = 1 to n:

a. waiting_time[i] = waiting_time[i-1] + bt[i-1].

b. turnaround_time[i] = waiting_time[i] + bt[i].

5. Calculate total waiting and turnaround times.

6. Print waiting and turnaround times for each process and averages.

Step 7: Stop the program.

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h> typedef struct
{
int pid; int btime; int wtime;
}
sp;
int main()
```

```
{
int i,j,n,tbm=0,towtwtime=0,totttime sp*p,t;
printf("\n sjf schaduling ..\n");
 printf("enter the no of processor");
scanf("%d",&n); p=(sp*)malloc(sizeof(sp));
 printf("\n enter the burst time");
 for(i=0;i<n;i++)
{
printf("\n process %d\t",i+1);
scanf("%d",&p[i].btime); p[i].pid=i+1;
 p[i].wtime=0;
}
for(i=0;i<n;i++) for(j=j+1,j<n;j++)
{
if(p[i].btime>p[j].btime)
{
t=p[i]; p[i]=p[j]; p[j]=t;
}}
printf("\n process scheduling\n");
printf("\n process \tburst time \t w for(i=0;i<n;i++)
{
towtwtime+=p[i].wtime=tbm; tbm+=p[i].btime;
 printf("\n%d\t\t%d",p[i].pid,p[i].bt );
printf("\t\t%d\t\t%d",p[i].wtime,p[i]);
}
```

```
totttime=tbm+towtwtime;
printf("\n total waiting time :%d", totwtime );
printf("\n average waiting time :%f",(float)totwtime/n);
 printf("\n total turn around time :%d",totttime);
printf("\n average turn around time: :%f",(float)totttime/n);
}
```