Lab-2.

To be familiar with process scheduling algorithm.

Theory.

Process scheduling algorithms are used by the Operating System to schedule the processes on the processor in an efficient way.

Purpose of a Scheduling algorithm:

- Maximum CPU utilization
- Fair allocation of CPU
- Maximum throughput
- Minimum turnaround time
- Minimum waiting time
- Minimum response time.

These are the following algorithms which can be used to schedule the jobs.

i) First Come First Serve:

It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-preemptive type of scheduling.

ii) Round Robin:

In the Round Robin scheduling algorithm, the OS define a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a preemptive type of scheduling.

iii) Shortest Job First:

The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-preemptive type of scheduling.

iv) Shortest remaining time first:

It is the preemptive form of SJF. In this algorithm, the OS schedules the job according to the remaining time of the execution.

v) Priority based scheduling

In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two process is same then they will be scheduled according to the arrival time.

vi) Highest Response Ratio Next:

In this algorithm, the process with the highest response ratio will be scheduled next. This reduces starvation in the system.

Program 1

Source code (fcfs.c)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
struct process
{
      int pid;
      int bt;
      int wt,tt;
}p[10];
int main()
{
    int i,n,totwt, tottt, avg1,avg2
    printf("Enter the no. of process\n");
    scanf("%d",&n);
    for(i=1; i<=n; i++)
    {
        p[i].pid= i;
        printf("Enter the no. of burst time");
        scanf("%d",&p[i].bt);
    }
    p[i].wt=0;
    p[1].tt = p[1].bt + p[1].wt;
    i=2;
    while(i<=n)
    {
        p[i].wt = p[i-1].bt +p[i-1].wt;
        p[i].tt = p[i].bt +p[i].wt;
        i++;
    }
    i=1;
    towt=tottt=0;
    printf("\n processid\t bt\t wt\t tt\n");
    while(i<=n)
    {
        printf("\n %d\t%d\t%d\t%d",p[i].pid,p[i].bt,
        p[i].wt, p[i].tt);
        towt= p[i].wt + totwt;
        tottt= p[i].tt + tottt;
        i++;
    }
    avg1 = totwt/n;
    avg2 = tottt/n;
    printf("\n avg=%d\t avg2=%d\t",avg1,avg2);
    return 0;
```

Input and Output:

Input:

```
gcc -c fcfcs.c -o fcfs.o
gcc fcfs.o -o fcfs
./fcfs.
```

Output:

```
Enter the no. of process
5
Enter the burst time 10
Enter the burst time 1
Enter the burst time 2
Enter the burst time 1
Enter the burst time 2
```

Processid	bt	wt	tt
1	10	0	10
2	1	10	11
3	2	11	13
4	1	13	14
5	2	14	16

```
avg1 = 9.600000
avg2 = 12.800000
```

Discussion:

In this program, we used first come first serve process scheduling. In this algorithm, the process are scheduled on the basis of their arrival time. The waiting time and turnaround time was calculated and their averages were also displayed.

Program 2:

Source code (sjn.c)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
struct process
{ int pid, bt,wt, tt;
}p[10], temp;
int main()
{
    int i,j,n,totwt, tottt;
    float avg1, avg2;
    printf("\n Enter the no.of process");
    scanf("%d",&n);
```

```
for (i=1; i<=n; i++)
{
    p[i].pid = i;
    printf("\n Enter the burst time:\t");
    scanf("%d", &p[i].bt);
}
for(i=1; i<n; i++)
{
    for(j=i+1; j<=n; j++)
    {
        if (p[i].bt > p[j].bt)
        {
            temp.pid = p[i].pid;
            p[i].pid = p[j].pid;
            p[j].pid = temp.pid;
            temp.bt = p[i].bt;
            p[i].bt = p[j].bt;
            p[j].bt = temp.bt;
        }
    }
}
p[1].wt = 0;
p[1].tt = p[1].bt + p[1].wt;
i = 2;
while (i<=n)
{
    p[i].wt = p[i-1].bt + p[i-1].wt;
    p[i].tt = p[i].bt + p[i].wt;
    i++;
}
i=1;
totwt = tottt = 0;
printf("\n Process id \t bt \t wt \t tt");
while (i<=n)
{
    printf("\n\t %d \t %d \t %d \t %d\n", p[i].pid, p[i].bt,
    p[i].wt, p[i].tt);
    totwt = p[i].wt + totwt;
    tottt = p[i].tt + tottt;
    i++;
}
```

```
    avg1 = (float) totwt /n;
    avg2 = (float) tottt/n;
    Printf("\n AVG1 = %f \t AVG2 = %f ", avg1, avg2);
}   return 0;
```

Input and Output:

Input:

```
gcc -c sjn.c -o sjn.o
gcc sjn.o -o sjn
./sjn
```

Output:

```
Enter the no. of process 5
Enter the burst time 10
Enter the burst time 1
Enter the burst time 2
Enter the burst time 1
Enter the burst time 2
```

Process id	bt	wt	tt
2	1	0	1
4	1	1	2
3	2	2	4
5	2	4	6
1	10	6	10

```
avg1 = 2.600000
avg2 = 5.800000
```

Discussion:

In this program, we used shortest job first algorithm. In this algorithm, the processes are scheduled on the basis of the burst time.

Program 3

Source code: (rr.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
Struct process
{
    int pid, bt, tt, wt;
};
int main( )
{
    Struct process x[10], p[30];
    int i, j, k, tot=0, m, n;
    float wttime=0.0, tottime=0.0, a1, a2;
    printf("\n Enter the number of process: \t");
    scanf("%d", &n);
    for( i=1; i<=n; i++)
    {
        x[i].pid = i;
        printf("\n Enter the Burst time: \t");
        scanf("%d", &x[i].bt);
        tot = tot + x[i].bt;
    }
    printf("\n Total Burst Time: \t %d", tot);
    p[0].tt=0;
    k=1;
    printf("\n Enter the Time Slice: \t");
    scanf("%d", &m);
    for(j=1; j<=tot; j++)
    {
        for(i=1; i<=n; i++)
        {
            if (x[i].bt!=0)
            {
                p[k].pid=i;
                if (x[i].bt-m<0)
                {
                    p[k].wt = p[k-1].tt;
                    p[k].bt = x[i].bt;
                    p[k].tt = p[k].wt + x[i].bt
                    x[i].bt = 0
                    k++;
```



```
                }
                else
                {
                    p[k].wt = p[k-1].tt;
                    p[k].tt = p[k].wt + m;
                    x[i].bt = x[i].bt - m;
                    k++;
                }
            }
        }
    }
    printf("\n Process id \t wt \t tt");
    for( i=1; i<k; i++)
    {
        printf("\n \t %d \t %d \t %d", p[i].pid, p[i].wt, p[i].tt);
        wttime = wttime + p[i].wt;
        tottime = tottime + p[i].tt;
        a1 = wttime/n;
        a2 = tottime/n;
    }
    printf("\n\n Average Waiting Time: \t %f", a1);
    printf("\n\n Average Turn Around Time: \t %f", a2);
    return 0;
}
```

Input and Output:

Input:

```
gcc -o rr.c -o rr.o
gcc rr.o -o rr
./rr
```

Output:

```
Enter the no. of process : 5
Enter the burst time 10
Enter the burst time 1
Enter the burst time 2
Enter the burst time 1
Enter the burst time 2
```

Total Burst Time: 16
Enter the Time Slice 2

Process id	wt	tt
1	0	2
2	2	3
3	3	5
4	5	6
5	6	8
1	8	10
1	10	12
1	12	14
1	14	16

Average Waiting Time: 12.000000
Average Turnaround Time: 15.200000

Discussion:

In this process, we used round robin algorithm for process scheduling. The time slice was defined 2 units.

Conclusion:

Hence, in this lab we learned about various process scheduling algorithm.