Meerut Institute of Technology, Meerut

Department of Computer Science & Engineering

LAB MANUAL

Lab Name: Operating System
Lab Code: NCS-451

Jyotirmay Patel

Hardware/Software Requirements

Hardware:

1. 700 MHz processor (about Intel Celeron or better)
2. 512 MB RAM (system memory)
3. 5 GB of hard-drive space (or USB stick, memory card or external drive but see LiveCD for an alternative approach)
4. VGA capable of 1024x768 screen resolution
5. Either a CD/DVD drive or a USB port for the installer media

Software:

Ubuntu Desktop OS latest release/ Windows XP/7, Turbo C++ Compiler Latest Version

PROGRAM 01

STATEMENT OF THE PROBLEM: TO WRITE A C PROGRAM FOR IMPLEMENTING ‘ FIRST COME FIRST SERVE ’ ALGORITHM.

Concept: Perhaps, First-Come-First-Served algorithm is the simplest scheduling algorithm is the simplest scheduling algorithm. Processes are dispatched according to their arrival time on the ready queue. Being a nonpreemptive discipline, once a process has a CPU, it runs to completion. The FCFS scheduling is fair in the formal sense or human sense of fairness but it is unfair in the sense that long jobs make short jobs wait and unimportant jobs make important jobs wait.

FCFS is more predictable than most of other schemes since it offers time. FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time. The code for FCFS scheduling is simple to write and understand. One of the major drawback of this scheme is that the average time is often quite long.

The First-Come-First-Served algorithm is rarely used as a master scheme in modern operating systems but it is often embedded within other schemes.

Source code:

```
#include<stdio.h>

#include<conio.h>


void main()

{

        int n,i,j,sum=0;

        int arrv[10], ser[10], start[10], finish[10],wait[10], turn[10];

        float avgturn=0.0,avgwait=0.0;

        start[0]=0;

        clrscr();

        printf("\n ENTER THE NO. OF PROCESSES:");

        scanf("%d",&n);
```

```
for(i=0;i<n;i++)
{
        printf("\n ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS %d: ",i+1);
        scanf("%d%d",&arrv[i],&ser[i]);
}
for(i=0;i<n;i++)
{
        sum=0;
        for(j=0;j<i;j++)
                sum=sum+ser[j];
        start[i]=sum;
}
for(i=0;i<n;i++)
{
        finish[i]=ser[i]+start[i];
        wait[i]=start[i];
        turn[i]=ser[i]+wait[i];
}

for(i=0;i<n;i++)
{
        avgwait+=wait[i] ;
        avgturn+=turn[i];
}
avgwait/=n;
avgturn/=n;
```

```
        printf("\n PROCESS   ARRIVAL   SERVICE   START   FINISH   WAIT              TURN \n");
        for(i=0;i<n;i++)
        {
                printf("\n\tP%d\t%d \t %d \t %d \t %d \t %d \t %d                              \n",i ,arrv[i],
ser[i], start[i], finish[i],wait[i],turn[i]);
        }
        printf("\n AVERAGE WAITING TIME = %f  tu",avgwait);
        printf("\n AVERAGE TURN AROUND TIME = %f  tu",
        avgturn);
        getch();
}
```

FCFS CPU Scheduling Output:

```
 ENTER THE NO. OF PROCESSES:3

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 1: 0
3

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 2: 1
3

 ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 3: 2
2

 PROCESS   ARRIVAL  SERVICE  START  FINISH  WAIT        TURN

     P0    0      3     0     3     0     3

     P1    1      3     3     6     3     6

     P2    2      2     6     8     6     8

 AVERAGE WAITING TIME = 3.000000  tu
 AVERAGE TURN AROUND TIME = 5.666667  tu
```

```
ENTER THE NO. OF PROCESSES:4


ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 1: 1

2

ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 2: 3

2

ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 3: 1

5

ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 4: 4

6

PROCESS   ARRIVAL   SERVICE   START   FINISH   WAIT        TURN


     P0    1      2     0     2     0     2



     P1    3      2     2     4     2     4



     P2    1      5     4     9     4     9



     P3    4      6     9     15     9     15



AVERAGE WAITING TIME = 3.750000  tu

AVERAGE TURN AROUND TIME = 7.500000  tu
```

```
ENTER THE NO. OF PROCESSES:5


ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 1: 1

2

ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 2: 3

5

ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 3: 2

6

ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 4: 3

6

ENTER THE ARRIVAL TIME AND SERVICE TIME OF PROCESS 5: 5

3

PROCESS   ARRIVAL   SERVICE   START   FINISH   WAIT         TURN


    P0    1      2    0     2     0     2


    P1    3      5    2     7     2     7


    P2    2      6    7     13    7     13


    P3    3      6    13    19    13    19


    P4    5      3    19    22    19    22


AVERAGE WAITING TIME = 8.200000  tu

AVERAGE TURN AROUND TIME = 12.600000  tu
```

PROGRAM 02

STATEMENT OF THE PROBLEM: TO WRITE A C PROGRAM FOR IMPLEMENTING ' SHORTEST JOB FIRST ' ALGORITHM.

Concept: Shortest-Job-First (SJF) is a non-preemptive discipline in which waiting job (or process) with the smallest estimated run-time-to-completion is run next. In other words, when CPU is available, it is assigned to the process that has smallest next CPU burst. The SJF scheduling is especially appropriate for batch jobs for which the run times are known in advance. Since the SJF scheduling algorithm gives the minimum average time for a given set of processes, it is probably optimal.

The SJF algorithm favors short jobs (or processors) at the expense of longer ones. The obvious problem with SJF scheme is that it requires precise knowledge of how long a job or process will run, and this information is not usually available. The best SJF algorithm can do is to rely on user estimates of run times.

Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int n,i,j,temp1,temp2,sum=0;
        int pro[10],arrv[10],ser[10],start[10],finish[10],wait[10],
                                                        turn[10];
        float avgturn=0.0,avgwait=0.0;
        start[0]=0;
        clrscr();
        printf("\n ENTER THE NO. OF PROCESSES:");

        scanf("%d",&n);


        for(i=0;i<n;i++)

        {

                printf("\n ENTER THE ARRIVAL TIME AND SERVICE TIME
OF  PROCESS %d:",i+1);

                scanf("%d%d",&arrv[i],&ser[i]);

                pro[i]=i;

        }


        for(i=0;i<n;i++)

        {

                for(j=0;j<n-i-1;j++)

                {


                        if(ser[j]>ser[j+1])

                        {

                                temp1=ser[j];

                                ser[j]=ser[j+1];

                                ser[j+1]=temp1;
                                temp2=arrv[j];

                                arrv[j]=arrv[j+1];

                                arrv[j+1]=temp2;

                        }

                }

        }


        for(i=0;i<n;i++)

        {

                sum=0;

                for(j=0;j<i;j++)

                        sum=sum+ser[j];

                start[i]=sum;

        }



        for(i=0;i<n;i++)

        {

                finish[i]=ser[i]+start[i];

                wait[i]=start[i];

        turn[i]=ser[i]+wait[i];
```

```
        }



        for(i=0;i<n;i++)
        {
                avgwait +=wait[i] ;
                avgturn +=turn[i];
        }
        avgwait/=n;
        avgturn/=n;
        printf("\n PROCESS   ARRIVAL   SERVICE   START   FINISH  WAIT
TURN \n");


        for(i=0;i<n;i++)
        {
                printf("\n\tP%d\t%d \t %d \t %d \t %d \t %d \t %d \n",
pro[i],arrv[i], ser[i], start[i], finish[i],wait[i],turn[i]);
        }
        printf("\n AVERAGE WAITING TIME = %f  tu",avgwait);
        printf("\n AVERAGE TURN AROUND TIME = %f tu" ,avgturn) ;
        getch();
}
```

SJF CPU Scheduling Output:

```
 ENTER THE NO. OF PROCESSES:3


 ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 1:5

9


 ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 2:2

4


 ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 3:5

6


 PROCESS    ARRIVAL   SERVICE   START   FINISH  WAIT TURN


      P0     2       4     0      4      0      4


      P1     5       6     4      10     4      10


      P2     5       9     10     19     10     19


 AVERAGE WAITING TIME = 4.666667  tu

 AVERAGE TURN AROUND TIME = 11.000000 tu
```

```
ENTER THE NO. OF PROCESSES:4


ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 1:2

3


ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 2:9

1


ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 3:3

3


ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 4:5

2


PROCESS   ARRIVAL  SERVICE  START  FINISH  WAIT TURN


   P0   9    1    0    1    0    1


   P1   5    2    1    3    1    3


   P2   2    3    3    6    3    6


   P3   3    3    6    9    6    9


AVERAGE WAITING TIME = 2.500000  tu

AVERAGE TURN AROUND TIME = 4.750000 tu
```

```
ENTER THE NO. OF PROCESSES:5


ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 1:1

5

ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 2:2

3

ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 3:4

5

ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 4:6

1

ENTER THE ARRIVAL TIME AND SERVICE TIME OF  PROCESS 5:5

2

PROCESS   ARRIVAL   SERVICE   START   FINISH  WAIT TURN

    P0    6     1     0     1     0     1


    P1    5     2     1     3     1     3


    P2    2     3     3     6     3     6


    P3    1     5     6     11    6     11


    P4    4     5     11    16    11    16


AVERAGE WAITING TIME = 4.200000  tu

AVERAGE TURN AROUND TIME = 7.400000 tu
```

PROGRAM 03

STATEMENT OF THE PROBLEM: TO WRITE A C PROGRAM FOR IMPLEMENTING ‘ PRIORITY SCHEDULING ’ ALGORITHM.

Concept: Each process is assigned a priority, and priority is allowed to run. Equal-Priority processes are scheduled in FCFS order. The shortest-Job-First (SJF) algorithm is a special case of general priority scheduling algorithm.

Source code :

```
#include<stdio.h>

#include<conio.h>

void main()

{

        int n,i,j,temp1,temp2,sum=0;

        int pro[10],ser[10],start[10],finish[10],wait[10],prior[10],          turn[10];

        float avgturn=0.0,avgwait=0.0;

        start[0]=0;

        clrscr();

        printf("\n ENTER THE NO. OF PROCESSES:");

        scanf("%d",&n);

        for(i=0;i<n;i++)

        {

                printf("\n ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS%d:",i+1);

                scanf("%d%d",&ser[i],&prior[i]);

                pro[i]=i;
}
for(i=0;i<n;i++)
{
        for(j=0;j<n-i-1;j++)
        {
                if(prior[j]>prior[j+1])
                {       temp1=ser[j];
                        ser[j]=ser[j+1];
                        ser[j+1]=temp1;
                 temp2=prior[j];
                        prior[j]=prior[j+1];
                        prior[j+1]=temp2;
                }
        }
}
for(i=0;i<n;i++)
{       sum=0;
        for(j=0;j<i;j++)
                sum=sum+ser[j];
        start[i]=sum;
}
for(i=0;i<n;i++)
{
        finish[i]=ser[i]+start[i];
        wait[i]=start[i];
        turn[i]=ser[i]+wait[i];
}
```

```
        for(i=0;i<n;i++)
        {       avgwait+=wait[i] ;
                avgturn+=turn[i];
        }
        avgwait/=n;
        avgturn/=n;
        printf("\n PROCESS    SERVICE   PRIORITY   START   FINISH  WAIT TURN \n");
        for(i=0;i<n;i++)
        {       printf("\n\tP%d\t%d \t %d \t %d \t %d \t %d \t %d \n", pro[i],ser[i], prior[i],
                start[i],finish[i],wait[i],turn[i]);
        }
        printf("\n AVERAGE WAITING TIME = %f  tu",avgwait);
        printf("\n AVERAGE TURN AROUND TIME = %f tu ", avgturn);
        getch();
}
```

Priority CPU Scheduling Output:

```
 ENTER THE NO. OF PROCESSES:3


 ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS1:4

2


 ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS2:1

5


 ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS3:3

4


 PROCESS    SERVICE   PRIORITY   START   FINISH  WAIT TURN


     P0    4      2     0     4     0     4


     P1    3      4     4     7     4     7


     P2    1      5     7     8     7     8


 AVERAGE WAITING TIME = 3.666667  tu

 AVERAGE TURN AROUND TIME = 6.333333 tu
```

```
ENTER THE NO. OF PROCESSES:4


ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS1:1

3


ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS2:3

5


ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS3:2

6


ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS4:3

4


PROCESS    SERVICE   PRIORITY   START   FINISH  WAIT TURN


    P0    1     3     0     1     0     1


    P1    3     4     1     4     1     4


    P2    3     5     4     7     4     7


    P3    2     6     7     9     7     9


AVERAGE WAITING TIME = 3.000000  tu

AVERAGE TURN AROUND TIME = 5.250000 tu
```

```
ENTER THE NO. OF PROCESSES:5

ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS1:4

2

ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS2:3

6

ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS3:1

6

ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS4:2

7

ENTER THE SERVICE TIME AND PRIORITY OF  PROCESS5:3

4

PROCESS    SERVICE   PRIORITY   START   FINISH  WAIT TURN


    P0    4     2    0    4    0    4


    P1    3     4    4    7    4    7


    P2    3     6    7    10    7    10


    P3    1     6    10    11    10    11


    P4    2     7    11    13    11    13


AVERAGE WAITING TIME = 6.400000  tu

AVERAGE TURN AROUND TIME = 9.000000 tu
```

PROGRAM 04

STATEMENT OF THE PROBLEM: TO WRITE A C PROGRAM FOR IMPLEMENTING ‘ ROUND ROBIN SCHEDULING ’ ALGORITHM.

Concept: In the round robin scheduling, processes are dispatched in a FIFO manner but are given a limited amount of CPU time called a time-slice or a quantum. If a process does not complete before its CPU-time expires, the CPU is preempted and given to the next process waiting in a queue. The preempted process is then placed at the back of the ready list.

Round Robin Scheduling is preemptive (at the end of time-slice) therefore it is effective in time-sharing environments in which the system needs to guarantee reasonable response times for interactive users. The only interesting issue with round robin scheme is the length of the quantum. Setting the quantum too short causes too many context switches and lower the CPU efficiency. On the other hand, setting the quantum too long may cause poor response time and appoximates FCFS.

Source code :

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int count=0,swt=0,stat=0,i,temp,sq=0;
        int pro[10],st[10],bt[10],wt[10],tat[10],n,tq;
        float atat=0.0,awt=0.0;
        clrscr();
        printf("\n ENTER THE NO. OF PROCESSES:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
```

```
{
        printf("\n ENTER THE SERVICE TIME  OF PROCESS %d:",i);
        scanf("%d",&bt[i]);
        st[i]=bt[i];
        pro[i]=i;
}
printf("\n ENTER THE TIME QUANTUM:");
scanf("%d",&tq);

while(1)
{
        for(i=0,count=0;i<n;i++)
        {
                temp=tq;
                if(st[i]==0)
                {
                        count++;
                        continue;
                }
                if(st[i]>tq)
                        st[i]=st[i]-tq;
                else
                        if(st[i]>=0)
                        {
                                temp=st[i];
                                st[i]=0;
                        }
                        sq=sq+temp;
                        tat[i]=sq;
        }
        if(count==n)
                break;
}
for(i=0;i<n;i++)
{
        wt[i]=tat[i]-bt[i];
        stat=stat+tat[i];
        swt=swt+wt[i];
}
awt=(float)swt/n;
atat=(float)stat/n;
```

```
        printf("\n PROCESS    BURST TIME     WAIT     TURN \n");


        for(i=0;i<n;i++)

        {

                printf("\n\tP%d\t%d \t %d \t%d\n" ,
        pro[i],bt[i],wt[i],tat[i]);

        }


        printf("\n AVERAGE WAITING TIME = %f  tu",awt);

        printf("\n AVERAGE TURN AROUND TIME = %f tu",atat);


        getch();

}
```

Round Robin CPU Scheduling Output:

```
ENTER THE NO. OF PROCESSES:3

ENTER THE SERVICE TIME  OF PROCESS 0:30

ENTER THE SERVICE TIME  OF PROCESS 1:40

ENTER THE SERVICE TIME  OF PROCESS 2:20

ENTER THE TIME QUANTUM:10

PROCESS    BURST TIME     WAIT     TURN

    P0     30      40     70

    P1     40      50     90

    P2     20      40     60

AVERAGE WAITING TIME = 43.333332  tu
AVERAGE TURN AROUND TIME = 73.333336 tu
```

```
ENTER THE NO. OF PROCESSES:4

ENTER THE SERVICE TIME  OF PROCESS 0:12

ENTER THE SERVICE TIME  OF PROCESS 1:20

ENTER THE SERVICE TIME  OF PROCESS 2:9

ENTER THE SERVICE TIME  OF PROCESS 3:5

ENTER THE TIME QUANTUM:5

PROCESS    BURST TIME     WAIT     TURN

    P0     12      24    36

    P1     20      26    46

    P2     9      25    34

    P3     5      15    20

AVERAGE WAITING TIME = 22.500000  tu
AVERAGE TURN AROUND TIME = 34.000000 tu
```

```
ENTER THE NO. OF PROCESSES:5

ENTER THE SERVICE TIME  OF PROCESS 0:22

ENTER THE SERVICE TIME  OF PROCESS 1:11

ENTER THE SERVICE TIME  OF PROCESS 2:34

ENTER THE SERVICE TIME  OF PROCESS 3:2

ENTER THE SERVICE TIME  OF PROCESS 4:12

ENTER THE TIME QUANTUM:6

PROCESS    BURST TIME     WAIT     TURN


   P0     22      43    65



   P1     11      26    37



   P2     34      47    81



   P3     2      18    20



   P4     12      37    49


AVERAGE WAITING TIME = 34.200001  tu

AVERAGE TURN AROUND TIME = 50.400002 tu
```