

# Assignment

**CSE316**

**Submission: 10th April 2020**

**Section-K18ZV**

**Name-Sanket Deshmukh**

**Registration No:-11813665**

**Email-sanketdeshmukh885@gmail.com**

**Roll No:-69**

**Github link:-**

**Question No.:10 and 17**

***Solution no:-10***

1. Write a C program to solve the following problem: Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests, in FIFO

order, is:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

**Introduction:** In operating systems, time interval is extremely important. Since all device requests are linked in queues, the time interval is increased causing the system to hamper. Disk Scheduling Algorithms are used to reduce the entire time interval of any request.

<stdio> : this library is used for input and output.

<stdlib> : this library is used for string.

In the question values are given but they are constant values but

In the code written above we can actually change the values as

Per our request. The requests are fulfilled in the order in which they come. This algorithm does not cause starvation problem.

Total head movements that occur while serving these requests are:

$$(125-86)+(1470-86)+(1470-913)+(1774-913)+(1774-948)+(1509-948)+\\(1509-1022)+(1750-1022)+(1750-130)\\Rightarrow 7063$$



### Code:

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include <stdlib.h>
```

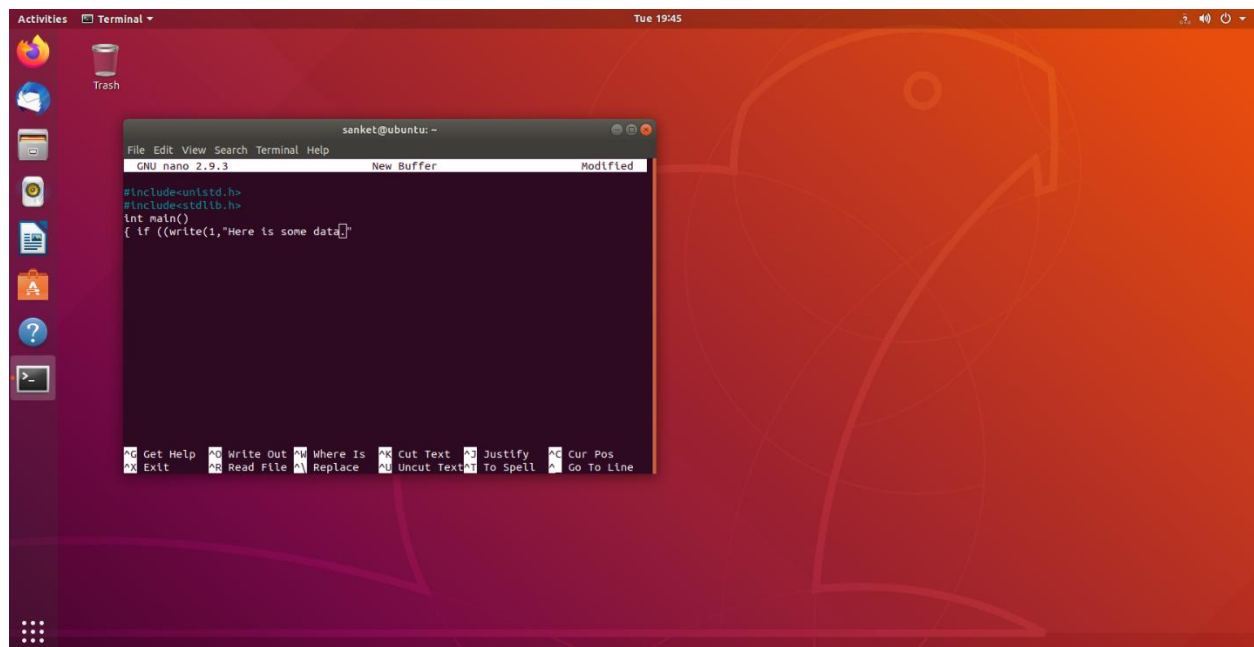
```
int main()
{
    int h,n,m;
    printf("\n");
    printf("Head Position :\n");
    scanf("%d",&h);
    printf("\n");
    printf("No.of Request:\t");
    scanf("%d",&m);
    int req[m];
    for(n=0;n<m;n++)
    {
        scanf("%d",&req[n]);
    }
    int diff=req[0]-h;
    if(diff<0)
    {
        diff=diff*-1;
    }
    for(n=1;n<m;n++)
    {
        if((req[n]-req[n-1])>0)
            diff=diff+(req[n]-req[n-1]);
        else
```

```
        diff=diff+(req[n-1]-req[n]);  
    }  
    printf("Seek time = %d\n",diff);  
  
}
```

### Command to run code:

1. gcc filename.c
2. ./a.out
3. Enter head positi

Complexity of above code:  $n*n$



## *Solution no:-17*

1. Design a scheduling program to implements a Queue with two levels:

Level 1 : Fixed priority preemptive Scheduling

Level 2: Round Robin Scheduling

For a Fixed priority preemptive Scheduling (Queue1), the Priority 0 is highest priority. If one process P1 is scheduled and running, another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2. All the processes in second level queue will complete their execution according to round robin scheduling.

Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.

2. Priority of Queue 2 has lower priority than in Queue 1.

### **Introduction:**

Round robin scheduling algorithm, is employed to schedule process fairly each job a slot or quantum and therefore the interrupting the work if it's not completed by then the job come after the opposite job which are arrived within the quantum time that make these scheduling fairly. Fixed priority preemptive scheduling It is a scheduling system used in real time systems.

<stdio> : this library is used for input and output.

<stdlib> : this library is used for string.

## Code:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
    char z[10][5],tmp[5];
    int m,p,pt[10],wt[10],totwt=0,pr[10],tmp1,t;
    float avgwt;
    printf(" No. Of Processes:");
    scanf("%d",&t);
    for(m=0;m<t;m++)
    {
        printf(" Process %d Name:",m+1);
        scanf("%s",&z[m]);
        printf("Process Time:");
        scanf("%d",&pt[m]);
        printf("Enter Priority:");
        scanf("%d",&pr[m]);
    }
    for(m=0;m<t-1;m++)
    {
        for(p=m+1;p<t;p++)
        {
            if(pr[m]>pr[p])
            {
                tmp1=pr[m];
                pr[m]=pr[p];
                pr[p]=tmp1;
                tmp1=pt[m];
                pt[m]=pt[p];
                pt[p]=tmp1;
                strcpy(tmp,z[m]);
                strcpy(z[m],z[p]);
                strcpy(z[p],tmp);
            }
        }
    }
    wt[0]=0;
    for(m=1;m<t;m++)
    {
        wt[m]=wt[m-1]+wt[m-1];
        totwt=totwt+wt[m];
    }
    avgwt=(float)totwt/t;
    printf("p_name\t p_time\t priority\t w_time\n");
    for(m=0;m<t;m++)
    {
        printf(" %s\t %d\t %d\t %d\n",z[m],pt[m],pr[m],wt[m]);
    }
    printf("Total Waiting Time Of = %d\n Avgerage Waiting Time Of =%f",totwt,avgwt);

    int ts,pid[10],need[10],wt1[10],tat[10],m1,p1,n2,n1;
    int bt[10],flag[10],ttat=0,twt=0;
    float awt,atat;
    printf("\n Number Of Processors \n");
```

```

scanf("%d",&t);
n1=t;
printf("\n Enter Timeslice \n");
scanf("%d",&ts);
for(m=1;m<=t;m++)
{
    printf("\n The process ID %d",m);
    scanf("%d",&pid[m]);
    printf("\n Burst Time For The process");
    scanf("%d",&bt[m]);
    need[m]=bt[m];
}
for(m=1;m<=t;m++)
{
    flag[m]=1;
    wt[m]=0;
}
while(t!=0)
{
    for(m=1;m<=t;m++)
    {
        if(need[m]>=ts)
        {
            for(p=1;p<=t;p++)
            {
                if((m!=p)&&(flag[m]==1)&&(need[p]!=0))
                    wt[p]+=ts;
            }
            need[m]-=ts;
            if(need[m]==0)
            {
                flag[m]=0;
                t--;
            }
        }
        else
        {
            for(p=1;p<=t;p++)
            {
                if((m!=p)&&(flag[m]==1)&&(need[p]!=0))
                    wt[p]+=need[m];
            }
            need[m]=0;
            t--;
            flag[m]=0;
        }
    }
}
for(m=1;m<=n1;m++)
{
    tat[m]=wt[m]+bt[m];
    twt=twt+wt[m];
    ttat=ttat+tat[m];
}
awt=(float)twt/n1;
atat=(float)ttat/n1;
printf("\n\n Process \t Process ID \t BurstTime \t Waiting Time \t TurnaroundTime \n
");
for(m=1;m<=n1;m++)
{

```

```

        printf("\n %5d \t %5d \t\t %5d \t\t %5d \t\t %5d \n",
m,pid[m],bt[m],wt[m],tat[m]);
    }
    printf("\n The average Waiting Time=4.2f",awt);
    printf("\n The average Turn around Time=4.2f",atat);
}
}

```

## Command to run code:

1. gcc filename.c
2. ./a.out
3. Enter no of process

The screenshot shows a Windows desktop with a Dev-C++ IDE. The IDE window displays the source code for a Round Robin scheduling algorithm. The code defines arrays for process ID, burst time, waiting time, and turn-around time. It prompts the user to enter the number of processes, process names, priorities, and burst times. The output window shows the execution results for two processes named 'dee'. The first process has a burst time of 15 and a waiting time of 10, resulting in a turn-around time of 25. The second process has a burst time of 20 and a waiting time of 0, resulting in a turn-around time of 20. The average waiting time is 4.2f and the average turn-around time is 4.2f.

```

52     int ts,pid[10],need[10],wt[10],tat[10],m1,p1,n2,n1;
53     int bt[10],flag[10],ttat=0,ttw=0;

```

Output:

```

No. Of Processes:2
Process 1 Name:dee
Process Time:8
Enter Priority:2
Process 2 Name:dee
Process Time:153
Enter Priority:1
p_name  p_time priority  w_time
dee      153      1      0
dee      8       2      0
Total Waiting Time Of  = 0
Average Waiting Time Of =0.000000
Number Of Processors
2
Enter Timeslice
10
The process ID 1      1
Burst Time For The process15
The process ID 2      2
Burst Time For The process20

```

Process	Process ID	BurstTime	Waiting Time	TurnaroundTime
1	1	15	10	25
2	2	20	0	20

Compiler Output:

```

- Warnings: 0
- Output Filename: C:\Users\HP\Desktop\Untitled1.exe
- Output Size: 131.44140625 KiB
- Compilation Time: 0.28s

```



```
52     int ts,pid[10],need[10],wt1[10],tat[10],m1,p1,n2,n1;
53     int bt[10],flag[10],ttat=0,twt=0;
```

Number Of Processors  
2

Enter Timeslice  
10

The process ID 1      1  
Burst Time For The process15

The process ID 2      2  
Burst Time For The process20

Process	Process ID	BurstTime	Waiting Time	TurnaroundTime
1	1	15	10	25
2	2	20	15	35

The average Waiting Time=4.2f  
The average Turn around Time=4.2f

Process exited after 48.58 seconds with return value 0  
Press any key to continue . . .

Compiler Resources

Shorten compiler paths

Warnings: 0  
Output Filename: C:\Users\HP\Desktop\Untitled1.exe  
Output Size: 131.44140625 KiB  
Compilation Time: 0.28s

Line: 62 Col: 40 Sel: 0 Lines: 119 Length: 3079 Insert Done parsing in 0.016 seconds

Type here to search

3:19 PM 4/10/2020

**Complexity of above code:  $n \log n$**