An Online System on

OPERATING SYSTEM

Course code: CSE 316

Section: K18GE

Submitted by:-

Name: Akshat Pareek

Roll No.: 31

Registration No.: 11811340

Submitted to:-

Name of the faculty: Milanjit Kaur

Date of submission of report: 10-April-2020

Department of Intelligent Systems
School of Computer Science Engineering
Lovely Professional University, Jalandhar
APRIL-2020

Student Name: **AKSHAT PAREEK**
Student ID: **11811340**
Email Address: **dprinceakshat9@gmail.com**
GitHub Link: **https://github.com/AkshatPareek9/linux_operating_system**
Code: **C language program**

```
//header files
#include<stdio.h>                    //define the printf and scanf function
struct process
{
int pid;                             //process id
char pname[20];                      //process name
int at;                              //arrival time
int bt;                              //burst time
int cmpt;                            //completion time
int rbt;                             //remaining burst time
}f[50],s[50],m[50];                  //f denote faculty && s denote student
int n, quanta, fc=0, sc=0, mc=0;
int instruction()                    //complexity is O(1)
{       int i;
printf("\n\tWELCOME TO THE WORLD OF OPERATING SYSTEM\n\tProject developed by Akshat Pareek\n\tRegistration number: 11811340\n\tRoll number: 31\n\tSection: K18GE\n\n");
for(i=0; i<100; i++)
{       printf("#");     }
printf("\n\tWelcome, please follow the instruction for proper functioning of this system...");
```

```
printf("\n	1. Enter time in 2400 format eg, 10:30 am should be enter as 1030.");
printf("\n	2. Enter query arrival time in ascending order, i.e in real time arrival manner.");
printf("\n	3. You can come between 10:00 am to 12:00 pm only.");
printf("\n	4. All time units are in minutes.");
printf("\n	5. Please enter the right value of burst time so that CPU cann't be idle.");
printf("\n	6. Round Robin and Multilevel Queue are used in this program.\n");
for(i=0;i<100;i++)
{	printf("#");	}
return 0;	}
void input()			//complexity is O(n)
{	int t,i;
char ch;
printf("\n\nEnter total no of queries: ");			//no of queries
scanf("%d",&n);
if(n==0)
{	printf("No queries");	}
else
{
printf("\nEnter Quanta number for each process: ");		//total quantum number
scanf("%d",&quanta);
printf("\nEnter f for faculty and s for student: \n");
for(i=0; i<n; i++)
{
printf("\nJob Type(f/s): ");
```

```
scanf("%s",&ch);
if(ch=='f')
{
printf("Query ID: ");
scanf("%d",&f[fc].pid);
printf("Process Name: ");
scanf("%s",f[fc].pname);
printf("Arrival Time: ");
scanf("%d",&t);
if(t<1000 || t>=1200)
{
printf("\nCome at another time between 10 am to 12pm");
i=i-1;
continue;
}
else
{
if(t>1059 && t<1200)
{
f[fc].at= t-1040;
}
else if(t>=1000 && t<1100)
{
f[fc].at= t-1000;
```

```
}
}
printf("Burst time: ");
scanf("%d",&f[fc].bt);
f[fc].rbt=f[fc].bt;
fc++;
}
else if(ch=='s')
{
printf("Query ID: ");
scanf("%d",&s[sc].pid);
printf("Process Name: ");
scanf("%s",s[sc].pname);
printf("Arrival Time: ");
scanf("%d",&t);
if(t<1000 || t>=1200)
{
printf("\nCome at another time between 10 am to 12pm");
i=i-1;
continue;
}
else
{
if(t>1059 && t<1200)
```

```
{
s[sc].at= t-1040;
}
else if(t>=1000 && t<1100)
{
s[sc].at= t-1000;
}
}
printf("Burst time: ");
scanf("%d",&s[sc].bt);
s[sc].rbt=s[sc].bt;
sc++;
}
else
{
printf("It is not a correct job type, please enter again...");
i--;
continue;
}       }       }       }
void ready()                    //this function is used to make process ready for system call
{
int isc=0, ifc= 0;
if(fc!=0  && sc!=0)
{
```

```
while(isc<sc && ifc<fc)
{
if(f[ifc].at == s[isc].at)
{
m[mc] = f[ifc];
mc++;
ifc++;
m[mc] = s[isc];
mc++;
isc++;
}
else if(f[ifc].at < s[isc].at)
{
m[mc]= f[ifc];
mc++;
ifc++;
}
else if(f[ifc].at > s[isc].at)
{
m[mc]= s[isc];
mc++;
isc++;
}
}
```

```
if(mc != (fc+sc))
{
while(ifc!=fc)
{
m[mc] = f[ifc];
mc++;
ifc++;
}
while(isc!=sc)
{
m[mc] = s[isc];
mc++;
isc++;
}
}
}
else if(fc==0)
{
while(isc!=sc)
{
m[mc] = s[isc];
mc++;
isc++;
}
```

```
}
else if(sc==0)
{
while(ifc!=fc)
{
m[mc] = f[ifc];
mc++;
ifc++;
}
}
else
{
printf("No jobs available...\n");
}       }
void round_robin()              //whole work is done in this function by calling system call
//complexity is O(n)
{
int time = m[0].at, mark=0, cc=0,i;
while(cc!=mc)
{
for(i=0; i<=mark; i++)
{
if(m[i].rbt > quanta)
{
```

```
time += quanta;
m[i].rbt -= quanta;
}
else if(m[i].rbt <= quanta && m[i].rbt !=0)
{
time += m[i].rbt;
m[i].rbt =0;
m[i].cmpt = time;
cc++;
}
else
{
continue;
}
}
int start = mark+1,rc;
for(rc = start; rc < mc; rc++)
{
if(m[rc].at <= time)
{
mark++;
}
}
}
```

```
}
void printer()                          //it summarize all work to the user as output
//complexity is O(n)
{
int total=0,i;
double avg,sum=0;
printf("\n\n\t\t\t**********SUMMARY FOR THE EXECUTION**********\n");
printf("\n\tQuery ID\tProcess Name\tArrival Time\tBurst Time\tCompletion Time\tTurn Around Time\tWaiting Time\n");
printf("\t_____
_____\n");
for(i=0; i<mc; i++)
{
if(m[i].at>59 && m[i].at<120)
{
if(m[i].cmpt>59 && m[i].cmpt<120)
{
printf("\n\t  %d\t\t%s\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d", m[i].pid, m[i].pname, (m[i].at+1040), m[i].bt, (m[i].cmpt+1040), (m[i].cmpt-m[i].at), ((m[i].cmpt-m[i].at)- m[i].bt));
}
else if(m[i].cmpt>=120)
{
printf("\n\t  %d\t\t%s\t\t%d\t\t%d\t\t%d        \t\t%d\t\t%d", m[i].pid, m[i].pname, (m[i].at+1040), m[i].bt, (m[i].cmpt+1080), (m[i].cmpt-m[i].at), ((m[i].cmpt-m[i].at)- m[i].bt));
}
```

```
}
else if(m[i].at>=0 && m[i].at<60)
{
if(m[i].cmpt>=0 && m[i].cmpt<60)
{
printf("\n\t  %d\t\t%s\t\t%d\t\t%d\t\t%d         \t\t%d\t\t%d", m[i].pid, m[i].pname,
(m[i].at+1000), m[i].bt, (m[i].cmpt+1000), (m[i].cmpt-m[i].at), ((m[i].cmpt-m[i].at)-
m[i].bt));
}
else if(m[i].cmpt>59 && m[i].cmpt<120)
{
printf("\n\t  %d\t\t%s\t\t%d\t\t%d\t\t%d         \t\t%d\t\t%d", m[i].pid, m[i].pname,
(m[i].at+1040), m[i].bt, (m[i].cmpt+1040), (m[i].cmpt-m[i].at), ((m[i].cmpt-m[i].at)-
m[i].bt));
}
else if(m[i].cmpt>=120)
{
printf("\n\t  %d\t\t%s\t\t%d\t\t%d\t\t%d         \t\t%d\t\t%d", m[i].pid, m[i].pname,
(m[i].at+1000), m[i].bt, (m[i].cmpt+1080), (m[i].cmpt-m[i].at), ((m[i].cmpt-m[i].at)-
m[i].bt));
}
}
if(total<m[i].cmpt)
{
total= m[i].cmpt;
}
```

```
sum+= ((m[i].cmpt-m[i].at)-m[i].bt);
}
printf("\n\t_____
_____\n");
avg = sum/mc;
printf("\n\nTotal time Spent for all queries: %d",total);
printf("\nAverage query time: %f",avg);
printf("\n\n\t\t\t\t**********Process Execution Complete**********\n");
}
int main()
{
instruction();
input();
ready();
round_robin();
printer();
}
```

PROBLEM IN TERMS OF OPERATING SYSTEM

Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.

The given problem is based upon scheduling algorithm for solving queries of persons of different classes i.e. **Faculty** and **Student**. Thus, these queries can be compared to different processes in terms of operating system where each process has its demands and needs resources and time for its execution. This demands of different processes are handled by the CPU.

In the given scenario, Mr. Sudesh Sharma, Linux expert, can be considered as a CPU, who solves the queries of either Faculty or Student by allocating proper resources to their individual demands and processing them by allocating them time accordingly. Now, Mr. Sudesh Sharma, wants to provide priority for each query based upon its class, as well as, he wants to dedicate a fixed amount of time to every request. Thus in operating system if we divide the requests into two separate queues i.e. Faculty and Student such that the first queue contains faculty queries has higher priority and the second contains student queries which has lower priority, then we can resolve the problem, by allocating them required resources based upon their priorities as done in the scheduling algorithm in operating systems.

This program contains the concept of **Round Robin** and **Multilevel queue algorithm.**

Main() : contains 5 different functions which has their own function to be processed.

Instruction() : guides to follow some points for better functioning.

Input() : takes all the input values required in the program such as no of queries, quantum number, arrival time and burst time.

Ready() : Put 2 separate queues into single queue according to their priority and arrival time.

Round_robin() : tell the process when to start and when to end.

Printer() : It prints the summary that will be shown to user including average waiting time.

ALGORITHM:

1. Take all the inputs from the user.
2. Arrange the processes in two different queues i.e. faculty and student.
3. Put the two different queues into single queue according to arrival time and priority.
4. Select that process which has minimum arrival time and minimum burst time.
5. After completion of process, select next process which has minimum arrival time and minimum burst time.
6. After completion of all processes, summarize the process with their completion time.

COMPLEXITY OF THE ALGORITHM:

The complexity of the scheduling algorithm is O(n).

CONSTRAINTS

- Round robin CPU scheduling algorithm cannot be implemented in real time operating system due to high context switching, large waiting time, large response time, large turnaround time and less throughput.
- The proposed algorithm improves all the drawbacks of round robin CPU scheduling algorithm.
- It reduces the problem of starvation and also implement the concept of aging.

CODE SNIPPET:

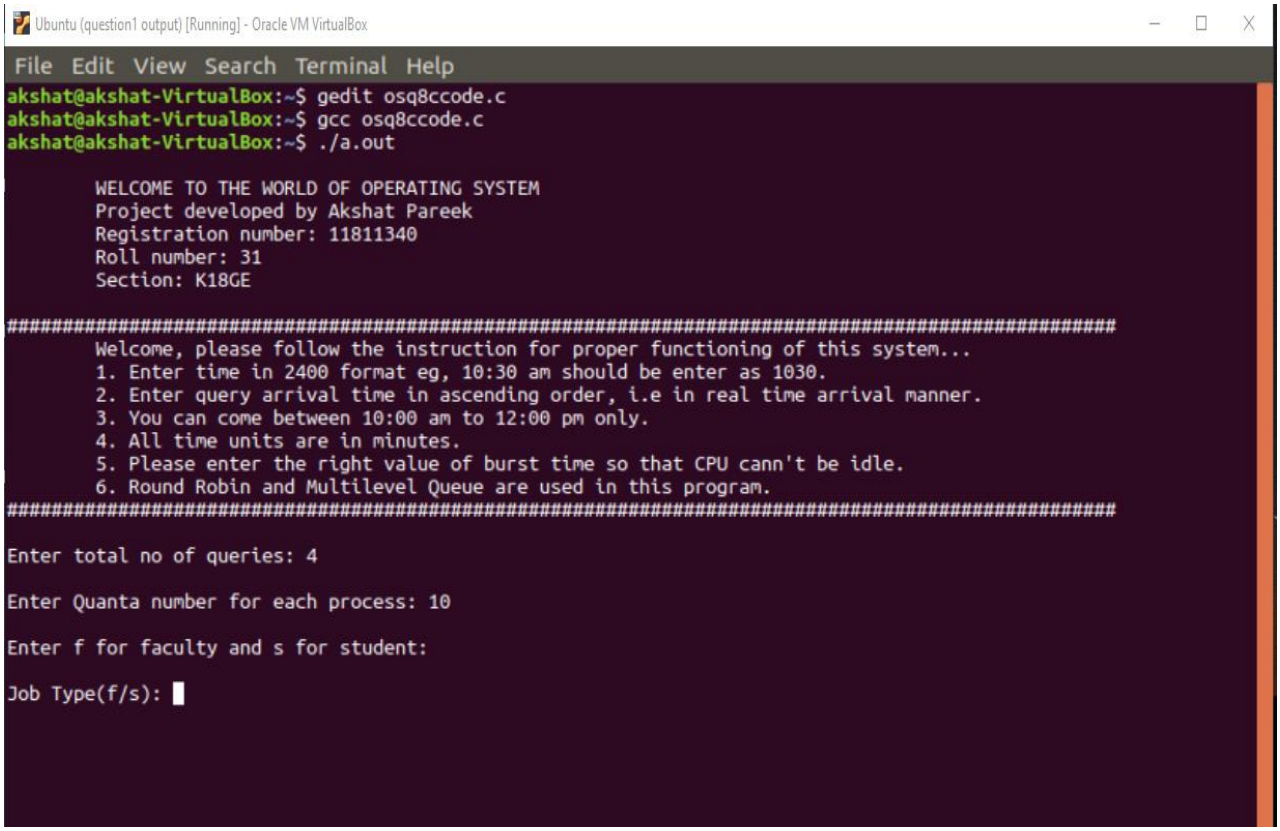
```
void round_robin()
{       int time = m[0].at, mark=0, cc=0,i;
while(cc!=mc)
{       for(i=0; i<=mark; i++)
        {       if(m[i].rbt > quanta)
                {       time += quanta;
```

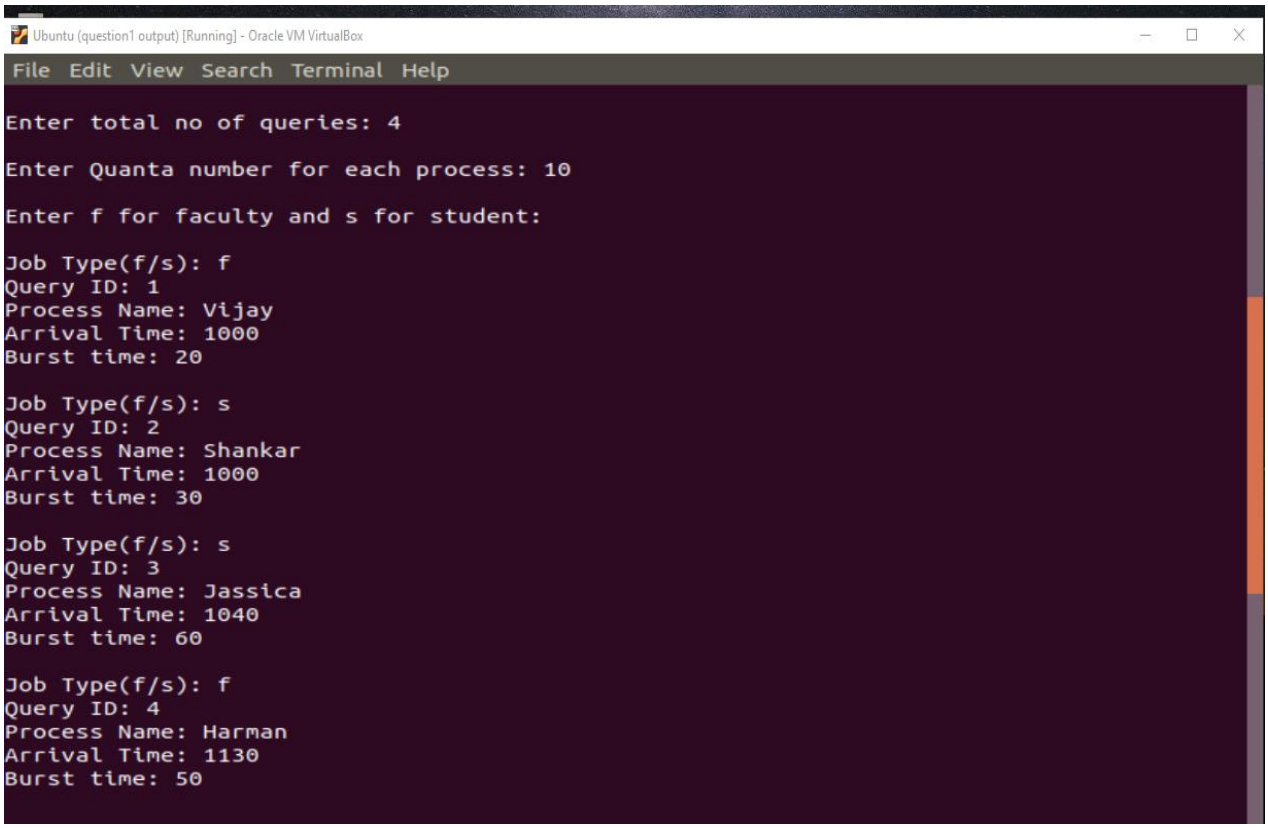
```
                m[i].rbt -= quanta;      }
            else if(m[i].rbt <= quanta && m[i].rbt !=0)
            {       time += m[i].rbt;
                    m[i].rbt =0;
                    m[i].cmpt = time;
                    cc++;           }
            else
            {       continue;       }
    }
    int start = mark+1,rc;
    for(rc = start; rc < mc; rc++)
    {       if(m[rc].at <= time)
            {       mark++;         }
    }       }       }
```

BOUNDARY CONDITION:

- Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
- Process cannot be interrupted until it terminates itself or its time is up.
- If a process with long burst time is running CPU, then later coming process with less CPU burst time may starve.
- It does not have overheads.
- The process is rigid.
- No cost associated.

TEST CASE


Ubuntu (question1 output) [Running] - Oracle VM VirtualBox
File Edit View Search Terminal Help
akshat@akshat-VirtualBox:~$ gedit osq8ccode.c
akshat@akshat-VirtualBox:~$ gcc osq8ccode.c
akshat@akshat-VirtualBox:~$ ./a.out
WELCOME TO THE WORLD OF OPERATING SYSTEM
Project developed by Akshat Pareek
Registration number: 11811340
Roll number: 31
Section: K18GE
Welcome, please follow the instruction for proper functioning of this system...
1. Enter time in 2400 format eg, 10:30 am should be enter as 1030.
2. Enter query arrival time in ascending order, i.e in real time arrival manner.
3. You can come between 10:00 am to 12:00 pm only.
4. All time units are in minutes.
5. Please enter the right value of burst time so that CPU cann't be idle.
6. Round Robin and Multilevel Queue are used in this program.
Enter total no of queries: 4
Enter Quanta number for each process: 10
Enter f for faculty and s for student:
Job Type(f/s):


Ubuntu (question1 output) [Running] - Oracle VM VirtualBox
File Edit View Search Terminal Help
Enter total no of queries: 4
Enter Quanta number for each process: 10
Enter f for faculty and s for student:
Job Type(f/s): f
Query ID: 1
Process Name: Vijay
Arrival Time: 1000
Burst time: 20
Job Type(f/s): s
Query ID: 2
Process Name: Shankar
Arrival Time: 1000
Burst time: 30
Job Type(f/s): s
Query ID: 3
Process Name: Jassica
Arrival Time: 1040
Burst time: 60
Job Type(f/s): f
Query ID: 4
Process Name: Harman
Arrival Time: 1130
Burst time: 50

Ubuntu (question1 output) [Running] - Oracle VM VirtualBox

File Edit View Search Terminal Help

**********SUMMARY FOR THE EXECUTION**********

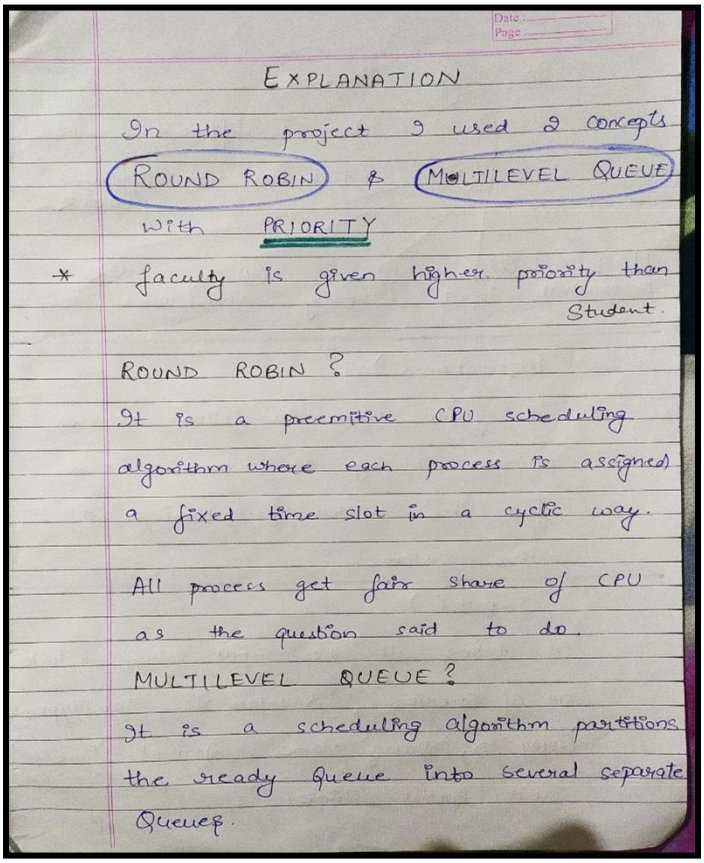
Query ID	Process Name	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
1	Vijay	1000	20	1020	20	0
2	Shankar	1000	30	1050	50	20
3	Jassica	1040	60	1200	80	20
4	Harman	1130	50	1240	70	20

Total time Spent for all queries: 160
Average query time: 15.000000

**********Process Execution Complete**********

```
akshat@akshat-VirtualBox:~$
akshat@akshat-VirtualBox:~$
akshat@akshat-VirtualBox:~$
akshat@akshat-VirtualBox:~$
akshat@akshat-VirtualBox:~$
```

EXPLANATION



EXPLANATION

In the project I used 2 concepts ROUND ROBIN & MULTILEVEL QUEUE with PRIORITY

* faculty is given higher priority than Student.

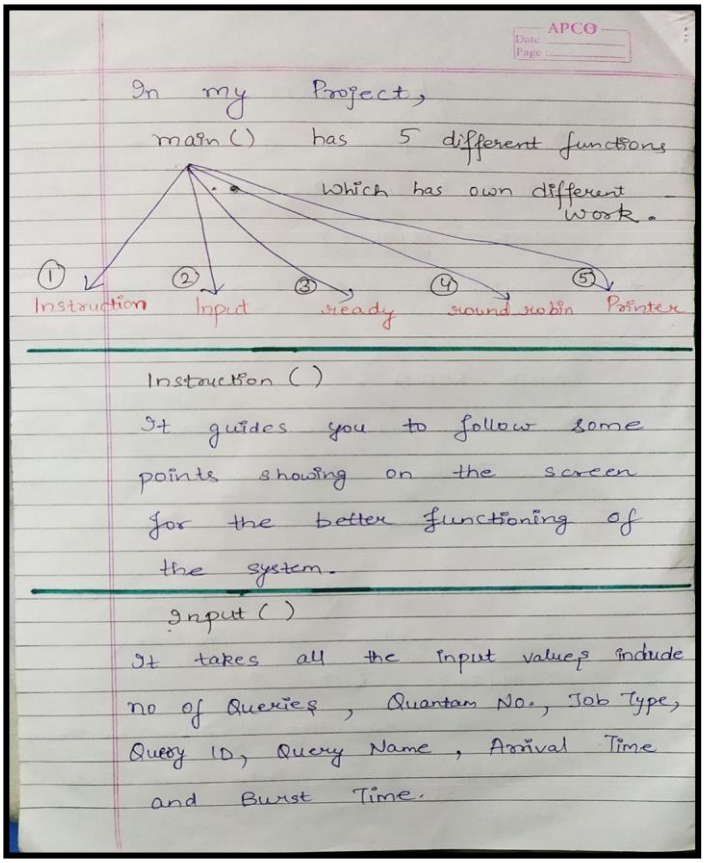
ROUND ROBIN ?

It is a preemitive CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way.

All process get fair share of CPU as the question said to do.

MULTILEVEL QUEUE ?

It is a scheduling algorithm partitions the ready queue into several separate Queues.



In my Project, main() has 5 different functions which has own different work.

Instruction()

It guides you to follow some points showing on the screen for the better functioning of the system.

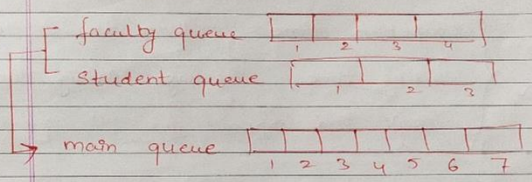
Input()

It takes all the input values include no of Queries, Quantam No., Job Type, Query ID, Query Name, Arrival Time and Burst Time.

ready ()

It is used for system call.

It puts 2 different separate queues into 1 queue according to their priority and arrival time.



Round_Robin () → important function

All the main work for which this LINUX system has been made are contain in this function.

It tell the process when to start and according to quanta no, priority & arrival time, it complete the process and save their completion time.

Printer ()

This function is used as a Output function. All the summary will be executed and output will be shown to the user through this function.

o/p shown
- Completion time, Turn Around Time
- Waiting time, Average Waiting Time,
- Total time spend for all queries.

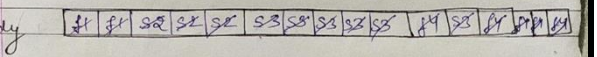
Turn Around Time = Completion Time - Arrival Time

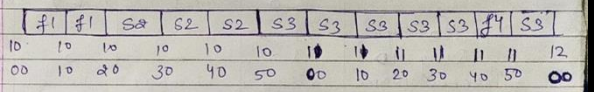
Waiting Time = TAT - Burst Time.

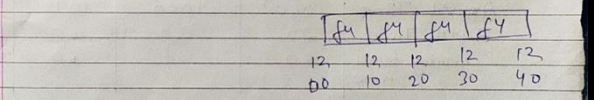
Round_Robin ()

Process	AT	BT	RBT	
f1	1000	20	~~20~~	~~10~~ 0
s2	1000	30	~~30~~	~~20~~ ~~10~~ 0
s3	1040	60	~~60~~	~~50~~ ~~40~~ ~~30~~ ~~20~~ ~~10~~ 0
f4	1130	50	~~50~~	~~40~~ ~~30~~ ~~20~~ ~~10~~ 0

Ready







Printer () → Output function

avg = sum / no

↓ total waiting time

→ total no of process

Example →

Input ()

No of Queries (n) → 4

(n ≠ 0)

Quanta No (quanta) → 10

for (i = 0 ; i < 4 ; i++)

	i=0	i=1	i=2	i=3
Job Type	f	s	s	f
Query ID	1	2	3	4
Process Name	Vijay	Shankar	Jassica	Harman
Arrival Time	1000	1000	1040	1130
Burst Time	20	30	60	50
Remaining burst time	20	30	60	50

Ready ()

faculty [f1 | f4]

student [s2 | s3]

main [f1 | s2 | s3 | f4]