MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

Operating System Mini-Project

TITLE: SCHEDULING ALGORITHMS

Presented By:

➔Kishan Nayak K	190905008	Roll No. 3
➔Shreyas Kamath	190905022	Roll No. 7
➔Shyam Vaideeaswaran	190905041	Roll No. 9
➔Jaideep	190905092	Roll No. 18

Section:B Batch: B1

Submitted To:

Mr. Manjunath Mulimani

Assistant Professor

Computer Science and Engineering

Manipal Institute of Technology

Hardware/ Programming languages used:

C program

1. Abstract:

Developing Efficient Round Robin Scheduling algorithm which will perform superior than the current Round Robin algorithm in terms of minimizing average waiting time(AWT),average turnaround time(ATT) and number of contact switches.

2. Description:

Scheduling technique plays an important role in process management. We are here to demonstrate the modified Round Robin Scheduling algorithm by implementing it in C language. Thus we are calculating Average Waiting time(AWT),Average Turnaround time(ATT) by drawing a simple table and comparing the same with the already existing algorithm mainly First Come First Serve(FCFS), Shortest Job First(SJF) and Round Robin Technique(RR).

Here we programmatically implement all 3 algorithms along with modified Round Robin. Users have the privilege to enter Arrival Time and Burst time as Input and check the result for all the cases.

3. CPU Scheduling Algorithms

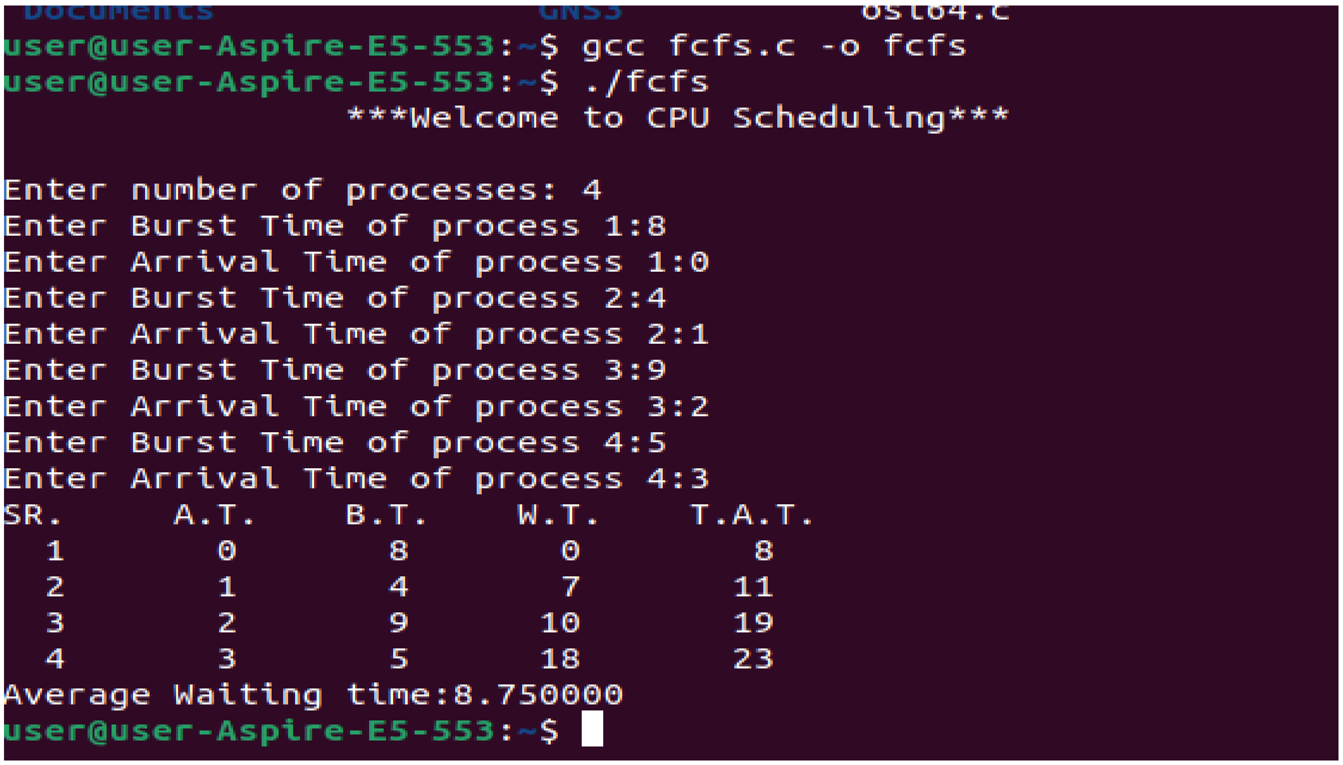
First Come First Serve(FCFS):

- Jobs are executed on a first come, first serve basis.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as the average wait time is high.

Code:

```
#include<stdio.h>
#include<curses.h>
#include<curses.h>
#include<math.h>
#include<string.h>
int wt[100],at[100],bt[100],tat[100],n,p[100];
float awt[5],atat[5];
int temp1,temp2,temp3,sqt,avg;
void fcfs(){
wt[0]=0;
atat[0]=tat[0]=bt[0];
int btt=bt[0];
int i;
for(i=1;i<n;i++){
wt[i]=btt-at[i];
btt+=bt[i];
awt[0]+=wt[i];
tat[i]=wt[i]+bt[i];
atat[0]+=tat[i];
}
atat[0]/=n;
awt[0]/=n;
printf("SR.\tA.T.\tB.T.\tW.T.\tT.A.T.\n");
for(i=0;i<n;i++)
{
printf("%3d\t%3d\t%3d\t%3d\t%4d\n",i+1,at[i],bt[i],wt[i],tat[i]);
}
}
```

```
void input()
{
printf("Enter number of processes: ");
scanf("%d",&n);
int i;
for(i=0;i<n;i++)
p[i]=i+1;
for(i=0;i<n;i++)
{
printf("Enter Burst Time of process %d:",i+1);
scanf("%d",&bt[i]);
printf("Enter Arrival Time of process %d:",i+1);
scanf("%d",&at[i]);}

for(i=0;i<5;i++)
{
awt[i]=0.0;
atat[i]=0.0;
}
}
void changeArrival(){
int a=at[0];
int i;
for(i=0;i<n;i++){
if(at[i]<a)
a=at[i];
}
if(a!=0){
for(i=0;i<n;i++)
at[i]=at[i]-a;
}
}
void display(int c)
{
int i;
printf("Average Waiting time:%f\nAverage Turn Around Time:%f",awt[c-2],atat[c-2]);
}
int main(){char c;
printf("\t\t***Welcome to CPU Scheduling***\n\n");
input();
fcfs();
display(2);
}
```



```
 Documents                      GNS3                    os164.c
user@user-Aspire-E5-553:~$ gcc fcfs.c -o fcfs
user@user-Aspire-E5-553:~$ ./fcfs
                ***Welcome to CPU Scheduling***

Enter number of processes: 4
Enter Burst Time of process 1:8
Enter Arrival Time of process 1:0
Enter Burst Time of process 2:4
Enter Arrival Time of process 2:1
Enter Burst Time of process 3:9
Enter Arrival Time of process 3:2
Enter Burst Time of process 4:5
Enter Arrival Time of process 4:3
SR.     A.T.     B.T.     W.T.     T.A.T.
  1       0        8        0         8
  2       1        4        7        11
  3       2        9       10        19
  4       3        5       18        23
Average Waiting time:8.750000
user@user-Aspire-E5-553:~$ 
```

Shortest Job First(SJF):

- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processor should know in advance how much time the process will take.

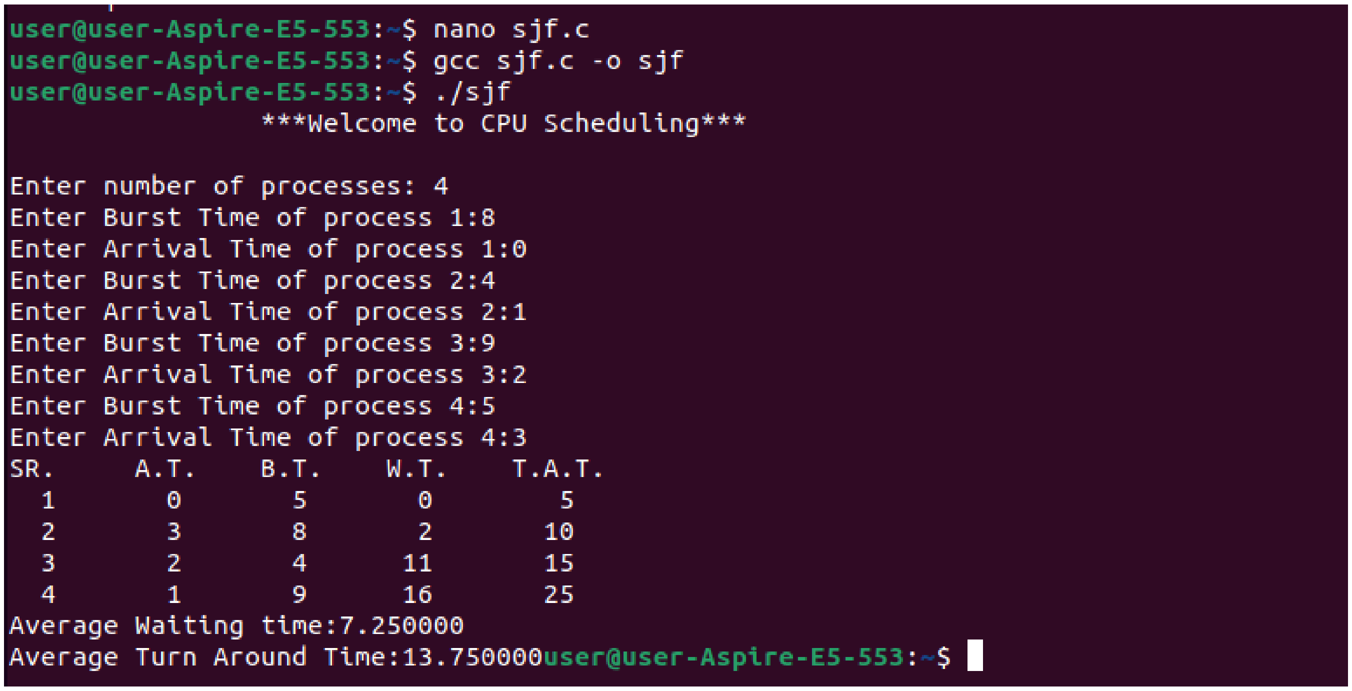
Code:

```
#include<stdio.h>

#include<curses.h>

#include<math.h>

#include<string.h>

int wt[100],bt[100],at[100],tat[100],n,p[100];

float awt[5],atat[5];

void sjf()

{

  float wavg=0,tavg=0,wsum=0,tsum=0;

  int i,j,temp;

  int sum=0,ta=0;

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

  {

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

      {

           if(at[i]<at[j])

           {
```

```
                        temp=p[j];
                        p[j]=p[i];
                        p[i]=temp;
                        temp=bt[j];
                        bt[j]=bt[i];
                        bt[i]=temp;
                }
        }
}
int btime=0,min,k=1;
for(j=0;j<n;j++)
{
        btime=btime+bt[j];
        min=bt[k];
        for(i=k;i<n;i++)
        {
                if(btime>=at[i] && bt[i]<min)
                {
                        temp=p[k];
                        p[k]=p[i];
                        p[i]=temp;
                        temp=at[k];
                        at[k]=at[i];
                        at[i]=temp;

                        temp=bt[k];
                bt[k]=bt[i];
                bt[i]=temp;
            }
        }
        k++;
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        sum+=bt[i-1];
        wt[i]=sum-at[i];
        wsum=wsum+wt[i];
    }
    awt[1]=wsum/n;
    for(i=0;i<n;i++)
    {
        ta+=bt[i];
        tat[i]=ta-at[i];
        tsum+=tat[i];
    }
    atat[1]=(tsum/n);
    printf("SR.\tA.T.\tB.T.\tW.T.\tT.A.T.\n");
    for(i=0;i<n;i++)
    {
        printf("%3d\t%3d\t%3d\t%3d\t%4d\n",i+1,at[i],bt[i],wt[i],tat[i]);
    }
```

```
}
void input()
{
    printf("Enter number of processes: ");
    scanf("%d",&n);
    int i;
    for(i=0;i<n;i++)
    p[i]=i+1;
    for(i=0;i<n;i++)
    {
           printf("Enter Burst Time of process %d:",i+1);
           scanf("%d",&bt[i]);
           printf("Enter Arrival Time of process %d:",i+1);
           scanf("%d",&at[i]);
    }
    for(i=0;i<5;i++)
    {
           awt[i]=0.0;
           atat[i]=0.0;
    }
}
void changeArrival()
{
    int a=at[0];
    int i;
    for(i=0;i<n;i++)
```

```
   {
         if(at[i]<a)
         a=at[i];
   }
   if(a!=0)
   {
         for(i=0;i<n;i++)
         at[i]=at[i]-a;
   }
}
void display(int c)
{
   int i;
   printf("Average Waiting time:%f\nAverage Turn Around Time:%f",awt[c-2],atat[c-2]);
}
int main()
{
   int  c=3;
   printf("\t\t***Welcome to CPU Scheduling***\n\n");
   input();
   changeArrival();
   sjf();
   display(c);
}
```



```
user@user-Aspire-E5-553:~$ nano sjf.c
user@user-Aspire-E5-553:~$ gcc sjf.c -o sjf
user@user-Aspire-E5-553:~$ ./sjf
                ***Welcome to CPU Scheduling***

Enter number of processes: 4
Enter Burst Time of process 1:8
Enter Arrival Time of process 1:0
Enter Burst Time of process 2:4
Enter Arrival Time of process 2:1
Enter Burst Time of process 3:9
Enter Arrival Time of process 3:2
Enter Burst Time of process 4:5
Enter Arrival Time of process 4:3
SR.     A.T.    B.T.    W.T.    T.A.T.
  1       0       5       0        5
  2       3       8       2       10
  3       2       4      11       15
  4       1       9      16       25
Average Waiting time:7.250000
Average Turn Around Time:13.750000user@user-Aspire-E5-553:~$ 
```

Round Robin:

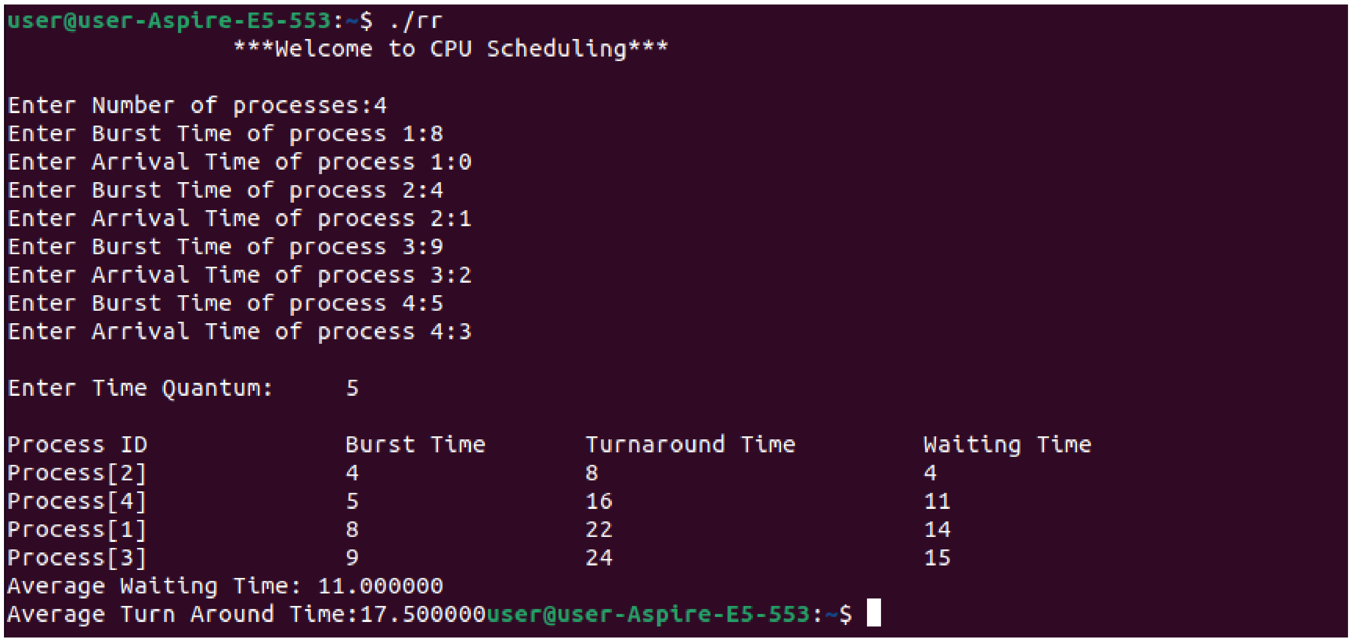
- Each process is provided a fixed time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and another process executes for a given time period.
- Context switching is used to save states of preempted processes.
- Performance of a simple Round Robin algorithm depends upon the size of the time quantum. If we select the value of time quantum is too short, number of context switches increases due to more context switches, the performance of CPU or the efficiency of CPU is degraded.
- On the other hand, if we select the value of time quantum is too large the number of context switches is less but the waiting time of processes is increasing.

Code:

```
#include <stdio.h>

#include<stdlib.h>
```

```
#include<math.h>

#include<string.h>

// #include<conio.h>


int wt[100],bt[100],at[100],tat[100],n,p[100];

float awt[5],atat[5];

int temp1,temp2,temp3,sqt,avg;

void changeArrival()

{

    int a = at[0];

    int i;

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

    {

        if(at[i] < a)

        {

            a = at[i];

        }

    }

    if(a != 0){

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

            at[i] = at[i] - a;

    }

}

void input()

{

 printf("Enter Number of processes:");
 scanf("%d",&n);
 int i;
for(i=0;i<n;i++)
p[i]=i+1;
 for(i=0;i<n;i++)
 {
 printf("Enter Burst Time of process %d:",i+1);
 scanf("%d",&bt[i]);
 printf("Enter Arrival Time of process %d:",i+1);
 scanf("%d",&at[i]);
 }
for(i=0;i<5;i++)
{
awt[i]=0.0;
{
int i, total = 0, x, counter = 0, time_quantum;
 int wait_time = 0, turnaround_time = 0, temp[100];
x=n;
for(i = 0; i < n; i++)
 {
temp[i] = bt[i];
 tem }
 printf("\nEnter Time Quantum:\t");
 scanf("%d", &time_quantum);
 printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
 for(total = 0, i = 0; xatat[i]=0.0;
}

}


void rr()

 != 0;)

 {

 if(temp[i] <= time_quantum && temp[i] > 0)

 {

 total = total + temp[i];

 temp[i] = 0;

 counter = 1;

 }

 else if(temp[i] > 0)

 {

p[i] = temp[i] - time_quantum;

 total = total + time_quantum;

 }

 if(temp[i] == 0 && counter == 1)

 {

 x--;

printf("Process[%d]\t\t%d\t\t %d\t\t\t %d\n", i + 1, bt[i], total -

at[i], total - at[i] - bt[i]);

 wait_time = wait_time + total - at[i] - bt[i];

 turnaround_time = turnaround_time + total - at[i];

 counter = 0;

 }
 if(i == n - 1)
 {
i = 0;
}
 else if(at[i + 1] <= total)
 {
i++;
}
 else
 {
i = 0;
}
}
awt[2] = wait_time * 1.0 / n;
atat[2] = turnaround_time * 1.0 / n;
}

void display(int c)
{
 int i;
 printf("Average Waiting Time: %f\nAverage Turn Around Time:%f",awt[c-2],atat[c-2]);
}

int main(){
 int  c=4;
          printf("\t\t***Welcome to CPU Scheduling***\n\n");
```

```
        input();

        changeArrival();

        rr();

        display(c);

}
```



```
user@user-Aspire-E5-553:~$ ./rr
                ***Welcome to CPU Scheduling***

Enter Number of processes:4
Enter Burst Time of process 1:8
Enter Arrival Time of process 1:0
Enter Burst Time of process 2:4
Enter Arrival Time of process 2:1
Enter Burst Time of process 3:9
Enter Arrival Time of process 3:2
Enter Burst Time of process 4:5
Enter Arrival Time of process 4:3

Enter Time Quantum:     5

Process ID              Burst Time       Turnaround Time        Waiting Time
Process[2]              4                8                      4
Process[4]              5                16                     11
Process[1]              8                22                     14
Process[3]              9                24                     15
Average Waiting Time: 11.000000
Average Turn Around Time:17.500000user@user-Aspire-E5-553:~$
```

Modified Round robin algorithm

The proposed algorithm centers around the change on CPU scheduling algorithmn. The calculation diminishes the waiting time and turnaround time definitely contrasted with the other Scheduling calculation and straightforward Round Robin booking calculation. This proposed calculation works comparably as yet with some adjustment. It executes the most limited activity having least blasted time first rather than FCFS basic Round robin calculation and it additionally utilizes Smart time quantum rather than static time quantum. Rather than giving static time quantum in the CPU scheduling time, our calculation ascertains the Smart time quantum itself as per the burst time all things considered.

Algorithm:

Stage 1: START

Stage 2: Arrange the process in the expanding request of CPU burst time in the ready queue.

Stage 3: Calculate the Mean of the CPU burst time of the total number of Processes

Mean (M)=(B1+B2+B3... ... .Bn)/n;

Stage 4: Set the Smart time quantum (STQ) as

STQ=(Mean+ Highest burst time)/2;

Stage 5: Allocate the CPU to the First process in the Ready Queue the time of 1 Smart Time Quantum.

stage 6: If the rest of the CPU burst time of the present process is less at that point or equivalent to 1 time quantum. Reallocate the CPU to the current process again for the rest of the burst time.
After the total execution of the present process, expel it from the ready queue.Else expel the present process from the ready queue and put it on the tail of the ready queue for further execution.

Stage 7: Pick the following process from the ready queue and give the CPU to it up to the age of 1 Smart time quantum and afterward again to the stage 6.

Stage 8: Process the line until the point when it will be vacant.

Stage 9: Calculate the Average Waiting time and Average Turnaround time of all procedures.

code:

```
 #include <stdio.h>
 #include<curses.h>
 #include <stdio.h>

 #include<math.h>
 #include<string.h>
 int wt[100],bt[100],at[100],tat[100],n,p[100];
 float awt[5],atat[5];
 int temp1,temp2,temp3,sqt,avg;

 void input()
 {
         printf("Enter Number of processes:");
         scanf("%d",&n);
         int i;
  for(i=0;i<n;i++)
   p[i]=i+1;
         for(i=0;i<n;i++)
         {
         printf("Enter Burst Time of process %d:",i+1);
         scanf("%d",&bt[i]);
         printf("Enter Arrival Time of process %d:",i+1);
         scanf("%d",&at[i]);
         }
  for(i=0;i<5;i++)
  {
   awt[i]=0.0;
   atat[i]=0.0;
}
```

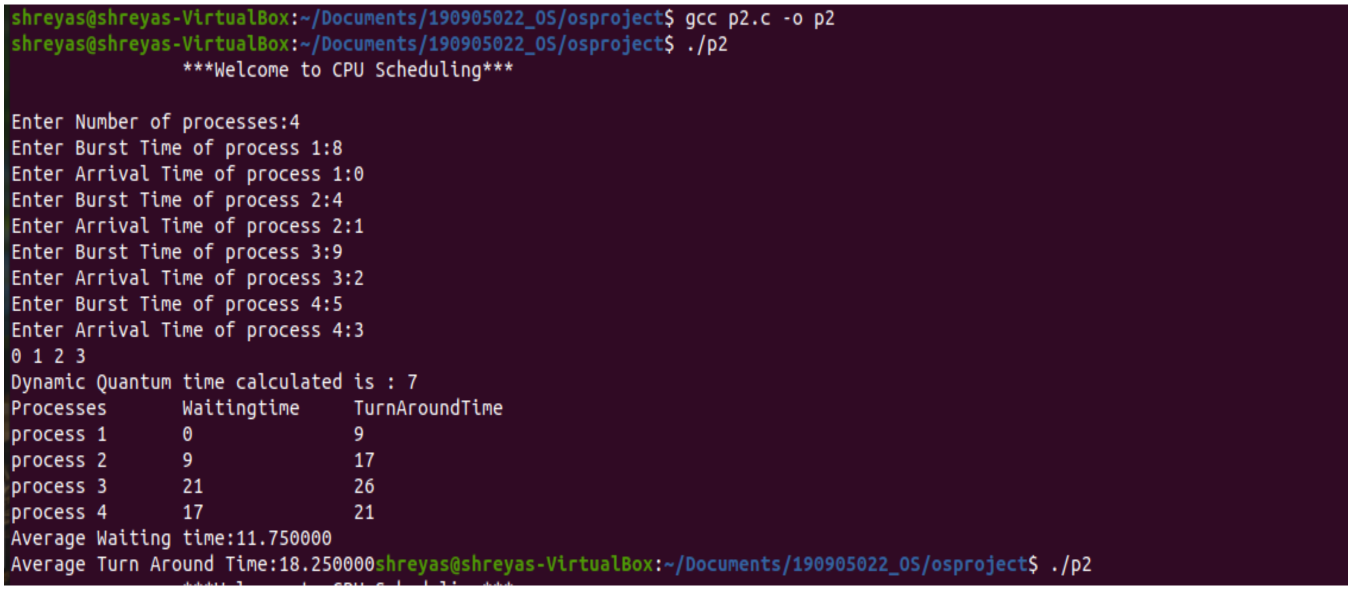
```
}
void changeArrival(){
 int a=at[0];
 int i;
 for(i=0;i<n;i++){
  if(at[i]<a)
        a=at[i];
 }
 if(a!=0){
  for(i=0;i<n;i++)
        at[i]=at[i]-a;
 }
}

void innovative()
{
 int bt1[n],i,j,temp,qt;
 int b[n];
float twt,ttat;
for(i=0;i<n;i++)
 bt1[i]=bt[i];
for(i=0;i<n;i++)
 b[i]=bt[i];
int num=n;
int time=0;
int max;
int sum,t,a,ap;
ap=0;
 //sorting in ascending order
        for (i = 0; i < n; i++)
        {
```

```
        for (j = 0; j < n-i- 1; j++)
        {
                if (bt[j] > bt[j + 1])
                {
                temp1 = bt[j];
                temp2 = p[j];
                temp3 = at[j];
                bt[j] = bt[j + 1];
              p[j] = p[j + 1];
                at[j] = at[j + 1];
                bt[j + 1] = temp1;
              p[j + 1] = temp2;
                at[j + 1] = temp3;
                }
                }
        }
        max=bt[n-1];
        sum=0;
        for(i=0;i<n;i++)
        {
        sum=sum+bt[i];
        }
        avg=sum/n;

        //printf("\n %d  %d \n",max,avg);

        qt=(avg+max)/2;
        printf("\nDynamic Quantum time calculated is : %d\n",qt);

    while(num>0){
    a=0;
```

```
max=0;
sum=0;
t=0;
for(i=0;i<n;i++){



 if(at[i]<=time && b[i]!=0)
 {
        if(b[i]<qt)
        {
        t+=b[i];
        b[i]=0;
        }
        else
        {
        t+=qt;
        b[i]-=qt;
        }
        if(b[i]<qt && b[i]!=0)
        {
        t+=b[i];
        b[i]=0;
        }
        if(b[i]==0){
        wt[i]=(time+t)-bt1[i];
        tat[i]=time+t;
        num--;
        }
 }
}
```

```
 time+=t;
 }
 printf("Processes\tWaitingtime\tTurnAroundTime\n");
 for(j=1;j<=n;j++)
        {
                for(i=0;i<n;i++)
                {
                if(j==p[i])
        printf("process %d\t%d\t\t%d\n",p[i],wt[i],tat[i]);
        }
  }

        twt=0;
        ttat=0;
 for(i=0;i<n;i++)
  {twt=twt+wt[i];}
 awt[4]=twt/n;
 for(i=0;i<n;i++)
  {ttat=ttat+tat[i];}
 atat[4]=(ttat/n);

}
void display(int c)
{
   int i;
   printf("Average Waiting time:%f\nAverage Turn Around Time:%f",awt[c-2],atat[c-2]);
}

int main(){
 char c;
 printf("\t\t***Welcome to CPU Scheduling***\n\n");
```

```
   input();
   changeArrival();
   innovative();
   display(6);
   return 0;
 }
```



```
shreyas@shreyas-VirtualBox:~/Documents/190905022_OS/osproject$ gcc p2.c -o p2
shreyas@shreyas-VirtualBox:~/Documents/190905022_OS/osproject$ ./p2
                ***Welcome to CPU Scheduling***

Enter Number of processes:4
Enter Burst Time of process 1:8
Enter Arrival Time of process 1:0
Enter Burst Time of process 2:4
Enter Arrival Time of process 2:1
Enter Burst Time of process 3:9
Enter Arrival Time of process 3:2
Enter Burst Time of process 4:5
Enter Arrival Time of process 4:3
0 1 2 3
Dynamic Quantum time calculated is : 7
Processes       Waitingtime     TurnAroundTime
process 1       0               9
process 2       9               17
process 3       21              26
process 4       17              21
Average Waiting time:11.750000
Average Turn Around Time:18.250000shreyas@shreyas-VirtualBox:~/Documents/190905022_OS/osproject$ ./p2
```

4.Meaning of some Terms used in the Project:

- **Average Waiting Time (AWT)** - It is the average time a call remains in the queue until an agent answers it.
- **Average Turnaround time(ATT)** – It is the total amount of time spent by the process from coming in the ready state for the first time to its completion.
- **Burst time(BT)** – It is the amount of time required by a process for executing on a CPU. It is also called execution time or running time. Burst time of a process cannot be known in advance before executing the process. It can be known only after the process has been executed.

Contribution from each student:

FCFS algorithm --- Kishan Nayak K

SJF algorithm --- Shyam Vaideeaswaran

Round Robin algorithm --- Jaideep

Efficient Round Robin algorithm --- Shreyas Kamath

Learning outcome:

In this project, we tried to build an efficient algorithm which reduces average waiting time and turnaround time. But unfortunately now we are not in a position to prove that our algorithm is efficient. But we have tried our best. Also we tried to go through and analyse already available algorithms like fcfs,sjf and round robin algorithms. Also arrived that round robin is efficient among all available algorithms.

Also this project helped us to understand the concept of scheduling, waiting time,turnaround time, burst time etc.

Future Scope:

This efficient Round robin scheduling algorithm is very much helpful in saving waiting, turnaround time. Also it helps in memory management along with time management. It is useful when we are dealing with thousands of processes at a time.

References:

- Operating System Concepts - Abraham Silberschatz ,Peter B Galvin ,Gerg Gagne - 9th edition
- https://sersc.org/journals/index.php/IJAST/article/view/20022/10166