Name: Neelotpal Singh Gour

Reg No: 11611919

E-mail id: [1997gour@gmail.com](mailto:1997gour@gmail.com)

Git Hub link: <https://github.com/Nsg19/os>

**Code:**

#include<iostream>

#include<unistd.h>

#include<windows.h>

using namespace std;

int time=0;

class Scheduling

{ public:

int arrival\_time[10],at[10],burst\_time[10],bt[10],waiting\_time[10],cpu\_assign\_time[10];

int process\_completion\_time[10],flag[10],stop[10],process[10];

};

int q[100];

int front=-1,rear=-1;

void push(int i)

{

if(rear==100)

printf("overflow");

rear++;

q[rear]=i;

if(front==-1)

front=0;

}

int pop()

{

if(front==-1)

printf("underflow");

int temp=q[front];

if(front==rear)

front=rear=-1;

else

front++;

return temp;

}

main()

{

system("color f3");

Scheduling a[10];

int i,j,k,n,time\_quantum;

cout<<"\n============================================= ROUND ROBIN SCHEDULING ===================================================\n";

cout<<"\nEnter number of Process:-\n";

cin>>n;

cout<<"\nEnter Time Quantum for all process:\n";

cin>>time\_quantum;

cout<<"\n================================================= ARRIVAL TIME =========================================================\n";

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

{

cout<<"\nEnter Arrival Time for Process-"<<i+1<<":\n";

cin>>a[i].arrival\_time[i];

a[i].process[i]=i;

}

cout<<"\n================================================ CPU BURST TIME ========================================================\n";

int sum\_bt=0;

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

{

cout<<"\nEnter CPU Burst for Process-"<<j+1<<":\n";

cin>>a[j].burst\_time[j];

a[j].bt[j]=a[j].burst\_time[j];

a[j].flag[j]=0;

a[j].stop[j]=0;

sum\_bt+=a[j].burst\_time[j];

}

int temp;

for (i=1;i<n;i++)

{

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

{

if (a[j].arrival\_time[j]>a[j+1].arrival\_time[j+1])

{

temp=a[j].arrival\_time[j];

a[j].arrival\_time[j]=a[j+1].arrival\_time[j+1];

a[j+1].arrival\_time[j+1]=temp;

temp=a[j].burst\_time[j];

a[j].burst\_time[j]=a[j+1].burst\_time[j+1];

a[j+1].burst\_time[j+1]=temp;

temp=a[j].process[j];

a[j].process[j]=a[j+1].process[j+1];

a[j+1].process[j+1]=temp;

}

}

}

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

{

a[i].bt[i]=a[i].burst\_time[i];

}

int max=a[0].bt[0],index=0;

for(i=1;i<n;i++)

{

if(max<a[i].bt[i])

{

max=a[i].bt[i];

index=i;

}

}

if(time\_quantum!=0)

{

cout<<"\n\n================================================= FINAL RESULT =========================================================\n";

cout<<"\nTime Quantum:- "<<time\_quantum<<"\n";

cout<<"Gantt Chart:-\n\n";

cout<<" ";

int count=1;

int jump=1;

int r;

push(0);

while(a[index].burst\_time[index]!=0)

{

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

{

r=pop();

if((a[r].burst\_time[r]<time\_quantum)&&(a[r].burst\_time[r]!=0))

{

if(a[r].flag[r]==0)

{

if(a[r].arrival\_time[r]<=time)

{

a[r].cpu\_assign\_time[r]=time;

}

else

{

a[r].cpu\_assign\_time[r]=a[r].arrival\_time[r];

time=a[r].arrival\_time[r];

}

a[r].flag[r]=1;

}

cout<<"----";

cout<<time;

time=time+a[r].burst\_time[r];

system("color 2B");

cout<<"\_\_\_|P"<<a[r].process[r]+1<<"|\_\_\_";

sleep(a[r].burst\_time[r]);

cout<<time;

a[r].burst\_time[r]=0;

}

if(a[r].burst\_time[r]>=time\_quantum)

{

system("color f3");

if(a[r].flag[r]==0)

{

if(a[r].arrival\_time[r]<=time)

{

a[r].cpu\_assign\_time[r]=time;

}

else

{

a[r].cpu\_assign\_time[r]=a[r].arrival\_time[r];

time=a[r].arrival\_time[r];

}

a[r].flag[r]=1;

}

cout<<"----";

cout<<time;

if(a[r].burst\_time[r]!=0)

{

a[r].burst\_time[r]-=time\_quantum;

time+=time\_quantum;

}

cout<<"\_\_\_|P"<<a[r].process[r]+1<<"|\_\_\_";

sleep(time\_quantum);

cout<<time;

}

if(a[r].burst\_time[r]==0&&a[r].stop[r]==0)

{

system("color 2B");

Beep(600,1500);

a[r].process\_completion\_time[r]=time;

a[r].stop[r]=1;

}

for(int s=jump;s<n;s++)

{

if(a[s].arrival\_time[s]<=time&&a[s].stop[s]!=1)

{

push(s);

count++;

}

if(a[s].arrival\_time[s]>time&&a[r].stop[r]==1)

{

push(s);

count++;

}

}

if(a[r].stop[r]!=1)

{

push(r);

}

jump=count;

}

}

system("color f3");

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

{

a[j].waiting\_time[j]=a[j].process\_completion\_time[j]-a[j].bt[j]-a[j].arrival\_time[j];

}

cout<<"\n\nPROCESS\t"<<" ARRIVAL TIME\t"<<" CPU BURST TIME\t"<<" CPU ASSING TIME\t"<<"COMPLETION TIME\t"<<" WAITING TIME\n";

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

{

cout<<"P"<<a[i].process[i]+1<<"\t\t"<<a[i].arrival\_time[i]<<"\t\t"<<a[i].bt[i]<<"\t\t"<<a[i].cpu\_assign\_time[i]<<"\t\t\t"<<a[i].process\_completion\_time[i]<<"\t\t"<<a[i].waiting\_time[i]<<"\n";

}

}

}

1. Explain the problem in terms of operating system concept?

**Description:**

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

* It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
* One of the most commonly used technique in CPU scheduling as a core.
* It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
* The disadvantage of it is more overhead of context switching

Round robin scheduling is an algorithm mainly used by operating systems and applications that serve multiple clients that request to use resources. It handles all requests in a circular first-in-first-out (FIFO) order and eschews priority so that all processes/applications may be able to use the same resources in the same amount of time and also have the same amount of waiting time each cycle; hence it is also considered as cyclic executive.  
  
It is one of the oldest, simplest, fairest and most widely used scheduling algorithms of all time, partly because it is very easy to implement as there are no complicated timings or priorities to consider, only a FIFO system and a fixed time constraint for each usage of the resource. This also solves the problem of starvation, a problem in which a process is not able to use resources for a long time because it always gets preempted by other processes thought to be more important

2. Write the algorithm for proposed solution of the assigned problem.

**Algorithm:**

1. First declare and initialize the array of burst time, arrival time, and process.

2. Create the two queues for insertion and deletion.

3. Initialize time(global variable)=0

4. Sort the array of burst time, arrival time and process on the basis of arrival time in the increasing order. 5. Insert the process index in the queue and remove from the queue and apply Round Robin Algorithm.

6. Calculate waiting time:

Waiting time= Completion time - Burst time - arrival time.

3. Calculate complexity of implemented algorithm.

**Description:**

The overall complexity of code is n^2.

4. Explain all the test cases applied on the solution of assigned problem**.**

**Description:**

**Test Cases-**

1.)

Time Quantum=2

**Process** **Arrival Time** **Burst Time**

P1 0 6

P2 1 4

P3 4 2

2.)

Time Quantum=5

**Process** **Arrival Time** **Burst Time**

P1 2 6

P2 1 10

P3 4 3

3.)

Time Quantum=4

**Process** **Arrival Time** **Burst Time**

P1 15 1

P2 10 2

P3 1 3

4.)

Time Quantum=5

**Process** **Arrival Time** **Burst Time**

P1 0 10

P2 1 15

P3 2 20

5.)

Time Quantum=10

**Process** **Arrival Time** **Burst Time**

P1 2 20

P2 9 40

P3 1 15

5. Explain the boundary conditions of the implemented code.

**Description:**

1. Max process schedule at a time=20
2. Time quantum should not be zero or negative.
3. Ready queue space is 100.

6. Have you made minimum 5 revisions of solution on GitHub?

**GitHub Link:** <https://github.com/Nsg19/os>