**Student Name: Kaushalendra Singh**

**Student ID: 11808594**

**Email Address:** [**vs102019@gmail.com**](mailto:vs102019@gmail.com)

**Problem :**

Considering 4 processes with the arrival time and the burst time requirement of the processes the scheduler schedules the processes by interrupting the processor after every 3 units of time and does consider the completion of the process in this iteration. The schedulers then checks for the number of processes waiting for the processor and allots the processor to the process but interrupting the processor after every 6 units of time and considers the completion of the process in this iteration. The scheduler after the second iteration checks for the number of processes waiting for the processor and now provides the processor to the process with the least time requirement to go in the terminated state.

The inputs for the number of requirements, arrival time and burst time should be provided by the user.

|  |  |  |
| --- | --- | --- |
| Processes | Arrival Time | Burst |
| P1 | 0 | 18 |
| P2 | 2 | 23 |
| P3 | 4 | 13 |
| P4 | 13 | 10 |

Develop a scheduler which submits the processes to the processor in the above defined scenario, and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time.

**Ans**: I solved this problem with the help of Round Robin Algorithm of Operating System.

**Algorithm:**

1. The queue structure in ready queue is of First In First Out (FIFO) type.

2. A fixed time is allotted to every process that arrives in the queue. This fixed time is known as time slice or time quantum.

3. The first process that arrives is selected and sent to the processor for execution. If it is not able to complete its execution within the time quantum provided, then an interrupt is generated using an automated timer.

4. The process is then stopped and is sent back at the end of the queue. However, the state is saved and context is thereby stored in memory. This helps the process to resume from the point where it was interrupted.

5. The scheduler selects another process from the ready queue and dispatches it to the processor for its execution. It is executed until the time Quantum does not exceed.

6. The same steps are repeated until all the process are finished.

**Algorithm for waiting times of all processes:**

1- Create an array rem\_bt[] to keep track of remaining

burst time of processes. This array is initially a

copy of bt[] (burst times array)

2- Create another array wt[] to store waiting times

of processes. Initialize this array as 0.

3- Initialize time : t = 0

4- Keep traversing the all processes while all processes

are not done. Do following for i'th process if it is

not done yet.

a- If rem\_bt[i] > quantum

(i) t = t + quantum

(ii) bt\_rem[i] -= quantum;

b- Else // Last cycle for this process

(i) t = t + bt\_rem[i];

(ii) wt[i] = t - bt[i]

(ii) bt\_rem[i] = 0; // This process is over

**Code:** written in C Programming language

#include<stdio.h>

#include<conio.h>

void rr(int no,int remt[10],int Cur\_t,int arT[10], int bsT[10]);

main()

{

int Proc\_no,j,no,CurT,RemProc,indicator,time\_quan,wait,tut,arT[10],bsT[10],remt[10],x=1;

indicator = 0;

wait = 0;

tut = 0;

printf("Enter number of processes ");

scanf("%d",&no);

RemProc = no;

printf("\nEnter the arrival time and burst time of the processes\n");

for(Proc\_no = 0;Proc\_no < no;Proc\_no++)

{

printf("\nProcess P%d\n",Proc\_no+1);

printf("Arrival time = ");

scanf("%d",&arT[Proc\_no]);

printf("Burst time = ");

scanf("%d",&bsT[Proc\_no]);

remt[Proc\_no]=bsT[Proc\_no];

}

printf("The details of time quantum are as follows:\n");

printf("The time quantum for first round is 3.\n");

time\_quan=3;

CurT=0;

for(Proc\_no=0;RemProc!=0;)

{

if(remt[Proc\_no]<=time\_quan && remt[Proc\_no]>0)

{

CurT+=remt[Proc\_no];

remt[Proc\_no]=0;

indicator=1;

}

else if(remt[Proc\_no]>0)

{

remt[Proc\_no]-=time\_quan;

CurT+=time\_quan;

}

if(remt[Proc\_no]==0 && indicator==1)

{ printf("%d",Proc\_no);

RemProc--;

printf("P %d",Proc\_no+1);

printf("\t\t\t%d",CurT-arT[Proc\_no]);

printf("\t\t\t%d\n",CurT-bsT[Proc\_no]-arT[Proc\_no]);

wait+=CurT-arT[Proc\_no]-bsT[Proc\_no];

tut+=CurT-arT[Proc\_no];

indicator=0;

}

if(Proc\_no==no-1){

x++;

if(x==2){

Proc\_no=0;

time\_quan=6;

printf("The time quantum for second round is 6. \n");

}

else{

break;

}

}

else if(CurT >= arT[Proc\_no+1]){

Proc\_no++;

}

else{

Proc\_no=0;

}

}

rr(no,remt,CurT,arT,bsT);

return 0;

}

void rr(int no,int remt[10],int Cur\_t,int arT[10], int bsT[10]){

float avg\_wait,avg\_tut;

int i,j,n=no,temp,btime[20],Proc\_no[20],w\_time[20],tut\_t[20],total=0,loc;

printf("Third round with least burst time.\n");

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

{

btime[i]=remt[i];

w\_time[i]=Cur\_t-arT[i]-btime[i];

Proc\_no[i]=i+1;

}

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

{

loc=i;

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

{

if(btime[j]<btime[loc]){

loc=j;

}

}

temp=btime[i];

btime[i]=btime[loc];

btime[loc]=temp;

temp=Proc\_no[i];

Proc\_no[i]=Proc\_no[loc];

Proc\_no[loc]=temp;

}

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

{

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

w\_time[i]+=btime[j];

}

total+=w\_time[i];

}

avg\_wait=(float)total/n;

total=0;

printf("\nProcess\t\tBurst time\t\twaiting time\t\tTurnaround Time");

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

{

tut\_t[i]=btime[i]+w\_time[i];

total=total + tut\_t[i];

printf("\nP%d\t\t\t%d\t\t\t%d\t\t\t%d",Proc\_no[i],btime[i],w\_time[i],tut\_t[i]);

}

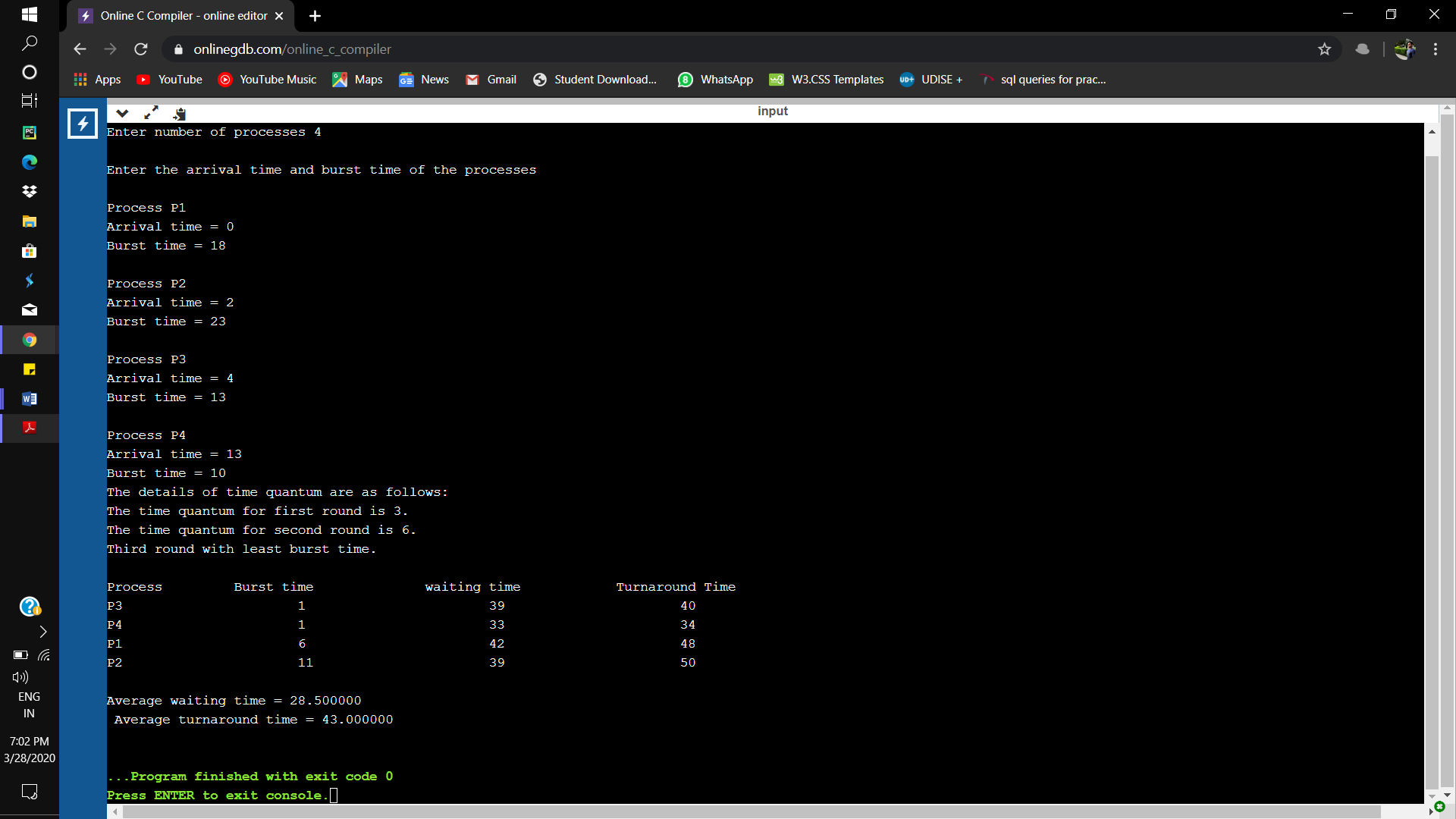
avg\_tut=(float)total/n;

printf("\n\nAverage waiting time = %f",avg\_wait);

printf("\n Average turnaround time = %f\n",avg\_tut);

}

**Output :**



**Problem In Terms of Operating System**

This question consists of Round Robin scheduling, **Round robin scheduling algorithm**is used to schedule process fairly each job a time slot or quantum and the interrupting the job if it is not completed by then the job come after the other job which are arrived in the quantum time that make these scheduling fairly.

Round robin is cyclic in nature so starvation doesn’t occur.

Round robin is variant of first come, first served scheduling.

No priority, special importance given to any process or task.

RR scheduling is also known as Time slicing scheduling.

Here in this question we have two different time Quantum -3 unit, and 6 unit.

|  |  |  |
| --- | --- | --- |
| Processes | Arrival Time | Burst |
| P1 | 0 | 18 |
| P2 | 2 | 23 |
| P3 | 4 | 13 |
| P4 | 13 | 10 |

The details of time quantum are as follows

The time quantum for first round is 3.

The time quantum for first round is 6.

Third round with least burst time.

|  |  |  |  |
| --- | --- | --- | --- |
| Processes | Burst Time | Waiting Time | Turnaround Time |
| P3 | 1 | 39 | 40 |
| P4 | 1 | 33 | 34 |
| P1 | 6 | 42 | 48 |
| P2 | 11 | 39 | 50 |

Average waiting time = 28.500000

Average turnaround time = 43.000000

**Complexity of the algorithm :**

The complexity of the Round Robin algorithm is O(1).

**Constraints**

* Round robin is cyclic in nature so starvation doesn’t occur
* Round robin is variant of first come, first served scheduling
* No priority, special importance given to any process or task
* RR scheduling is also known as Time sSlicing scheduling

**Code Snippet:**

for(Proc\_no=0;RemProc!=0;)

{

if(remt[Proc\_no]<=time\_quan && remt[Proc\_no]>0)

{

CurT+=remt[Proc\_no];

remt[Proc\_no]=0;

indicator=1;

}

else if(remt[Proc\_no]>0)

{

remt[Proc\_no]-=time\_quan;

CurT+=time\_quan;

}

if(remt[Proc\_no]==0 && indicator==1)

{ printf("%d",Proc\_no);

RemProc--;

printf("P %d",Proc\_no+1);

printf("\t\t\t%d",CurT-arT[Proc\_no]);

printf("\t\t\t%d\n",CurT-bsT[Proc\_no]-arT[Proc\_no]);

wait+=CurT-arT[Proc\_no]-bsT[Proc\_no];

tut+=CurT-arT[Proc\_no];

indicator=0;

}

if(Proc\_no==no-1){

x++;

if(x==2){

Proc\_no=0;

time\_quan=6;

printf("The time quantum for second round is 6. \n");

}

else{

break;

}

}

else if(CurT >= arT[Proc\_no+1]){

Proc\_no++;

}

else{

Proc\_no=0;

}

}

**Boundary Condition:**

* Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called time quantum.
* When a process is given the CPU, a timer is set for whatever value has been set for a time quantum.
* If the process finishes its burst before the time quantum timer expires, then it is swapped out of the CPU just like the normal FCFS algorithm.
* If the timer goes off first, then the process is swapped out of the CPU and moved to the back end of the ready queue.
* The ready queue is maintained as a circular queue, so when all processes have had a turn, then the scheduler gives the first process another turn, and so on.
* RR scheduling can give the effect of all processors sharing the CPU equally, although the average wait time can be longer than with other scheduling algorithms.

**Test Cases:**

**Case 1:**

|  |  |  |
| --- | --- | --- |
| Processes | Arrival Time | Burst Time |
| P1 | 0 | 10 |
| P2 | 3 | 18 |
| P3 | 5 | 23 |
| P4 | 16 | 13 |

The details of time quantum are as follows

The time quantum for first round is 3.

The time quantum for first round is 6.

Third round with least burst time.

|  |  |  |  |
| --- | --- | --- | --- |
| Processes | Burst Time | Waiting Time | Turnaround Time |
| P1 | 0 | 43 | 43 |
| P4 | 4 | 34 | 38 |
| P2 | 6 | 31 | 37 |
| P3 | 11 | 33 | 44 |

Average waiting time = 24.500000

Average turnaround time = 40.500000

**Case 2:**

|  |  |  |
| --- | --- | --- |
| Processes | Arrival Time | Burst Time |
| P1 | 0 | 20 |
| P2 | 5 | 36 |
| P3 | 13 | 19 |
| P4 | 26 | 42 |

The details of time quantum are as follows

The time quantum for first round is 6.

The time quantum for first round is 10.

Third round with least burst time.

|  |  |  |  |
| --- | --- | --- | --- |
| Processes | Burst Time | Waiting Time | Turnaround Time |
| P1 | 0 | 74 | 74 |
| P3 | 3 | 55 | 58 |
| P2 | 14 | 61 | 75 |
| P4 | 26 | 39 | 65 |

Average waiting time =38.750000

Average turnaround time =68.000000

**Case 3:**

|  |  |  |
| --- | --- | --- |
| Processes | Arrival Time | Burst |
| P1 | 0 | 18 |
| P2 | 2 | 23 |
| P3 | 4 | 13 |
| P4 | 13 | 10 |

The details of time quantum are as follows

The time quantum for first round is 3.

The time quantum for first round is 6.

Third round with least burst time.

|  |  |  |  |
| --- | --- | --- | --- |
| Processes | Burst Time | Waiting Time | Turnaround Time |
| P3 | 1 | 39 | 40 |
| P4 | 1 | 33 | 34 |
| P1 | 6 | 42 | 48 |
| P2 | 11 | 39 | 50 |

Average waiting time = 28.500000

Average turnaround time = 43.000000