Name: Shrutika Gawas Date: /11/21
Roll no: 191105072 Sem: 05

EXPERIMENT NO. 6

ROUND ROBIN SCHEDULING

AIM: To implement Round Robin scheduling algorithm

THEORY:

Round Robin Scheduling:
Round Robin scheduling algorithm is one of the most popular scheduling algorithms which can actually be implemented in most of the operating systems.
This is the pre-emptive version of first come first serve scheduling.
The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way.
A certain time slice is defined in the system which is called time quantum.
Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

Characteristics of Round Robin Scheduling:
Round robin is a pre-emptive algorithm
The CPU is shifted to the next process after fixed interval time, which is called time quantum/time slice.
The process that is pre-empted is added to the end of the queue.
Round robin is a hybrid model which is clock-driven
Time slice should be minimum, which is assigned for a specific task that needs to be processed. However, it may differ from OS to OS.
It is a real time algorithm which responds to the event within a specific time limit.
Round robin is one of the oldest, fairest, and easiest algorithms.
Widely used scheduling method in traditional OS.

Advantages:
It can be actually implementable in the system because it is not depending on the burst time.
It doesn't suffer from the problem of starvation or convoy effect.
All the jobs get a fare allocation of CPU.

Disadvantages:
The higher the time quantum, the higher the response time in the system.
The lower the time quantum, the higher the context switching overhead in the system.
Deciding a perfect time quantum is really a very difficult task in the system.

Example:

Consider the set of 6 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 3, calculate the average waiting time and average turnaround time.

Ready Queue:

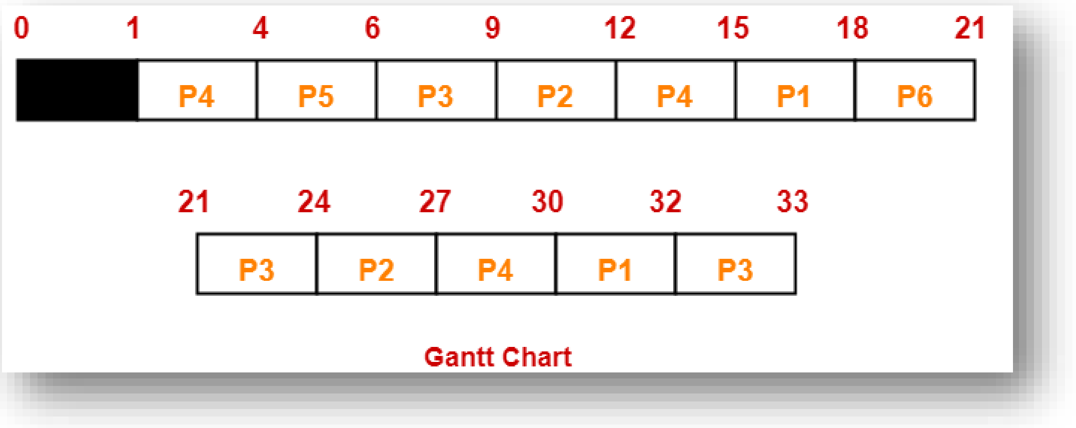
P3, P1, P4, P2, P3, P6, P1, P4, P2, P3, P5, P4

Now, we know,

Turn Around time = Completion time – Arrival time
Waiting time = Turn Around time – Burst time

Process Id	Completion time	Turn Around time	Waiting time
P1	32	32 – 5 = 27	27 – 5 = 22
P2	27	27 – 4 = 23	23 – 6 = 17
P3	33	33 – 3 = 30	30 – 7 = 23
P4	30	30 – 1 = 29	29 – 9 = 20
P5	6	6 – 2 = 4	4 – 2 = 2
P6	21	21 – 6 = 15	15 – 3 = 12

Now,
Average Turn Around time = (27 + 23 + 30 + 29 + 4 + 15) / 6 = 128 / 6 = 21.33 unit
Average waiting time = (22 + 17 + 23 + 20 + 2 + 12) / 6 = 96 / 6 = 16 unit



CODE:

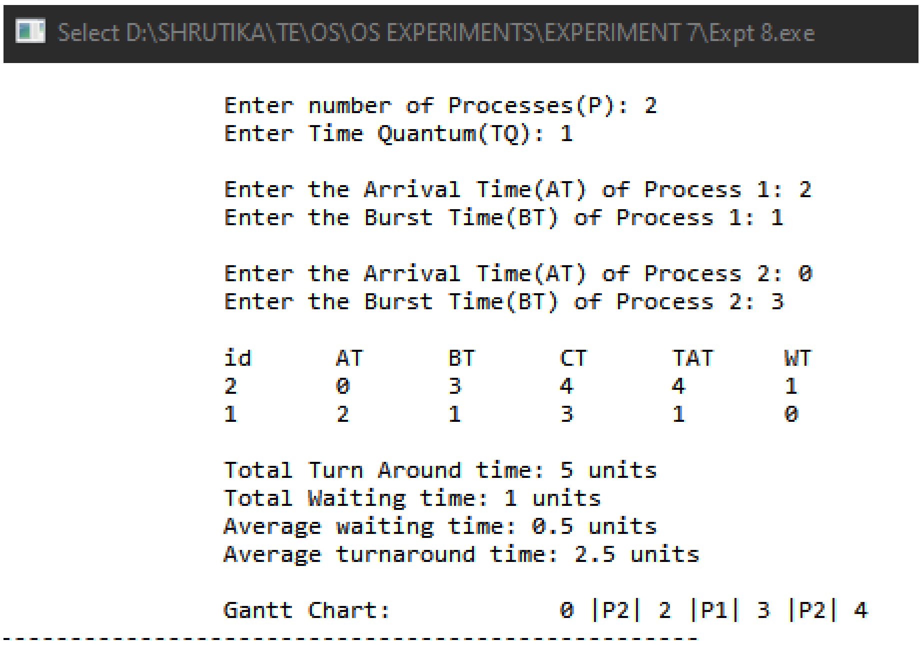
```
//Round Robin Scheduling
#include<bits/stdc++.h>
using namespace std;
struct process{
        int id,
        arrival_time,
        burst_time,
        completion_time,
        turnAround_time,
        waiting_time,
        response_time,
        firstAllocation_time,
        remaining_time;
};
bool compare(process p1,process p2)
{
        return p1.arrival_time<p2.arrival_time;
}
int main()
{
        int n;
        cout<<"\n\t\tEnter number of Processes(P): ";
        cin>>n;
        int tq;
        cout<<"\t\tEnter Time Quantum(TQ): ";
        cin>>tq;
        struct process p[n];
        for(int i=0;i<n;i++)
        {
                p[i].id=i+1;
                cout<<"\n\t\tEnter the Arrival Time(AT) of Process "<<p[i].id<<": ";
                cin>>p[i].arrival_time;
                cout<<"\t\tEnter the Burst Time(BT) of Process "<<p[i].id<<": ";
                cin>>p[i].burst_time;
                p[i].remaining_time = p[i].burst_time;
                p[i].firstAllocation_time=-1;
        }
        sort(p,p+n,compare);
        vector<int> chart(250,-1);
        queue<int> q;
        vector<int> mark(n,0);
        q.push(0);
        mark[0]=1;
        int cur_time=INT_MAX;
        for(int i=0;i<n;i++)
        {
                if(cur_time>p[i].arrival_time)
                        cur_time=p[i].arrival_time;
        }
        int process_completed=0;
        while(process_completed!=n)
        {
                int process=q.front();
                q.pop();
                if(p[process].firstAllocation_time==-1)
                {
                        p[process].firstAllocation_time=cur_time;
           }
                if(p[process].remaining_time-tq>0)
                {
                        for(int i=cur_time;i<(cur_time+tq);i++)
                                chart[i]=p[process].id;
                        p[process].remaining_time-=tq;
                        cur_time+=tq;
                }
                else
                {
                        for(int i=cur_time;i<(cur_time+p[process].remaining_time);i++)
                                chart[i]=p[process].id;
                        cur_time+=p[process].remaining_time;
                        p[process].remaining_time=0;
                        process_completed++;
                        p[process].completion_time=cur_time;
                        p[process].response_time=p[process].firstAllocation_time-
p[process].arrival_time;
                        p[process].turnAround_time=p[process].completion_time-
p[process].arrival_time;
```

```
                    p[process].waiting_time=p[process].turnAround_time-
p[process].burst_time;
            }
            for(int i=1;i<n;i++)
            {
                    if(p[i].remaining_time>0 && p[i].arrival_time<=cur_time &&
mark[i]==0)
                    {
                            q.push(i);
                            mark[i]=1;
                    }
            }
            if(p[process].remaining_time>0)
            {
                    q.push(process);
            }
            if(q.empty())
            {
        for(int i=1;i<n;i++)
                    {
          if(p[i].remaining_time>0)
                            {
            q.push(i);
            break;
          }
        }
      }
        }
        int tat=0,twait=0;
        for(int i=0;i<n;i++)
        {
                tat = tat+p[i].turnAround_time;
                twait +=p[i].waiting_time;
        }
        cout<<"\n\t\t"<<"id \t"<<"AT\t"<<"BT\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<endl;
        for(int i=0;i<n;i++)

        cout<<"\t\t"<<p[i].id<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<
p[i].completion_time<<"\t"<<p[i].turnAround_time<<"\t"<<p[i].waiting_time<<"\n";
```

```
        cout<<"\n\t\tTotal Turn Around time: "<<tat<<" units"<<endl;
        cout<<"\t\tTotal Waiting time: "<<twait<<" units"<<endl;
        cout<<"\t\tAverage waiting time: "<<float(twait)/n<<" units"<<endl;
        cout<<"\t\tAverage turnaround time: "<<float(tat)/n<<" units"<<endl;
        int prev = 0;
        cout<<"\n\t\tGantt Chart: \t\t";
        for(int i=0;i<cur_time;i++)
        {
                if(chart[i]!=-1 && prev!=chart[i])
                {
                        cout<<i<<" |"<<"P"<<chart[i]<<"| ";
                        prev = chart[i];
                }
        }
        cout<<cur_time<<endl;
        cout<<"-------------------------------------------------"<<endl;
        return 0;
}
```

OUTPUTS:



```
Select D:\SHRUTIKA\TE\OS\OS EXPERIMENTS\EXPERIMENT 7\Expt 8.exe

                Enter number of Processes(P): 2
                Enter Time Quantum(TQ): 1

                Enter the Arrival Time(AT) of Process 1: 2
                Enter the Burst Time(BT) of Process 1: 1

                Enter the Arrival Time(AT) of Process 2: 0
                Enter the Burst Time(BT) of Process 2: 3

                id      AT      BT      CT      TAT     WT
                2       0       3       4       4       1
                1       2       1       3       1       0

                Total Turn Around time: 5 units
                Total Waiting time: 1 units
                Average waiting time: 0.5 units
                Average turnaround time: 2.5 units

                Gantt Chart:            0 |P2| 2 |P1| 3 |P2| 4
--------------------------------------------------
```

CONCLUSION: The code for PRE-EMPTIVE PRIORITY SCHEDULING was implemented and executed successfully.