DSA Lab
Name: Siddhi Parekh
Reg No.:221071047
Batch: C
SY Comps

Experiment No.: 5

AIM:

Design and implement a round-robin scheduling algorithm for a set of processors with variable execution times utilizing a circular linked list to represent the processors, and it simulates the execution of processes based on a predefined quantum time.

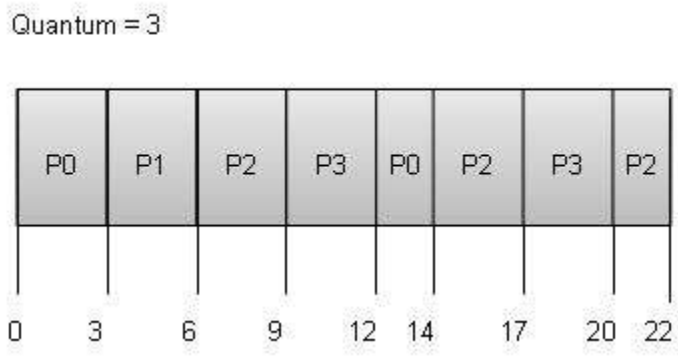
THEORY:

The Round-Robin scheduling algorithm is one of the simplest and widely used algorithms, especially in time-sharing systems. The program implemented here simulates a basic round-robin scheduling scenario using a circular linked list data structure**.**

The processors are represented using a circular linked list, where each node of the list corresponds to a processor. The circular nature ensures that the scheduling algorithm loops back to the beginning after reaching the end, creating a continuous cycle.

Each node of the linked list, represented by the node structure, contains information about a processor, including its ID, execution time, and a pointer to the next processor in the list. Additionally, each processor has an associated quantum time, representing the fixed time quantum for the round-robin scheduling.

EXAMPLE:



The time provided by the architecture is 3s in each iteration for the processor to run. The time when all its tasks are executed, the processor is no longer taken into consideration.

In the above diagram, processor p0 requires 5s , p1 requires 3s, p2 requires 8s and p3 requires 6s, respectively to complete all their tasks.

In the first iteration, i.e up to the block 4, all take 3s to execute.

For second iteration:
Time remaining(RT) for each processor to execute it's task:

1. po = 2s.
2. p1 = 0s.
3. p2 = 5s.
4. p3 = 3s.

   For the processors with RT less than quantum time, execute for RTs and ones with 0s, stop execution, and are no longer displayed.

So, with the help of a circular linked list execution time gets reduced and Round-Robin scheduling allows each processor to execute once in every iteration, till all of them are completed with their required execution(i.e their RT becomes 0s).

ALGORITHM:

Step 0: Start
Step 1: Create a node of linked list with the help of structure data type which contains the value of node and address of the next node.
Step 2: Declare the quantum time of architecture and create a pointer that points to the start of the linked list.
Step 3: Make a function of creating a node.
Step 4: In the create function, the insertion takes place in a sorted way, i.e the linked list gets arranged in ascending order by the end of this function.
Step 5: Make a function of deleting a node.
Step 6: In the deletion function, the node gets deleted for next iteration if it is done with executing all its task, or time remaining to execute the task is less than the quantum time.
Step 7: Make a function of executing the round-robin scheduling algorithm.
Step 8: In this execute function, the processors get executed according to the time remaining (call deletion function here)
Step 9: Inside int main, call the create and execute function.
Step 10: Stop

CONCLUSION:

This program provides a practical implementation of the round-robin scheduling algorithm, demonstrating its fairness in allocating CPU time to a set of processors. The simulation results can be used to evaluate the efficiency and fairness of the algorithm under different scenarios, making it a valuable tool for understanding and experimenting with CPU scheduling in time-sharing systems.

CODE:

```
#include<iostream>
using namespace std;

struct node
{
	string id;
	int data;
	struct node *next;
};

int exetime=4;
struct node *start;


void create(int n)
{
	string s;
	int ptime;
	node*p, *q;
	start = NULL;

	for(int i=1; i<=n; i++)
	{
	p=(node*)malloc(sizeof(node));
	cout<<"Enter the name of processor "<<i<<" and its execution time"<<endl;
```

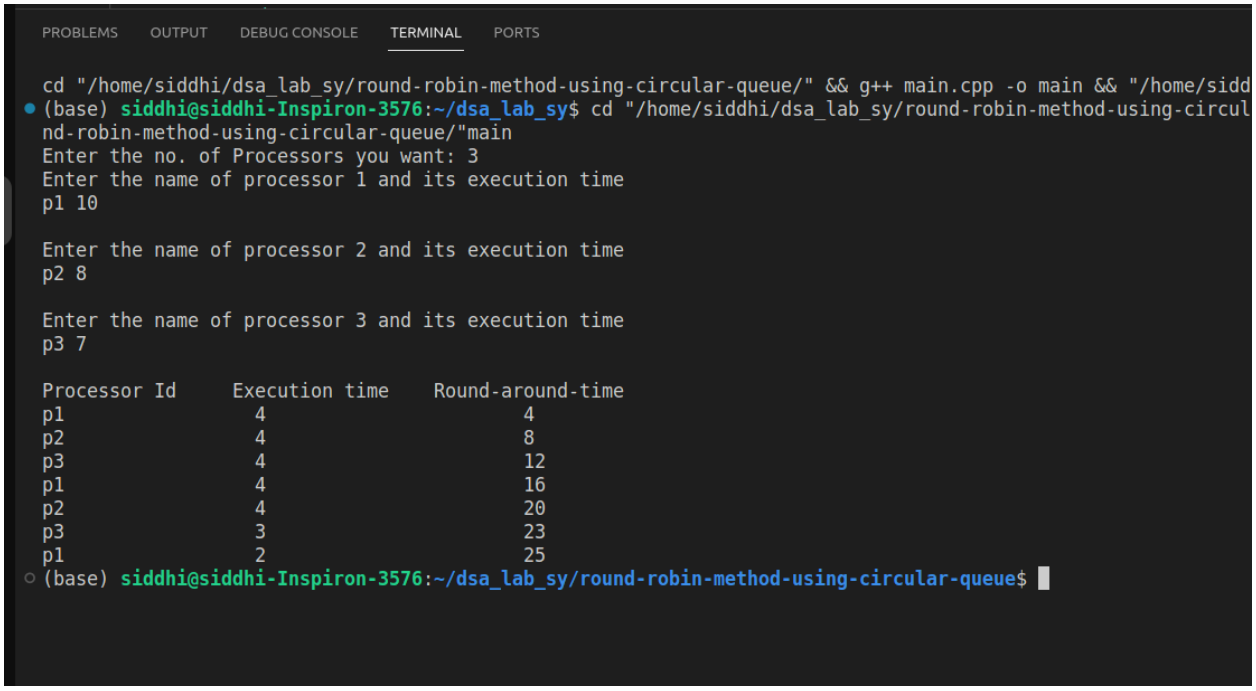
```
        cin>>s>>ptime;
        p->id=s;
        p->data=ptime;

        if(start==NULL)
        {

        p->next=p;
        start=p;
        }

        else
        {
        q=start;
        while(q->next!=start)
        {
                q=q->next;
        }

        q->next=p;
        p->next=start;
        }

        cout<<endl;
        }
}
void del(node *p)
{
        node *q;
        node *r = start;
        q=p;
        if(q->next==start)
        {
        while(r->next!=q)
        {
        r=r->next;
```

```
        }

        r->next=start;
        free(q);
        }

        else if(q==start)
        {
        while(r->next!=start)
        {
        r=r->next;
        }

        start=q->next;
        r->next=start;
        free(q);
        }

        else
        {
        while(r->next !=q)
        {
        r=r->next;
        }

        r->next=q->next;
        free(q);
        }

}

void execute()
{
        node *p,*r;
        p=start;
        int round_time=0;
```

```
	cout<<"Processor Id	Execution time	Round-around-time"<<endl;
while(start!=NULL)
	{
	if(p->data>exetime)
	{
	p->data=p->data-exetime;
	round_time=round_time+exetime;
	cout<<p->id<<"		"<<exetime<<"		"<<round_time<<endl;
	p=p->next;
	}

	else if(p->data == exetime)
	{
	r=p;
	p->data =0;
	round_time=round_time+exetime;
	cout<<p->id<<"		"<<exetime<<"		"<<round_time<<endl;
	del(r);
	if(p->next==p) {
		start=NULL;
		break;
	}
	else p=p->next;
	}

	else if(p->data<exetime)
	{
	r=p;
	round_time=round_time+p->data;
	cout<<p->id<<"		"<<p->data<<"		"<<round_time<<endl;
	p->data=0;
	del(r);
	if(p->next==p) {
		start=NULL;
		break;
	}
```

```
        else p=p->next;
        }


        }

}

int main()
{
        int n=0;
        cout<<"Enter the no. of Processors you want: ";
        cin>>n;

        create(n);
        execute();
}
```

OUTPUT:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
cd "/home/siddhi/dsa_lab_sy/round-robin-method-using-circular-queue/" && g++ main.cpp -o main && "/home/sidd
(base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy$ cd "/home/siddhi/dsa_lab_sy/round-robin-method-using-circul
nd-robin-method-using-circular-queue/"main
Enter the no. of Processors you want: 3
Enter the name of processor 1 and its execution time
p1 10
Enter the name of processor 2 and its execution time
p2 8
Enter the name of processor 3 and its execution time
p3 7
Processor Id Execution time Round-around-time
p1 4 4
p2 4 8
p3 4 12
p1 4 16
p2 4 20
p3 3 23
p1 2 25
(base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy/round-robin-method-using-circular-queue$