BANGALORE INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belgaum)

VV Puram, Bangalore - 560004

A MINI-PROJECT REPORT

ON

"ROUND ROBIN BASED CPU SCHEDULER"

Submitted by

PRITHVI K P	1BI19CS113
ROHAN M	1BI19CS128
LITHISH S	1BI19CS079
KOUSHIK G G	1BI19CS078
SHASHANK M J	1BI19CS080

B TECH

Semester III

Department of Computer & Science Engineering

Bangalore Institute of Technology

FEB 2021

BANGALORE INSTITUTE OF TECHNOLOGY

VV PURAM, BANGALORE – 560004

Department of Computer & Science Engineering

Certified that the mini-project work entitled **"ROUND ROBIN BASED CPU SCHEDULER"** is a bonafide work carried out by

PRITHVI K P	**1BI19CS113**
ROHAN M	**1BI19CS128**
LITHISH S	**1BI19CS079**
KOUSHIK G G	**1BI19CS078**
SHASHANK M J	**1BI19CS080**

The report has been approved as it satisfies the academic requirements in respect of mini-project work prescribed for the course.

.....	.....
Mrs. Shruthiba.A	**Dr. Asha.T**
Mini-Project Coordinator	Head of the Department

ACKNOWLEDGEMENT

I extend our sincere and heartfelt thanks to our respected coordinator, **Shruthiba.A** and for her exemplary guidance, monitoring and constant encouragement throughout the course at crucial junctures and for showing us the right way and also for permitting me to utilize all the necessary facilities of the Institute.

I would like to extend thanks to our respected Head of the department, **Dr. Asha.T** for allowing us to use the facilities available. We would like to thank other faculty members also

Last but not the least,

I would like to thank our friends and family for the support and encouragement they have given us during the course of our work.

Prithvi K P

ABSTRACT

In this project we are going to implement..“Round-robin scheduling algorithm”. This is one of the simplest scheduling algorithm for processes in an operating system, which assigns time slice to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive).

Round-robin scheduling is both simple and easy to implement and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. Round robin is most efficiently used method in operating system.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

1.1. CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair.

1.2. Operating System Schedulers

The scheduler is an operating system module that selects the next jobs to be admitted into the system and the next process to run. Operating systems may feature up to three distinct scheduler types: a **long-term scheduler**, a **mid-term scheduler**, and a **short-term scheduler**. The names suggest the relative frequency with which their functions are performed.

Long-Term Scheduling

Long-term scheduling involves selecting the processes from the storage pool in the secondary memory and loading them into the ready queue in the main memory for execution. This is handled by the long-term scheduler or job scheduler.

Short-Term Scheduling

Short-term scheduling involves selecting one of the processes from the ready queue and scheduling them for execution. This is done by the short-term scheduler. A scheduling algorithm is used to decide which process will be scheduled for execution next by the short-term scheduler.

A diagram that demonstrates scheduling using long-term and short-term schedulers is given as follows:

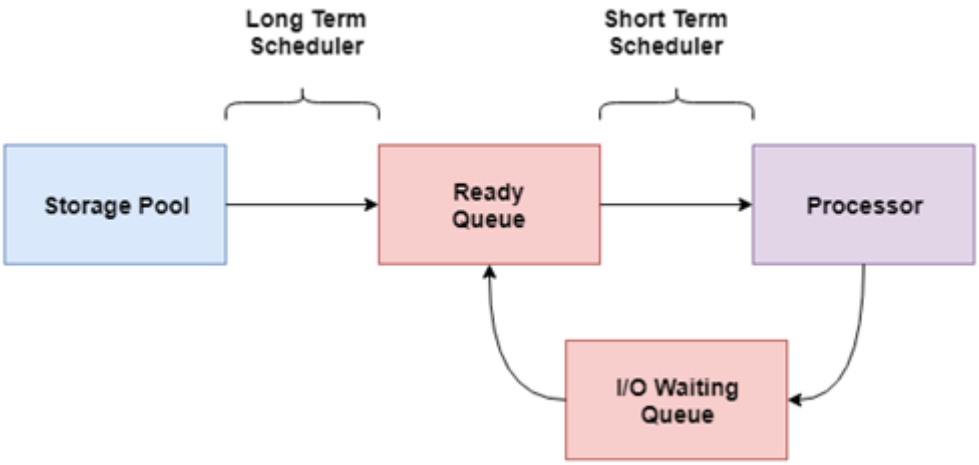

Fig.1.1. Short term and Long term Scheduler using a Queuing

Medium-Term Scheduling

Medium-term scheduling involves swapping out a process from main memory. The process can be swapped in later from the point it stopped executing. This can also be called as suspending and resuming the process and is done by the medium-term scheduler.

A diagram that demonstrates medium-term scheduling is given as follows :

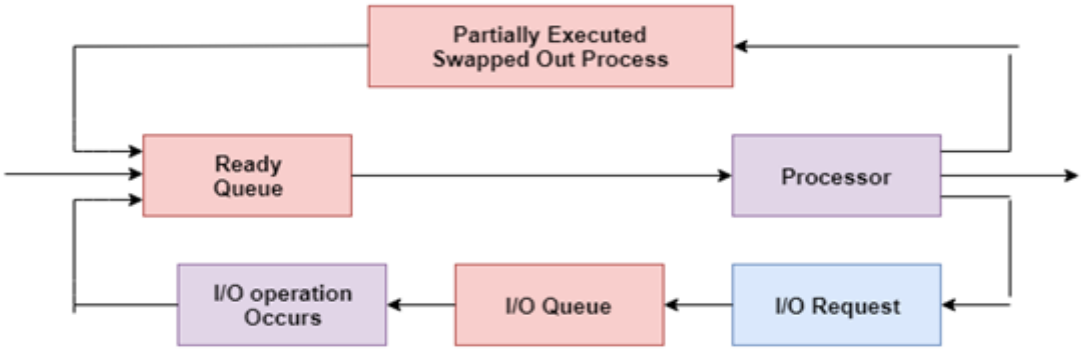

Fig.1.2. Mid Term Scheduler using a Queuing

1.3. Dispatcher

The **Dispatcher** is the module that gives control of the CPU to the process selected by the **short-term scheduler**. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program from where it left last time.

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time taken by the dispatcher to stop one process and start another process is known as the **Dispatch Latency**. Dispatch Latency can be explained using the below figure:

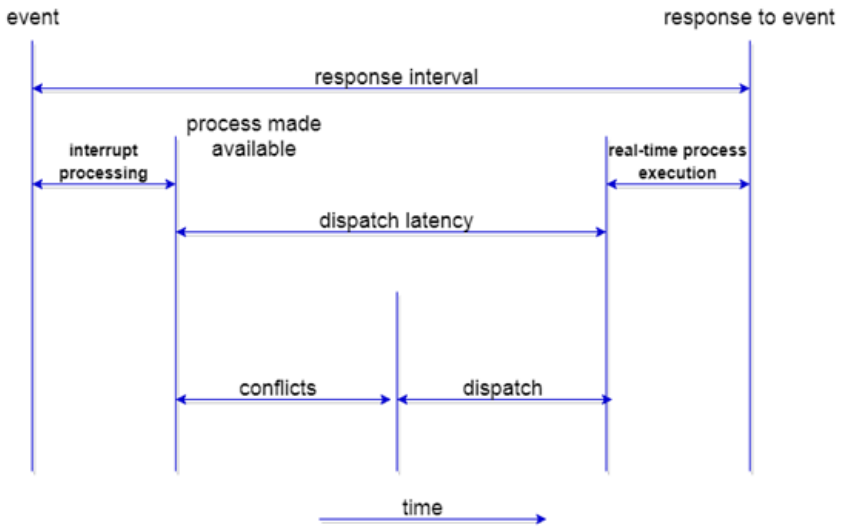

Fig.1.3. Dispatch Latency

1.4. Types of CPU Scheduling

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the **running** state to the **waiting** state (for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the **running** state to the **ready** state (for example, when an interrupt occurs).
3. When a process switches from the **waiting** state to the **ready** state (for example, completion of I/O).
4. When a process **terminates**.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3.

When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is **non-preemptive**; otherwise the scheduling scheme is **preemptive**.

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

1.5. Scheduling Algorithms

To decide which process to execute first and which process to execute last to achieve maximum CPU utilisation, computer scientists have defined some algorithms, they are:

1. First Come First Serve (FCFS) Scheduling
2. Shortest Job First (SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling

2. Round Robin Algorithm

The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turns. It is the oldest, simplest scheduling algorithm, which is mostly used for multitasking. In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

2.1. Characteristics

- 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 preempted 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 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 algorithm.
- Widely used scheduling method in traditional OS.

2.2. Scheduling Criteria

There are many different criterias to check when considering the best scheduling algorithm, they are:

CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time(Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Arrival Time

Time at which the process arrives in the ready queue.

Completion Time

It is the time at which process completes its execution.

Burst Time

Burst Time is the time required by a process for CPU execution.

TurnAround Time

The time difference between completion time and arrival time

Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU

Time Quantum

A special kind of an input is given with a specific time slice which is fixed based on which processes are terminated.

2.3. Example for Round Robin:

1. A process may block itself before its time slice expires.
2. If it uses its entire time slice, it is then preempted and put at the end of the ready queue.
3. The ready queue is managed as a FIFO queue and treated as a circular.
4. If there are **n** processes on the ready queue and the time quantum is **q,** then each process gets **1/n** time on the CPU in chunks of at most **q** time units.
5. No process waits for more than **(n-1)q** time units.
6. The choice of how big to make the time slice **(q)** is extremely important.
   - if **q** is very large, Round Robin degenerates into FCFS.
   - if **q** is very small, the context switch overhead defeats the benefits.

Example 1 (q = 20)

Process	P1	P2	P3	P4
Burst Time	53	17	68	24

Fig.2.1. Example 1

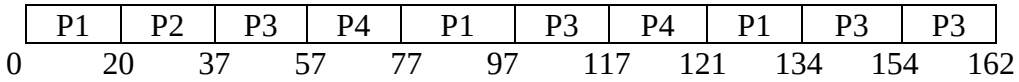

Fig.2.2. Gantt Chart for Example 1

Process	Burst Time	Turnaround Time	Wait Time
P1	53	134	81
P2	17	37	20
P3	68	162	94
P4	24	121	97

Fig.2.3. Turnaround and Wait Time for Example 1

- average turnaround time: (134+37+162+121)/4=113.5
- waiting times:
- P_1: 134-53 = 81
- P_2: 37-17 = 20
- P_3: 162-68= 94
- P_4: 121-24 = 97
- average waiting time: (81+20+94+97)/4 = 73

Example 2 (q = 4)

Process	P1	P2	P3
Burst Time	24	3	3

Fig.2.4. Example 2

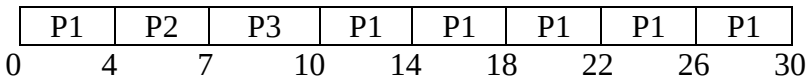

Fig.2.5. Gantt Chart for Example 2

Process	Burst Time	Turnaround Time	Wait Time
P1	24	30	6
P2	3	7	4
P3	3	10	7

Fig.2.6. Turnaround and Wait Time for Example 2

- average turnaround time: (30+7+10)/3 = 15.67
- waiting times:
- P_1: 30-24 = 6
- P_2: 7-3 = 4
- P_3: 10-3= 7
- average waiting time: (6+4+7)/3 = 5.67

Example 3 (q = 10)

Process	P1	P2	P3	P4	P5
Burst Time	10	29	3	7	12

Fig.2.7. Example 3

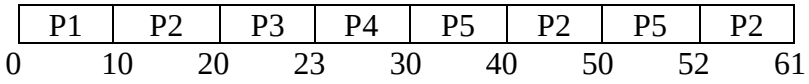

Fig.2.8. Gantt Chart for Example 3

Process	Burst Time	Turnaround Time	Wait Time
P1	10	10	0
P2	29	61	32
P3	3	23	20
P4	7	30	23
P5	12	52	40

Fig.2.9. Turnaround and Wait Time for Example 3

- average turnaround time: (10+61+23+30+52)/5 = 35.2
- waiting times:
- P_1: 10-10 = 0
- P_2: 61-29 = 32
- P_3: 23-3= 20
- P4: 30-7 =23
- P5: 52-12 = 40
- average waiting time: (0+32+20+23+40)/5 = 23

2.4. C Program for Round Robin

```
#include<stdio.h>
#define MAX 200
int main()
{
	int i, limit, total = 0, x, counter = 0, time_quantum;
	int wait_time = 0, turnaround_time = 0, burst_time[MAX], temp[MAX];
	float average_wait_time, average_turnaround_time;
	printf("Enter Total Number of Processes:");
	scanf("%d", &limit);
	x = limit;
	for(i = 0; i < limit; i++)		//enter burst time for each process
	{
		printf("Enter Burst Time[%d]: ", i + 1);
		scanf("%d", &burst_time[i]);
		temp[i] = burst_time[i];//copy each burst time to temporary variable
	}
	printf("Enter Time Quantum: ");
	scanf("%d", &time_quantum);
	printf("\nProcess ID\tBurst Time\tTurnaround Time\t\tWaiting Time\n");
	total=0;
	i=0;
	while(x!=0)
	{
		if(temp[i] <= time_quantum && temp[i] > 0)		//iterate through each
		{							// process when burst time is less
			total = total + temp[i];			// than time quantum
			temp[i] = 0;
			x--;
			printf("\nProcess[%d]\t%d\t\t%d\t\t\t%d",i+1,
			burst_time[i], total, total - burst_time[i]);
			wait_time = wait_time + total - burst_time[i];
			turnaround_time = turnaround_time + total;
		}
		else if(temp[i] > 0)
		{
			temp[i] = temp[i] - time_quantum;
			total = total + time_quantum;
		}
		if(i == limit - 1)		// when reached the final process of each cycle
			i = 0;
		else
			i++;
```

```
        }
        average_wait_time = wait_time * 1.0 / limit;
        average_turnaround_time = turnaround_time * 1.0 / limit;
        printf("\n\nAverage Waiting Time:%.2f", average_wait_time);
        printf("\nAverage Turnaround Time:%.2f\n", average_turnaround_time);
        return 0;
}
```

3. Conclusion and Result

3.1. Output

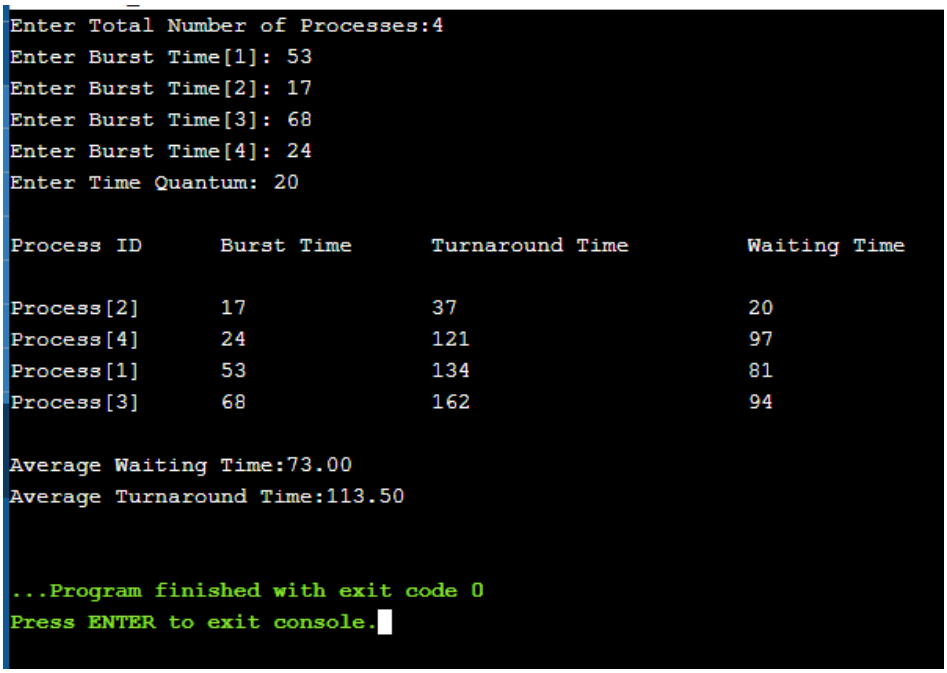

Fig.3.1. Output for Example 1

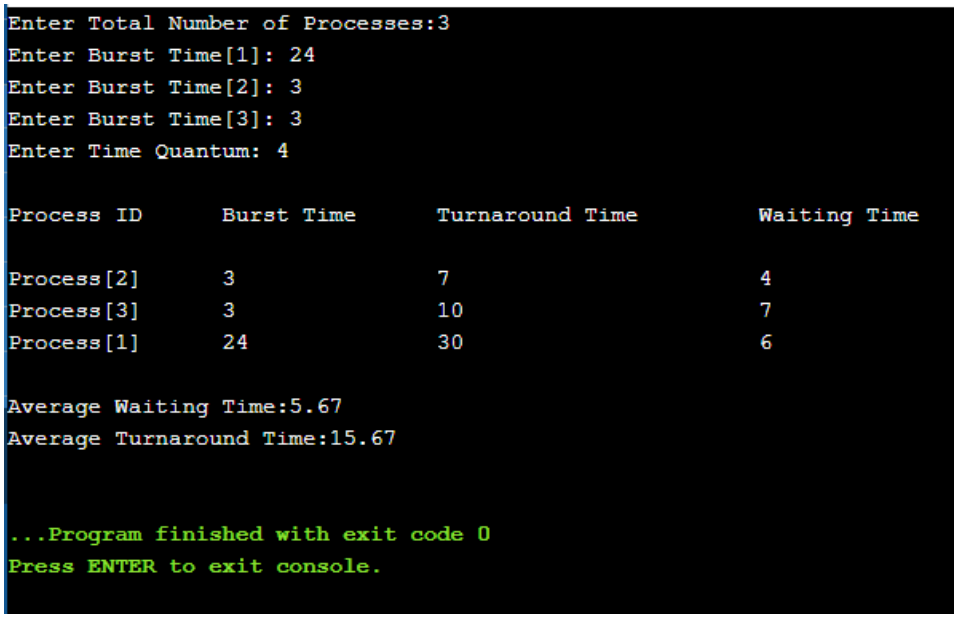

Fig.3.2. Output for Example 2

```
Enter Total Number of Processes:5
Enter Burst Time[1]: 10
Enter Burst Time[2]: 29
Enter Burst Time[3]: 3
Enter Burst Time[4]: 7
Enter Burst Time[5]: 12
Enter Time Quantum: 10

Process ID      Burst Time      Turnaround Time         Waiting Time

Process[1]      10              10                      0
Process[3]      3               23                      20
Process[4]      7               30                      23
Process[5]      12              52                      40
Process[2]      29              61                      32

Average Waiting Time:23.00
Average Turnaround Time:35.20


...Program finished with exit code 0
Press ENTER to exit console.
```

Fig.3.3. Output for Example 3

3.2. Advantages

1. It does not face any starvation issues or convoy effect.
2. Each process gets equal priority to the fair allocation of CPU.
3. It is easy to implement the CPU Scheduling algorithm.
4. Each new process is added to the end of the ready queue as the next process's arrival time is reached.
5. Each process is executed in circular order that shares a fixed time slot or quantum.
6. Every process gets an opportunity in the round-robin scheduling algorithm to reschedule after a given quantum period.

3.3. Disadvantage

1. If the time quantum is lower, it takes more time on context switching between the processes.
2. It does not provide any special priority to execute the most important process.
3. The waiting time of a large process is higher due to the short time slot.
4. The performance of the algorithm depends on the time quantum.
5. The response time of the process is higher due to large slices to time quantum.
6. Getting a correct time slot or quantum is quite difficult for all processes in the round-robin algorithm.

3.4. Performance

Overall performance depends on

Size of the time quantum

1. If time quantum is large than the CPU burst then this algorithm become same as FCFS and thus performance degrade.
2. If the time quantum size is very small, then the number of content switches increases and the time quantum almost equal the time taken to switch the CPU from one process to another. Therefore 50% of time spent in switching of processes.

Number of context switching

The number of context switches should not be too many to slow down the overall execution of all the processes. Time quantum should be large with respect to the context switch time. This is to ensure that a process keeps CPU for a maximum time as compared to the time spent in the context switching.

3.5. Applications and Future Scope

This algorithm can be implemented to improve the performance of CPU in time sharing systems.

Round-robin scheduling can be applied to other scheduling problems, such as data packet scheduling in computer networks.

Further, this work can be extended by executing this strategy in a real distributed cloud computing in balancing the load at the earliest as response time gets reduced effectively under various parameters and conditions and measure the system performance in genuine.

4. References

A. Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts (7th Edn., John Wiley and Sons Inc, 2005, ISBN 0-471-69466-5).

https://www.javatpoint.com/round-robin-program-in-c

https://en.m.wikipedia.org/wiki/Round-robin_scheduling