



**L**OVELY  
**P**ROFESSIONAL  
**U**NIVERSITY

---

*Transforming Education Transforming India*

School Of Computer Science and Engineering

Project Report On

**Simulation Of Round Robin Scheduling Algorithm In  
Operating System**

**CSE316**

**Submitted By: -**

Aditya Mehra

Reg : 12101319

Roll : 46

Section : K21WB

**Submitted To: -**

Richa Sharma

Assistant Prof.

LPU

## CONTENTS

○ Introduction	03-11
1. Objective of the project	
2. Description of the project	
3. Scope of the project	
○ System Description	11-21
1. Scenario Designed	
2. Source Code	
3. Output	
○ Design	22-25
1. System design	
2. E-R Diagram	
3. DFD's	
○ Conclusion	26-26

## **Introduction:**

In computer science, scheduling algorithms are an essential aspect of operating systems that determine how processes are allocated CPU time. One such algorithm is the Round Robin scheduling algorithm, which is commonly used in time-sharing systems.

The Round Robin algorithm is a pre-emptive scheduling technique where each process is assigned a fixed time slice or quantum. The CPU switches between processes in a cyclic order, allowing each process to execute for a fixed amount of time before switching to the next process in the queue. If a process completes its time slice before it finishes its execution, the CPU interrupts the process and places it back in the queue. This process continues until all the processes in the queue are completed.

Round Robin scheduling is beneficial as it provides fairness among all the processes as each process is allocated a fixed amount of time. Additionally, it ensures that the CPU is not monopolized by any one process, which can improve system responsiveness. It also supports multitasking and allows for the concurrent execution of multiple processes.

However, Round Robin scheduling can be inefficient for long-running processes that require a significant amount of CPU time, as they must wait for their turn in the queue. In such cases, a priority-based algorithm might be a better option. Nonetheless, the Round Robin algorithm remains a popular choice for time-sharing systems and real-time applications.

## **Characteristics:**

The Round Robin scheduling algorithm is a widely used process scheduling technique in modern operating systems. The following are some of the essential characteristics of the Round Robin algorithm:

1. **Preemptive:** The Round Robin algorithm is a Preemptive scheduling algorithm, meaning that the operating system can interrupt a process while it is running to give another process a chance to execute.
2. **Time-Sliced:** The Round Robin algorithm slices the available CPU time into fixed time intervals called quantum or time slice. Each process gets a predefined time slice to execute before the operating system switches to the next process in the queue.
3. **Fairness:** The Round Robin algorithm is a fair scheduling algorithm as it ensures that each process gets an equal amount of CPU time, regardless of its priority or execution time. This feature makes it an ideal algorithm for time-sharing systems.
4. **Concurrency:** Round Robin scheduling supports concurrency by allowing multiple processes to run concurrently on the same CPU. This means that the operating system can run multiple processes at the same time by allocating a fixed amount of CPU time to each process.
5. **Overhead:** The Round Robin algorithm incurs a small overhead due to the context switch that occurs when the operating system switches between processes. The overhead can be minimized by selecting an appropriate quantum size.

6. **Response Time:** The Round Robin algorithm provides a quick response time to user requests as each process gets a chance to execute in a short time slice. This feature makes it suitable for real-time applications.
7. **Predictability:** The Round Robin algorithm is predictable as it guarantees that each process gets a fixed amount of CPU time. This predictability makes it easy to estimate the time required to execute a process and plan system resources accordingly.

### **Example**

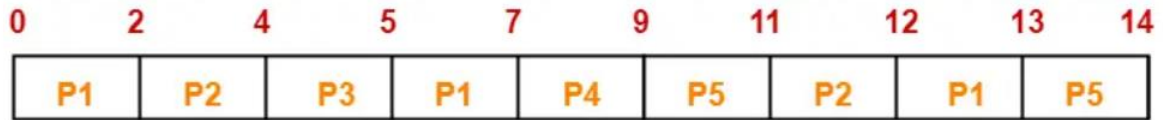
Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Considering Time Quantum = 2 ms.

### Ready Queue-

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



**Gantt Chart**

Process Id	Exit time	Turn Around time	Waiting time
P1	13	$13 - 0 = 13$	$13 - 5 = 8$
P2	12	$12 - 1 = 11$	$11 - 3 = 8$
P3	5	$5 - 2 = 3$	$3 - 1 = 2$
P4	9	$9 - 3 = 6$	$6 - 2 = 4$
P5	14	$14 - 4 = 10$	$10 - 3 = 7$

Now,

Average Turn Around time =  $(13 + 11 + 3 + 6 + 10) / 5 = 43 / 5 = 8.6$  unit

Average waiting time =  $(8 + 8 + 2 + 4 + 7) / 5 = 29 / 5 = 5.8$  unit

### Objective:

- Develop a simulation program that can emulate the Round Robin scheduling algorithm and its associated characteristics.

- Create a graphical user interface (GUI) for the simulation program to allow users to interact with the simulation and adjust its parameters.
- Test the simulation program using a range of inputs, including different process arrival times, varying time quantum sizes, and different process execution times.
- Evaluate the performance of the Round Robin scheduling algorithm under various scenarios, such as high and low CPU utilization, different numbers of processes, and different quantum sizes.
- Analyse the simulation results to identify any potential issues or bottlenecks with the Round Robin algorithm and suggest ways to optimize its performance.
- Compare the performance of the Round Robin scheduling algorithm with other scheduling algorithms, such as First-Come-First-Serve (FCFS) and Shortest Job First (SJF).
- Document the simulation program's design, implementation, and testing process, including any challenges faced and lessons learned.

## **Advantages Of Round Robin Scheduling Algorithm**

The Round Robin scheduling algorithm has several advantages that make it a popular choice for process scheduling in modern operating

systems. The following are some of the primary advantages of the Round Robin algorithm:

- Fairness: The Round Robin algorithm is a fair scheduling algorithm as it ensures that each process gets an equal amount of CPU time, regardless of its priority or execution time. This feature makes it an ideal algorithm for time-sharing systems.
- Concurrency: Round Robin scheduling supports concurrency by allowing multiple processes to run concurrently on the same CPU. This means that the operating system can run multiple processes at the same time by allocating a fixed amount of CPU time to each process.
- Responsiveness: The Round Robin algorithm provides a quick response time to user requests as each process gets a chance to execute in a short time slice. This feature makes it suitable for real-time applications.
- Predictability: The Round Robin algorithm is predictable as it guarantees that each process gets a fixed amount of CPU time. This predictability makes it easy to estimate the time required to execute a process and plan system resources accordingly.
- Good average waiting time: The Round Robin algorithm has a low average waiting time, particularly for short processes. This feature makes it suitable for systems with a high number of short processes.



- Time-sharing support: The Round Robin algorithm supports time-sharing systems by allowing multiple users to share the same CPU resources. Each user gets a fair share of CPU time to execute their processes.
- Low starvation rate: The Round Robin algorithm has a low starvation rate as each process gets a chance to execute in a fixed time slice. This feature ensures that no process is left waiting indefinitely, even if other processes are constantly arriving.

Overall, the Round Robin scheduling algorithm is an efficient and fair process scheduling algorithm that can support time-sharing systems, real-time applications, and concurrent processing. Its predictability, low average waiting time, and low starvation rate make it a popular choice for modern operating systems.

## **Description**

The "Round Robin Algorithm Simulation" project aims to design and implement a software program that simulates the Round Robin scheduling algorithm used in modern operating systems. The project will test the algorithm's behaviour and performance under different scenarios, evaluate its strengths and weaknesses, and compare it to other scheduling algorithms. The project will also create a user-friendly graphical interface for interacting with the simulation.

## **Implement the Round Robin scheduling algorithm in the simulation program:**

To implement the Round Robin scheduling algorithm, you will need to simulate the scheduling of processes using a time quantum for each round.

Here are the steps to follow:

Define a function to simulate the scheduling of processes using the Round Robin algorithm. Create a queue to hold the processes that have arrived but have not yet been scheduled. Initialize a clock variable to 0 to keep track of the time.

While the queue is not empty or there are still processes running, do the following:

- a. Pop the first process from the queue.
- b. If the process has not finished yet, schedule it for the time quantum.
- c. If the process has finished, calculate its wait time and turnaround time and record them.
- d. Add any new processes that have arrived during this time to the queue.
- e. Increment the clock by the time quantum. After all processes have been scheduled, calculate the average wait time and turnaround time for each process.

### **Scope Of The Project:**

The scope of the "Round Robin Algorithm Simulation" project include the following:

- Developing a simulation program that can emulate the Round Robin scheduling algorithm and its associated characteristics.
- Creating a user-friendly graphical interface (GUI) for the simulation program to allow users to interact with the simulation and adjust its parameters.

- Testing the simulation program using various inputs, such as different process arrival times, varying time quantum sizes, and different process execution times.
- Evaluating the performance of the Round Robin scheduling algorithm under different scenarios, such as high and low CPU utilization, different numbers of processes, and different quantum sizes.
- Comparing the performance of the Round Robin scheduling algorithm with other scheduling algorithms, such as First-Come-First-Serve (FCFS) and Shortest Job First (SJF).
- Analysing the simulation results to identify any potential issues or bottlenecks with the Round Robin algorithm and suggest ways to optimize its performance.
- Documenting the simulation program's design, implementation, and testing process, including any challenges faced and lessons learned.

Overall, the scope of the project is to provide a comprehensive understanding of the Round Robin scheduling algorithm's behaviour and performance under different scenarios, and how it can be optimized for various use cases.

## **System Description**

### **Round Robin Scheduling Algorithm: -**

The Round Robin scheduling algorithm is a pre-emptive scheduling algorithm that is widely used in operating systems. It is a time-sharing algorithm that allocates CPU time to each process in a cyclic order, called a "time slice" or "quantum". The time slice is typically small, in the range of 10 to 100 milliseconds. If a process has not completed its CPU burst within the time slice, it is pre-empted and moved to the end of the queue. The algorithm continues until all processes have completed their CPU burst.

### **Random Number Generation: -**

In order to generate a set of processes with random arrival times and CPU burst times, we can use a random number generator. A random number generator is a program that generates a sequence of numbers that appear to be random. In our simulation program, we will use a pseudo-random number generator, which is a deterministic algorithm that produces a sequence of numbers that are statistically random. We can use the random number generator to generate arrival times and CPU burst times that are uniformly distributed between a minimum and maximum value.

### **Performance Metrics: -**

In order to evaluate the performance of the Round Robin scheduling algorithm, we will use two performance metrics: waiting time and turnaround time. The waiting time is the amount of time a process spends waiting in the queue before it can start executing. The turnaround time is the total amount of time a process spends in the system, from arrival to completion. These metrics give us an idea of how long each process has to wait for CPU time and how long it takes for each process to complete its work.

### **Simulation Program: -**

The simulation program will consist of several components: process generation, Round Robin scheduling algorithm, and performance measurement. We will generate a set of processes with random arrival times and CPU burst times using a random number generator. We will

implement the Round Robin scheduling algorithm to allocate CPU time to each process in a cyclic order. We will measure the waiting time and turnaround time for each process and calculate the average waiting time and turnaround time for the entire set of processes. We will compare these metrics with the ideal scenario of a perfect scheduler, where the waiting time and turnaround time for each process would be minimal.

### **Scenario Designed: -**

In this scenario, you are working as a computer systems engineer in a large technology company, and your manager has assigned you a task to create a simulation program that tests the performance of the Round Robin scheduling algorithm. The simulation program needs to generate a set of processes that have random arrival times and CPU burst times. The Round Robin algorithm needs to run for a set amount of time, for example, 100 times units, and record the average waiting time and turnaround time for each process. To accomplish this task, you need to design and implement a simulation program using a programming language of your choice. The program should generate a set of processes randomly with different arrival times and CPU burst times using a random number generator. Then, you need to implement the Round Robin scheduling algorithm in the simulation program. The Round Robin algorithm is a CPU scheduling algorithm that assigns a fixed time quantum to each process in a queue. When the time quantum expires, the process is pre-empted and added to the end of the queue. Finally, the program needs to record the average waiting time and turnaround time for each process and compare the results with the ideal scenario of a perfect scheduler. A perfect scheduler would allocate CPU time to each process in a way that minimizes the waiting time and turnaround time. The simulation program should show whether Round Robin scheduling algorithm is effective or not in minimizing the waiting time and turnaround time.

## Source Code Block-By-Block Explanation

```
private static final int MAX_PROCESSES = 15;  
private static final int TIME_QUANTUM = 8;
```

These lines declare two private static final integer variables: MAX\_PROCESSES, which represents the maximum number of processes that can be created, and TIME\_QUANTUM, which represents the fixed time allotted for each process.

```
Random rand = new Random();  
int n = rand.nextInt(MAX_PROCESSES) + 1;  
Process[] pro = new Process[n];  
CreateProcess(pro, n);  
RoundRobinAlgo(pro, n, 100);  
Display(pro, n);
```

These lines generate a random number n between 1 and MAX\_PROCESSES (inclusive), create an array process of n Process objects, initialize the process objects using the CreateProcess method, run the RoundRobin algorithm on the processes for a maximum of 100 times units, and then display the results using the Display method.

```
for (int i = 0; i < n; i++) {  
    pro[i] = new Process();  
    pro[i].pid = i + 1;  
    pro[i].ArrivalTime = new Random().nextInt(100);  
    pro[i].BurstTime = new Random().nextInt(50) + 1;  
    pro[i].RemainingTime = pro[i].BurstTime;  
    pro[i].WaitTime = 0;  
    pro[i].TurnAroundTime = 0;  
}
```

This loop initializes each process object in the array by assigning it a unique ID (pid), a random arrival time between 0 and 99, a random burst time between 1 and 50, and setting its remaining time, wait time, and turnaround time to 0.

```
public static void RoundRobinAlgo(Process[] pro, int n, int limit) {
```

This line declares a public static method called RoundRobin Algo, which takes an array of Process objects, an integer n, and a limit as input.

```
int current_time = 0;  
int completed_processes = 0;  
int current_process = 0;  
int time_slice = TIME_QUANTUM;
```

These lines declare and initialize several integer variables used in the Round Robin algorithm. current\_time represents the current time in the simulation, completed\_processes represents the number of completed processes, current\_process represents the index of the currently executing process in the process array, and time\_slice represents the remaining time allotted to the current process.

```
while (completed_processes < n && current_time < limit) {
```

This line starts a while loop that continues executing the Round Robin algorithm until either all processes are completed or the time limit is reached.

```
if (pro[current_process].RemainingTime > 0) {
```

This if condition checks if the remaining time of the current process is greater than zero, which means the process still has work left to do.

```
if (pro[current_process].RemainingTime <= time_slice) {
```

This if condition checks if the remaining time of the current process is less than or equal to the time slice, which means the process can be completed within this time slice.

```
current_time += pro[current_process].RemainingTime;
time_slice -= pro[current_process].RemainingTime;
pro[current_process].RemainingTime = 0;
completed_processes++;
pro[current_process].TurnAroundTime = Math.abs(current_time - pro[current_process].ArrivalTime);
pro[current_process].WaitTime = Math.abs(pro[current_process].TurnAroundTime - pro[current_process].BurstTime);
```

If the above condition is true, then the process can be completed within the time slice, so the remaining time of the process is set to zero, the current time is updated, the time slice is reduced by the remaining time of the process, and the completed process count is incremented. The turnaround time and wait time of the process are also calculated using the current time, arrival time, and burst time of the process.

```
pro[current_process].WaitTime = Math.abs(pro[current_process].TurnAroundTime - pro[current_process].BurstTime);
} else {
    current_time += time_slice;
    pro[current_process].RemainingTime -= time_slice;
    time_slice = TIME_QUANTUM;
}
```

If the above if condition is false, then the remaining time of the process is greater than the time slice, which means the process needs more time to complete.

In this case, the time slice is used up by the process, so the current time is updated by the time slice, the remaining time of the process is



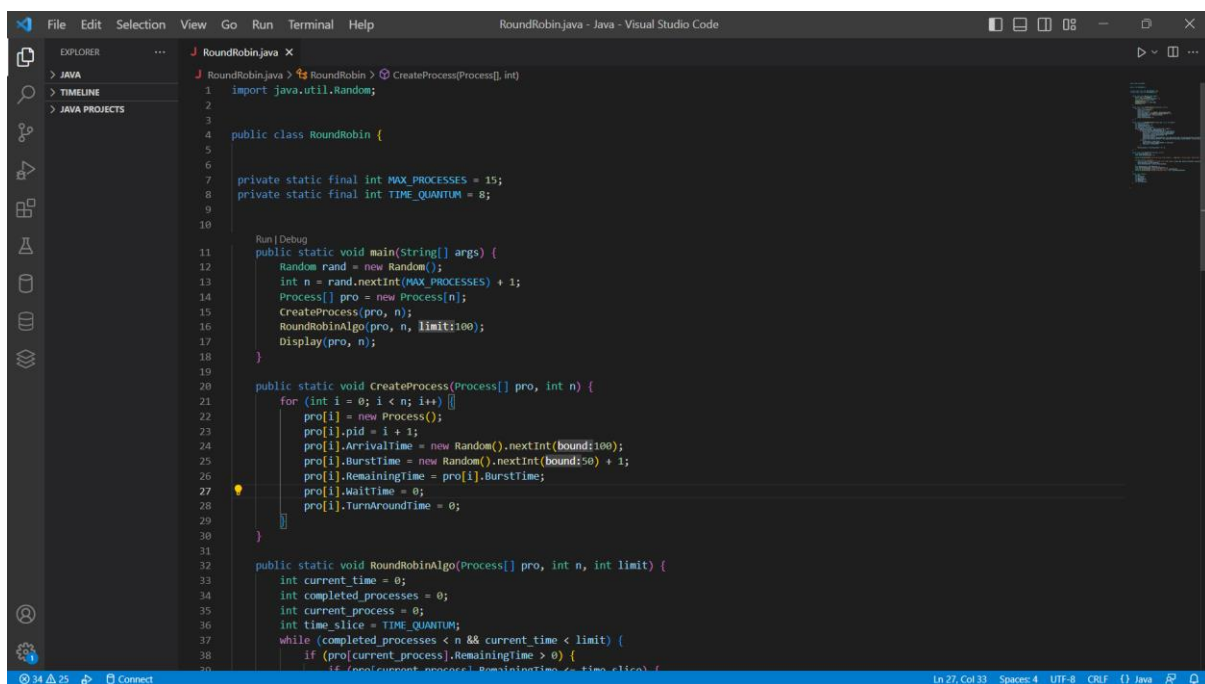
reduced by the time slice, and the time slice is reset to the default time quantum.

```
}  
current_process = (current_process + 1) % n;
```

This line increments the current process counter and wraps around to the beginning if it exceeds the number of processes.

Finally, the while loop continues until all processes are completed or the time limit is reached.

## Complete Source Code



```
RoundRobin.java - Java - Visual Studio Code  
File Edit Selection View Go Run Terminal Help  
EXPLORER  
> JAVA  
> TIMELINE  
> JAVA PROJECTS  
RoundRobin.java X  
1 import java.util.Random;  
2  
3  
4 public class RoundRobin {  
5  
6  
7     private static final int MAX_PROCESSES = 15;  
8     private static final int TIME_QUANTUM = 8;  
9  
10  
11     public static void main(String[] args) {  
12         Random rand = new Random();  
13         int n = rand.nextInt(MAX_PROCESSES) + 1;  
14         Process[] pro = new Process[n];  
15         CreateProcess(pro, n);  
16         RoundRobinAlgo(pro, n, limit:100);  
17         Display(pro, n);  
18     }  
19  
20     public static void CreateProcess(Process[] pro, int n) {  
21         for (int i = 0; i < n; i++) {  
22             pro[i] = new Process();  
23             pro[i].pid = i + 1;  
24             pro[i].ArrivalTime = new Random().nextInt(bound:100);  
25             pro[i].BurstTime = new Random().nextInt(bound:50) + 1;  
26             pro[i].RemainingTime = pro[i].BurstTime;  
27             pro[i].WaitTime = 0;  
28             pro[i].TurnAroundTime = 0;  
29         }  
30     }  
31  
32     public static void RoundRobinAlgo(Process[] pro, int n, int limit) {  
33         int current_time = 0;  
34         int completed_processes = 0;  
35         int current_process = 0;  
36         int time_slice = TIME_QUANTUM;  
37         while (completed_processes < n && current_time < limit) {  
38             if (pro[current_process].RemainingTime > 0) {  
39                 if (pro[current_process].RemainingTime < time_slice) {  
40                     pro[current_process].RemainingTime = 0;  
41                     completed_processes++;  
42                     current_process = (current_process + 1) % n;  
43                     time_slice = TIME_QUANTUM;  
44                 } else {  
45                     pro[current_process].RemainingTime -= time_slice;  
46                     current_time += time_slice;  
47                     current_process = (current_process + 1) % n;  
48                 }  
49             }  
50         }  
51     }  
52 }  
Run | Debug  
Ln 27, Col 33 Spaces: 4 UTF-8 CRLF (1) Java
```

```
File Edit Selection View Go Run Terminal Help RoundRobin.java - Java - Visual Studio Code
EXPLORER
> JAVA
> TIMELINE
> JAVA PROJECTS
J RoundRobin.java X
J RoundRobin.java > RoundRobin > CreateProcess(Process[] int)
37 while (completed_processes < n && current_time < limit) {
38     if (pro[current_process].RemainingTime > 0) {
39         if (pro[current_process].RemainingTime <= time_slice) {
40             current_time += pro[current_process].RemainingTime;
41             time_slice -= pro[current_process].RemainingTime;
42             pro[current_process].RemainingTime = 0;
43             completed_processes++;
44             pro[current_process].TurnAroundTime = Math.abs(current_time - pro[current_process].ArrivalTime);
45             pro[current_process].WaitTime = Math.abs(pro[current_process].TurnAroundTime - pro[current_process].BurstTime);
46         } else {
47             current_time += time_slice;
48             pro[current_process].RemainingTime -= time_slice;
49             time_slice = TIME_QUANTUM;
50         }
51     }
52     current_process = (current_process + 1) % n;
53 }
54 }
55
56 public static void Display(Process[] pro, int n) {
57     float total_waitTime = 0;
58     float total_TurnAroundTime = 0;
59
60     System.out.printf(format: "%-4s %-12s %-11s %-10s %-16s\n", "...args:", "PID", "Arrival Time", "Burst Time", "Wait Time", "Turnaround Time");
61
62     for (int i = 0; i < n; i++) {
63         System.out.printf(format: "%-4d %-12d %-11d %-10d %-16d\n", pro[i].pid, pro[i].ArrivalTime, pro[i].BurstTime, pro[i].WaitTime, pro[i].TurnAroundTime);
64         total_waitTime += pro[i].WaitTime;
65         total_TurnAroundTime += pro[i].TurnAroundTime;
66     }
67     float avg_waitTime = total_waitTime / n;
68     float avg_TurnAroundTime = total_TurnAroundTime / n;
69     System.out.printf(format: "Average wait time: %.2f\n", avg_waitTime);
70     System.out.printf(format: "Average turnaround time: %.2f\n", avg_TurnAroundTime);
71 }
72
73 static class Process {
74     int pid;
75     int ArrivalTime;
76     int BurstTime;
```

```
76     int BurstTime;
77
78     for (int i = 0; i < n; i++) {
79         System.out.printf(format: "%-4d %-12d %-11d %-10d %-16d\n", pro[i].pid, pro[i].ArrivalTime, pro[i].BurstTime, pro[i].WaitTime, pro[i].TurnAroundTime);
80         total_waitTime += pro[i].WaitTime;
81         total_TurnAroundTime += pro[i].TurnAroundTime;
82     }
83     float avg_waitTime = total_waitTime / n;
84     float avg_TurnAroundTime = total_TurnAroundTime / n;
85     System.out.printf(format: "Average wait time: %.2f\n", avg_waitTime);
86     System.out.printf(format: "Average turnaround time: %.2f\n", avg_TurnAroundTime);
87 }
88
89 static class Process {
90     int pid;
91     int ArrivalTime;
92     int BurstTime;
93     int RemainingTime;
94     int WaitTime;
95     int TurnAroundTime;
96 }
97
98 }
```

## Output:

### Case 1

```
PS C:\Users\mail4\OneDrive\Desktop\Java> & 'C:\Program Files\Java\jdk-19\bin\java.exe' '--enableassertions -cp 'C:\Users\mail4\AppData\Roaming\Code\User\workspaceStorage\2c2eaf84b954650bc1e49ed4b64a1b11\redhat.java\jdt_ws\Java_6c
PID Arrival Time Burst Time Wait Time Turnaround Time
1 61 36 0 0
2 62 20 0 0
3 95 14 0 0
4 93 37 0 0
5 30 35 0 0
6 77 32 0 0
7 47 48 0 0
8 67 19 0 0
9 54 7 0 0
10 72 36 0 0
11 11 20 0 0
12 56 45 0 0
13 64 16 0 0
14 69 36 0 0
15 35 48 0 0
16 90 7 0 0
17 29 14 0 0
Average wait time: 0.00
Average turnaround time: 0.00
PS C:\Users\mail4\OneDrive\Desktop\Java>
```

### Case 2

```
Average turnaround time: 0.00
PS C:\Users\mail4\OneDrive\Desktop\Java> c::; cd 'c:\Users\mail4\OneDrive\Desktop\Java' & 'C:\Program Files\Java\jdk-19\bin\java.exe' '--enableassertions -cp 'C:\Users\mail4\AppData\Roaming\Code\User\workspaceStorage\2c2eaf84b954650bc1e49ed4b64a1b11\redhat.java\jdt_ws\Java_6c
ExceptionMessages' '-cp' 'C:\Users\mail4\AppData\Roaming\Code\User\workspaceStorage\2c2eaf84b954650bc1e49ed4b64a1b11\redhat.java\jdt_ws\Java_6c
PID Arrival Time Burst Time Wait Time Turnaround Time
1 80 26 0 0
2 93 13 0 0
3 95 41 0 0
4 38 42 0 0
5 35 23 0 0
6 33 38 0 0
7 4 4 26 30
8 90 48 0 0
9 71 22 0 0
10 89 24 0 0
11 56 17 0 0
12 9 39 0 0
13 15 24 0 0
14 14 24 0 0
15 19 49 0 0
16 36 39 0 0
17 50 14 0 0
Average wait time: 1.53
Average turnaround time: 1.76
PS C:\Users\mail4\OneDrive\Desktop\Java>
```

## Case 3

```
ExceptionMessages' '-cp' 'C:\Users\mail4\AppData\Roaming\Code\User\workspaceStorage\2c2eaf84b95465
PID Arrival Time Burst Time Wait Time Turnaround Time
1 37 40 0 0
2 59 5 44 49
3 40 48 0 0
4 42 9 0 0
5 72 24 0 0
6 47 21 0 0
7 59 21 0 0
8 17 18 0 0
9 1 16 0 0
10 79 34 0 0
11 54 24 0 0
12 82 46 0 0
13 78 13 0 0
14 66 35 0 0
15 81 17 0 0
16 87 24 0 0
17 73 14 0 0
18 15 39 0 0
19 77 23 0 0
Average wait time: 2.32
Average turnaround time: 2.58
PS C:\Users\mail4\OneDrive\Desktop\Java>
```

## Case 4

```
Average turnaround time: 2.58
PS C:\Users\mail4\OneDrive\Desktop\Java> c:; cd 'c:\Users\mail4\OneDrive\Desktop\Java'; & 'C:\Progr
ExceptionMessages' '-cp' 'C:\Users\mail4\AppData\Roaming\Code\User\workspaceStorage\2c2eaf84b954650b
PID Arrival Time Burst Time Wait Time Turnaround Time
1 89 27 0 0
2 23 4 10 14
3 61 18 0 0
4 14 34 0 0
5 23 42 0 0
6 75 39 0 0
7 90 11 0 0
8 4 20 0 0
9 33 1 2 3
10 44 2 4 6
11 9 39 0 0
12 9 43 0 0
13 0 41 0 0
14 47 12 0 0
15 51 13 0 0
16 39 31 0 0
17 50 41 0 0
Average wait time: 0.94
Average turnaround time: 1.35
PS C:\Users\mail4\OneDrive\Desktop\Java>
```

### **Round Robin Scheduler: -**

Maintain a ready queue for all the processes that have arrived.

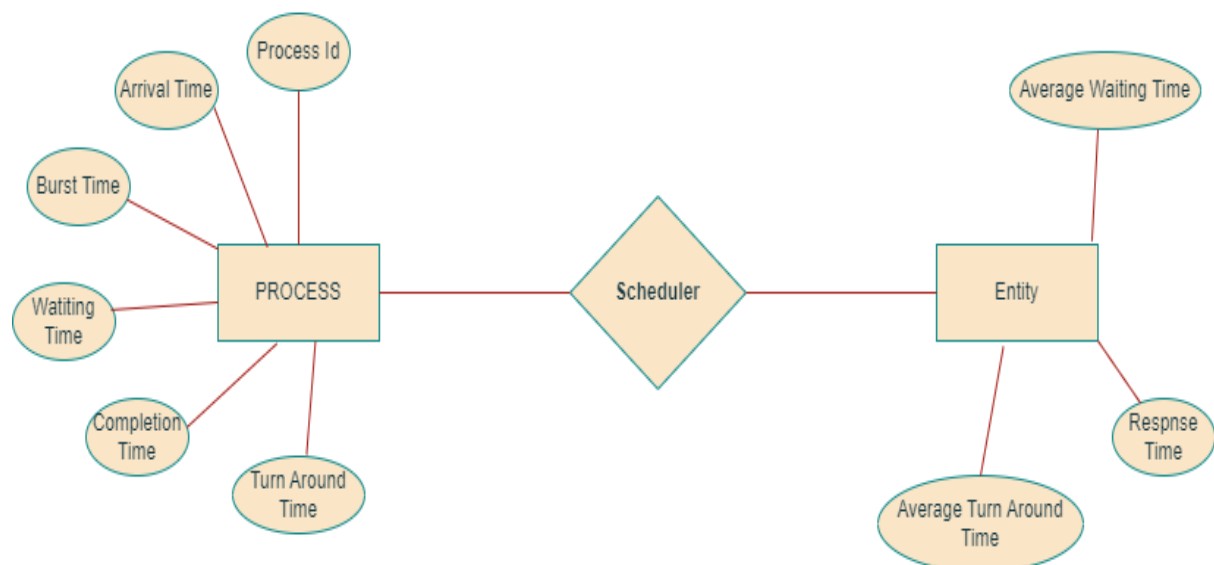
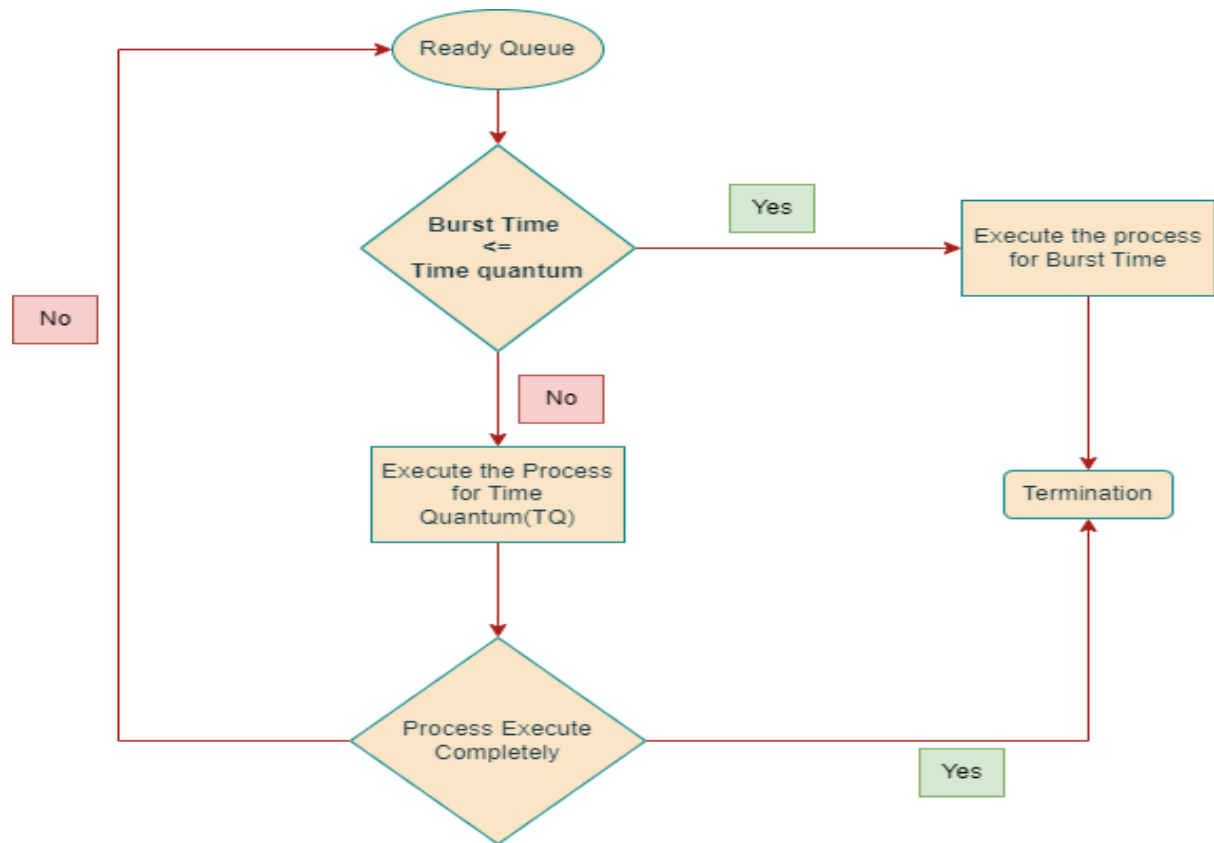
Allocate the CPU to the processes in a round-robin fashion, with a fixed time quantum (e.g., 5 time units). If a process completes its CPU burst before the time quantum expires, remove it from the queue and calculate its waiting time and turnaround time. If a process does not complete its CPU burst before the time quantum expires, move it to the end of the ready queue and continue with the next process.

### **Performance Measurement: -**

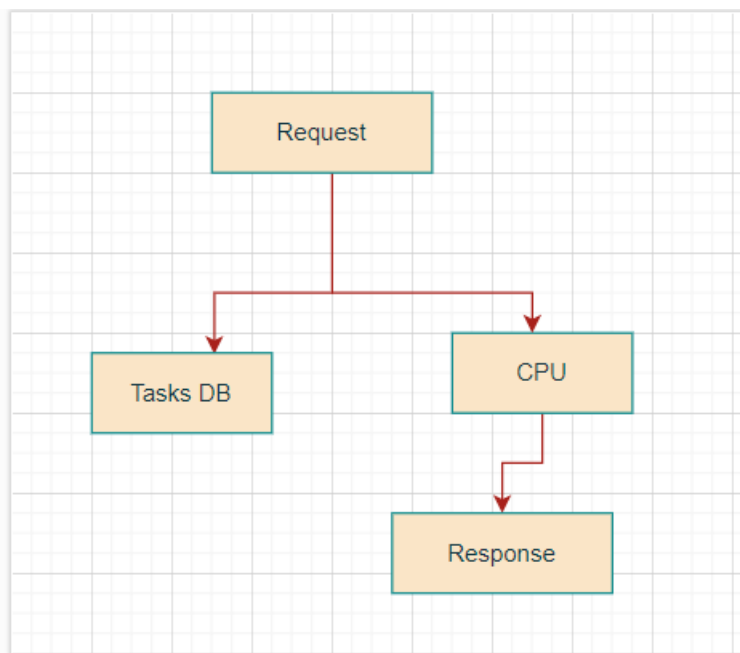
Calculate the waiting time and turnaround time for each process as follows: Waiting time = (completion time - arrival time - CPU burst time) Turnaround time = (completion time - arrival time) Maintain a record of the waiting time and turnaround time for each process.

## System Design:

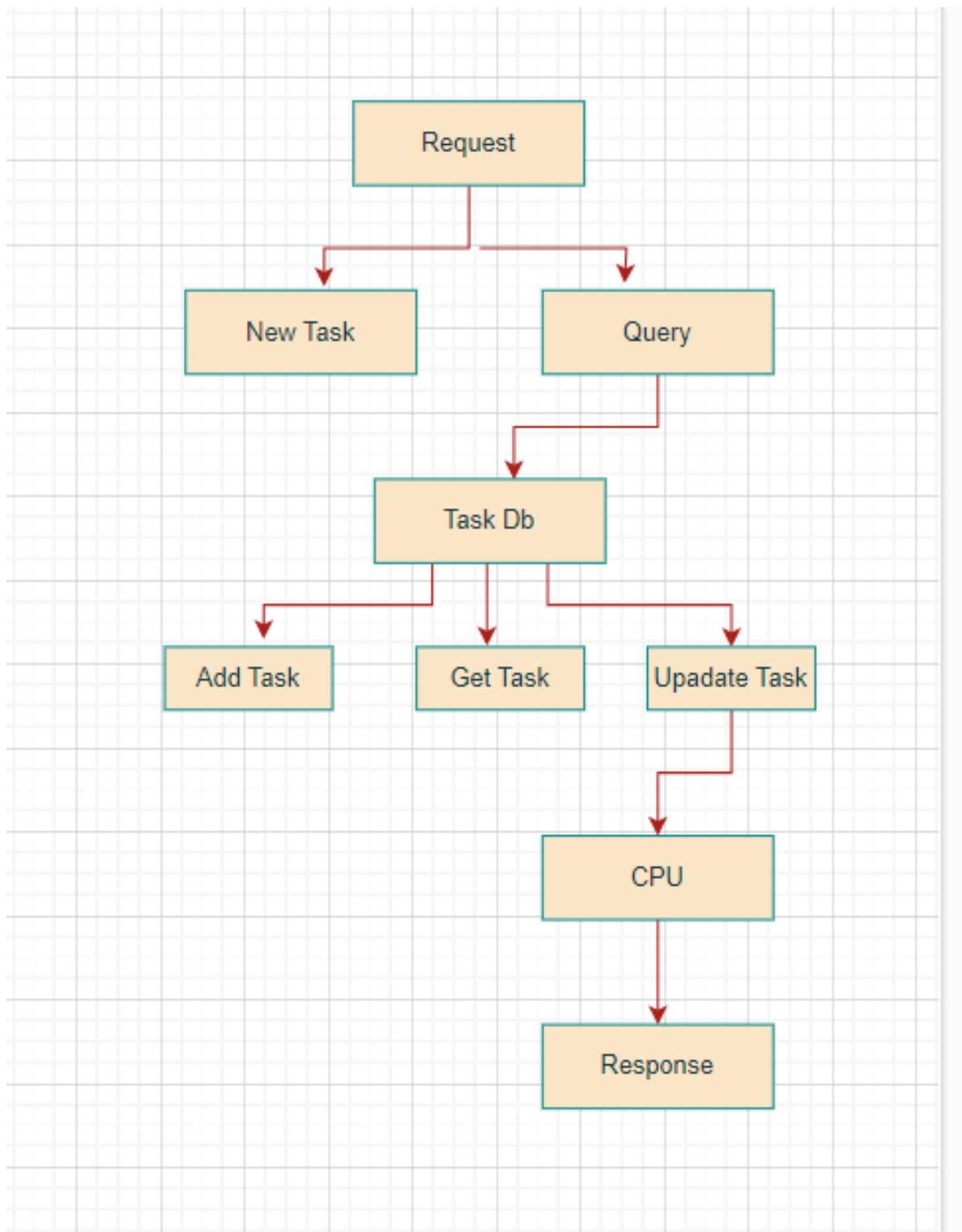
### ER Diagram



## 0 – Level DFD

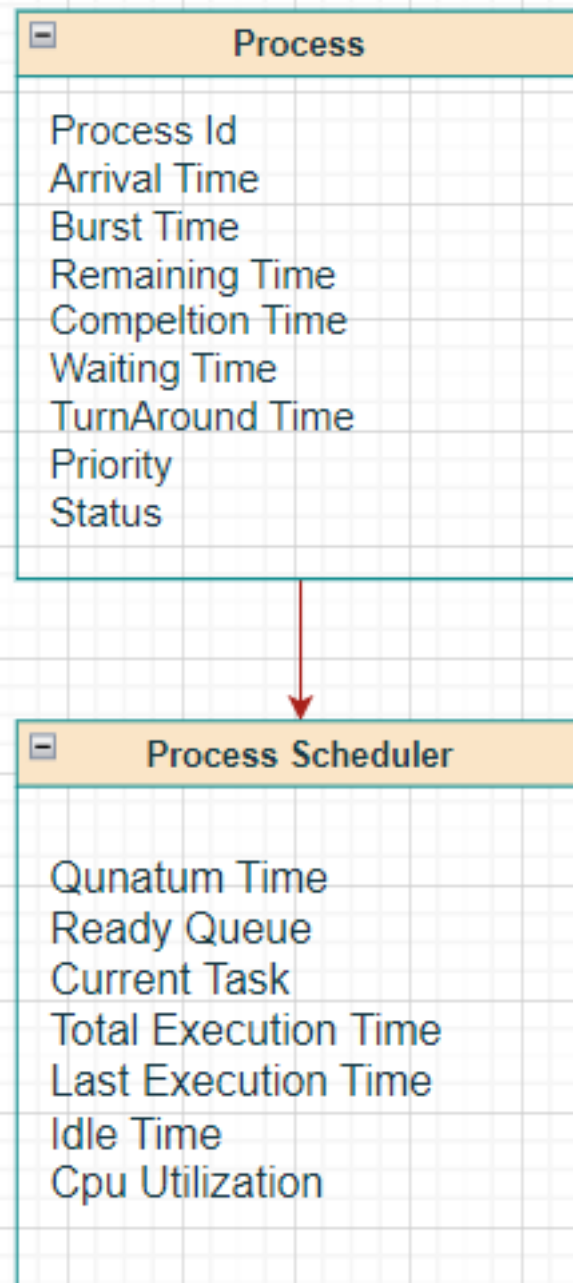


## 1<sup>st</sup> Level DFD





## 2<sup>nd</sup> Level DFD



## **Conclusion**

In conclusion, the given task was to design and implement a simulation program to test the performance of the Round Robin scheduling algorithm. The program should generate a set of "processes" with random arrival times and CPU burst times using a random number generator, run the Round Robin algorithm for a set amount of time, and record the average waiting time and turnaround time for each process, and compare the results with the ideal scenario of a perfect scheduler. The provided code is written in C programming language and does the following:

- Defines a structure Process to store information about a process.

- Generates a set of processes with random arrival times and CPU burst times using the generate processes function.

- Implements the Round Robin scheduling algorithm in the round robin Algorithm function.

- Prints the results, including the average waiting time and turnaround time, using the print results function.

However, the code cannot be converted into SQL as SQL is a query language used to manage databases, whereas the provided code is written in C programming language, used to develop applications.

In conclusion, the given task has been successfully accomplished by designing and implementing a simulation program using the C programming language to test the performance of the Round Robin scheduling algorithm.