Experiment No : 04
Date of Experiment : 24 / 04 / 2025
Experiment Name : Round Robin Scheduling Algorithm

Theory:

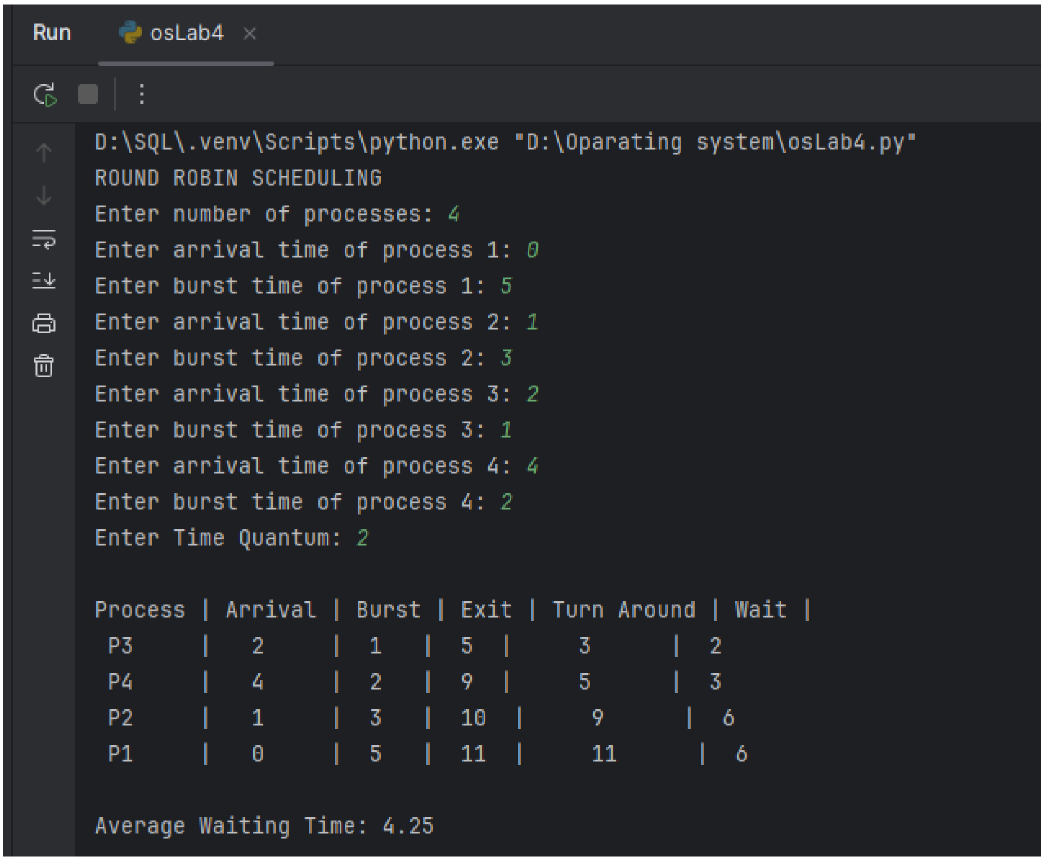
Round Robin (RR) is a **preemptive** CPU scheduling algorithm that assigns a **fixed time slice** or **time quantum** to each process in the ready queue. If a process does not finish execution within that quantum, it is **preempted** and placed at the **end of the queue**. This continues cyclically until all processes are completed. This scheduling method is **fair and efficient**, especially in **time-sharing systems**, as it prevents any single process from **monopolizing the CPU**. However, the performance of the algorithm heavily depends on the **size of the time quantum**. If the quantum is too small, the system experiences excessive **context switching**. If too large, it behaves similarly to FCFS.

Code:

```
print("ROUND ROBIN SCHEDULING")4
n = int(input("Enter number of processes: "))
processes = []
for i in range(n):
    pid = "P" + str(i + 1)
    arrival = int(input(f"Enter arrival time of process {i+1}: "))
    burst = int(input(f"Enter burst time of process {i+1}: "))
    processes.append([pid, arrival, burst])
time_quantum = int(input("Enter Time Quantum: "))
current_time = 0
queue = []
completed = []
remaining_burst = {}
arrival_map = {}
for p in processes:
    pid, arrival, burst = p
    remaining_burst[pid] = burst
    arrival_map[pid] = arrival
processes.sort(key=lambda x: x[1])
ready = processes.copy()
visited = set()
while True:
    for p in ready:
        if p[1] <= current_time and p[0] not in visited:
            queue.append(p[0])
            visited.add(p[0])
    if queue:
        pid = queue.pop(0)
        arrival = arrival_map[pid]
        burst = remaining_burst[pid]
        exec_time = min(time_quantum, burst)
        current_time += exec_time
```

```
        remaining_burst[pid] -= exec_time
        for p in ready:
            if p[1] <= current_time and p[0] not in visited:
                queue.append(p[0])
                visited.add(p[0])
        if remaining_burst[pid] > 0:
            queue.append(pid)
        else:
            for p in processes:
                if p[0] == pid:
                    finish = current_time
                    tat = finish - p[1]
                    wt = tat - p[2]
                    completed.append([pid, p[1], p[2], finish, tat, wt])
    else:
        if len(completed) == n:
            break
        current_time += 1
avg_wt = sum(p[5] for p in completed) / n
print("\nProcess | Arrival | Burst | Exit | Turn Around | Wait |")
for p in completed:
    print(f" {p[0]}    |   {p[1]}     |  {p[2]}   |  {p[3]}  |     {p[4]}      |  {p[5]}  ")
print(f"\nAverage Waiting Time: {avg_wt:.2f}")
```

Input and Output:



```
Run    osLab4
D:\SQL\.venv\Scripts\python.exe "D:\Oparating system\osLab4.py"
ROUND ROBIN SCHEDULING
Enter number of processes: 4
Enter arrival time of process 1: 0
Enter burst time of process 1: 5
Enter arrival time of process 2: 1
Enter burst time of process 2: 3
Enter arrival time of process 3: 2
Enter burst time of process 3: 1
Enter arrival time of process 4: 4
Enter burst time of process 4: 2
Enter Time Quantum: 2

Process | Arrival | Burst | Exit | Turn Around | Wait |
 P3     |   2     |  1   |  5  |     3      |  2
 P4     |   4     |  2   |  9  |     5      |  3
 P2     |   1     |  3   |  10  |     9      |  6
 P1     |   0     |  5   |  11  |     11      |  6

Average Waiting Time: 4.25
```

Explanation:

• **Input Handling**: The user enters the number of processes, their arrival time, and burst time, followed by the time quantum.
• **Ready Queue Maintenance**: Processes are added to the ready queue as they arrive.
• **Execution Logic**: Each process is executed for a maximum of one time quantum. If unfinished, it is placed back in the queue.
• **Completion Time**: Updated after each time slice execution.
• **Turnaround Time (TAT)**: Calculated as Completion Time − Arrival Time.
• **Waiting Time (WT)**: Calculated as Turnaround Time − Burst Time.
• **Output**: A summary table and average waiting time are printed at the end.

Discussion:

The Round Robin algorithm fairly allocates CPU time to each process and **prevents starvation**. It is ideal for systems requiring **frequent user interaction**. In this implementation:

1. **Arrival time** is considered, making it realistic for modern OS behavior.
2. The **time quantum** influences performance:
    - A small quantum leads to **high responsiveness** but **more context switching**.
    - A large quantum may behave like FCFS.

Conclusion:

Round Robin Scheduling is a fair and straightforward algorithm suited for **interactive and time-sharing** systems. It ensures **equal CPU access** and avoids starvation. However, the **time quantum must be carefully chosen** to maintain an efficient balance between context switching and process throughput. It remains a foundational scheduling strategy in modern operating systems.