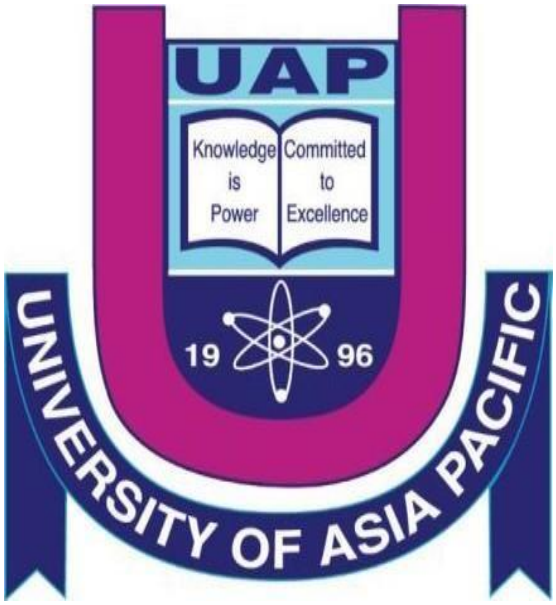

Department of Computer Science & Engineering

Course Title: Operating System Lab

Course Code: CSE 406

Lab Report No: 03

Lab Report: Round Robin Scheduling

Submission Date: 07-03-2025

Submitted To:
Atia Rahman Orthi
Lecturer,
Department of CSE, UAP

Submitted By:
Name: Amirul Islam Papon
Reg No: 21201076
Sec: B

Problem Statement: Round Robin is a preemptive scheduling algorithm where each process is assigned a fixed time quantum, ensuring fair CPU allocation among processes. The implementation should compute the Completion Time (CT), Turnaround Time (TAT), and Waiting Time (WT) for each process.

Steps to Implement Round Robin Scheduling:

1. **Input Process Details:** Accept process ID, arrival time, and burst time.
2. **Initialize Required Data Structures:** Create a queue for process scheduling and maintain lists for remaining burst time, completion time, turnaround time, and waiting time.
3. **Sort Processes by Arrival Time:** Ensure proper sequence for scheduling.
4. **Execute Round Robin Scheduling:**
    - Select the first process in the queue.
    - Allocate CPU time based on the time quantum.
    - If the process is not completed, re-add it to the queue.
    - Keep track of completion time for finished processes.

Source Code:

robin_round_scheduling.py > ...

```python
def antum):
    (variable) time: Literal[0]
    time = 0  # Current time
    result = []  # Store execution order
    remaining_bt = {p[0]: p[2] for p in processes}  # Remaining burst time
    arrival_times = {p[0]: p[1] for p in processes}  # Arrival times
    completion_time = {}  # Completion time
    turn_around_time = {}  # Turnaround time
    waiting_time = {}  # Waiting time
    processes.sort(key=lambda x: x[1])  # Sort by arrival time

    i = 0
    while i < len(processes) and processes[i][1] <= time:
        queue.append(processes[i][0])
        i += 1

    while queue:
        process = queue.pop(0)  # Fetch first process in queue
        if remaining_bt[process] > time_quantum:
            result.append((process, time, time + time_quantum))
            time += time_quantum
            remaining_bt[process] -= time_quantum
        else:
            result.append((process, time, time + remaining_bt[process]))
            time += remaining_bt[process]
            completion_time[process] = time
            remaining_bt[process] = 0

        # Add new processes that have arrived by now
        while i < len(processes) and processes[i][1] <= time:
            queue.append(processes[i][0])
            i += 1

        # Re-add the process if it's not finished yet
        if remaining_bt[process] > 0:
            queue.append(process)

    # Calculate turnaround time and waiting time
    for process, arrival, burst in processes:
        turn_around_time[process] = completion_time[process] - arrival
        waiting_time[process] = turn_around_time[process] - burst

    # Print full process table
    print("Process  Arrival Time  Burst Time  Completion Time  Turn Around Time  Waiting Time")
    for process, arrival, burst in processes:
        print(f"{process}\t\t\t{arrival}\t\t\t{burst}\t\t\t{completion_time[process]}\t\t\t\t{turn_around_time[process]}\t\t\t\t\t{waiting_time[process]}")

# Given process data
process_data = [
    ('P1', 0, 5),
    ('P2', 1, 4),
    ('P3', 2, 2),
    ('P4', 4, 1),
]
```

Output:

```
Process  Arrival Time  Burst Time  Completion Time  Turn Around Time  Waiting Time
P1          0           5           12               12                 7
P2          1           4           11               10                 6
P3          2           2           6                4                  2
P4          4           1           9                5                  4

[Done] exited with code=0 in 0.053 seconds
```

Discussion and Conclusion: Round Robin Scheduling ensures that all processes get CPU time fairly, reducing starvation. It is effective for time-sharing systems but may lead to higher average waiting times if the time quantum is too small. The choice of time quantum plays a crucial role in balancing CPU utilization and responsiveness

Source Code: https://github.com/Amirul-Islam-Papon/Operating-System/blob/main/robin_round_scheduling.py