Department Of Computer Science and Engineering

Course Title: Operating System Lab

Course Code: CSE 406

Title: CPU Scheduling Round Robin Algorithm

Submitted To

Atia Rahman Orthi
Lecturer
Department Of Computer Science & Engineering

Submitted By

Md. Atikul Islam Atik
Reg No:21201063
Sec:B1

Submission Date: 28.02.2025

1)Problem Statement:

Input:

Process	Arrival	Burst
P1	0	5
P2	1	4
P3	2	2
P4	4	1

Output:

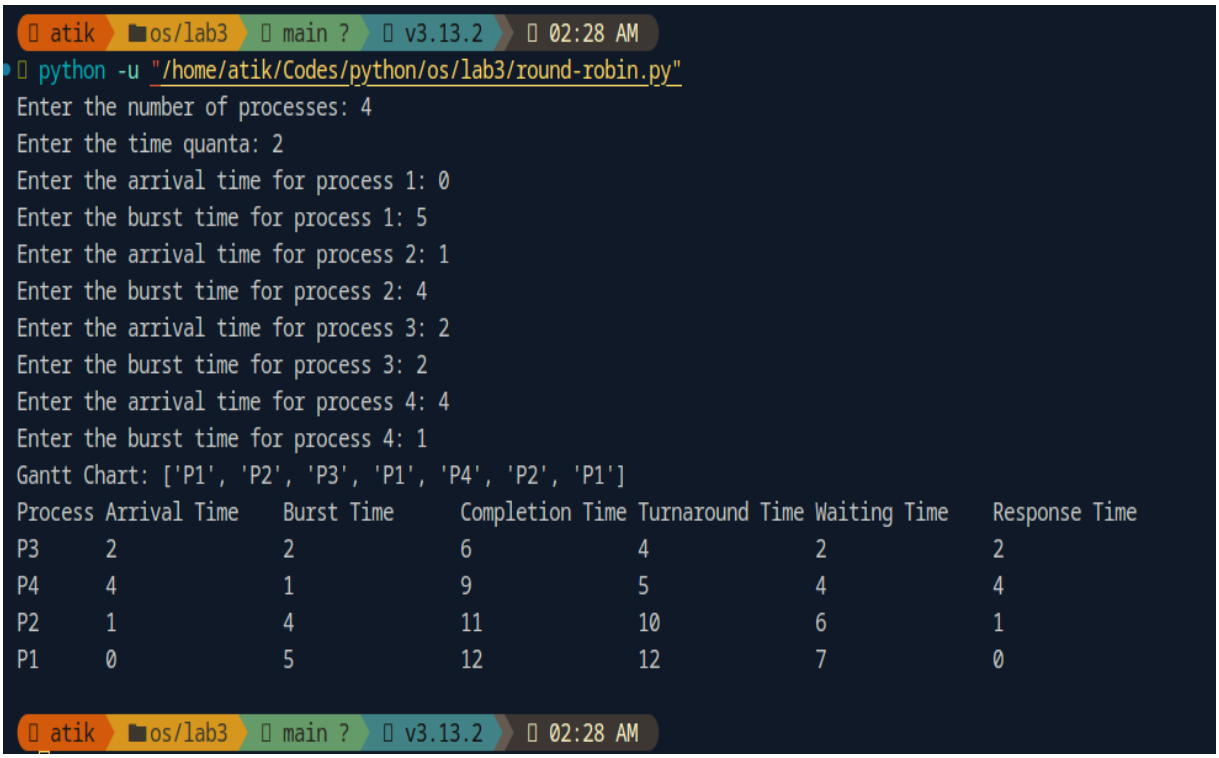
Process	AT	BT	CT	TAT	WT	RT
P3	2	2	6	4	2	2
P4	4	1	9	5	4	4
P2	1	4	11	10	6	1
P1	0	5	12	12	7	0

2)Source Code:

```python
from collections import deque

def round_robin(process_list, time_quanta):
    time = 0
    gantt_chart = []
    completed = {}
    backup = {}
    queue = deque()
    first_response = {}  # To track first execution time
    # Sort processes by arrival time
    process_list.sort()

    # Create a backup of original burst times
    for process in process_list:
        pid = process[2]
        arrival_time = process[0]
        burst_time = process[1]
        backup[pid] = {"arrival_time": arrival_time, "burst_time": burst_time}
        first_response[pid] = -1  # Initialize response time tracking

    remaining_processes = process_list[:]

    while remaining_processes or queue:
        # Add newly available processes to the queue
        for process in remaining_processes[:]:
            if process[0] <= time:
                queue.append(process)
                remaining_processes.remove(process)

        # If queue is empty, CPU remains idle
        if not queue:
            gantt_chart.append("Idle")
            time += 1
            continue

        # Process execution
        process = queue.popleft()
        pid = process[2]

        # Record first response time if it's the first execution
        if first_response[pid] == -1:
            first_response[pid] = time - process[0]

        # Execute process for at most time_quanta
        execution_time = min(time_quanta, process[1])
        gantt_chart.append(pid)
        time += execution_time
        process[1] -= execution_time

        # Check for new arrivals while executing
        for p in remaining_processes[:]:
            if p[0] <= time:
                queue.append(p)
                remaining_processes.remove(p)

        # If process is completed, record its completion time
        if process[1] == 0:
            completion_time = time
            arrival_time = backup[pid]["arrival_time"]
            burst_time = backup[pid]["burst_time"]
            turnaround_time = completion_time - arrival_time
            waiting_time = turnaround_time - burst_time
            response_time = first_response[pid]  # Fetch the recorded response time
            completed[pid] = [arrival_time, burst_time, completion_time, turnaround_time, waiting_time, response_time]
        else:
            queue.append(process)  # Re-add the process if it's not finished

    return {"gantt_chart": gantt_chart, "completed": completed}

def print_table(completed):
    print("Process\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\tResponse Time")
    for key, values in completed.items():
        print(f"{key}\t{values[0]}\t\t{values[1]}\t\t{values[2]}\t\t{values[3]}\t\t{values[4]}\t\t{values[5]}")

def collect_input():
    process_list = []
    n = int(input("Enter the number of processes: "))
    time_quanta = int(input("Enter the time quanta: "))
    for i in range(n):
        arrival_time = int(input(f'Enter the arrival time for process {i+1}: '))
        burst_time = int(input(f'Enter the burst time for process {i+1}: '))
        pid = f"P{i+1}"
        process_list.append([arrival_time, burst_time, pid])
    return process_list, time_quanta

if __name__ == "__main__":
    inputs = collect_input()
    result = round_robin(inputs[0], inputs[1])

    print("Gantt Chart:", result["gantt_chart"])
    print_table(result["completed"])
```

[Live Link of Code]

3)Output Consular Picture:



4)Round Robin Algorithm:

1) Sort all processes based on arrival time.
2) Initialize time = 0, an empty queue, and tracking dictionaries.
3) Add newly arrived processes to the queue.
4) If the queue is empty, increment time (CPU remains idle).
5) If the queue is not empty, dequeue the first process.
6) Record the response time if it's the first execution of the process.

7) Execute the process for at most time_quanta.

8) Update time += execution_time, reduce the process's burst time.

9) Add newly arrived processes to the queue during execution.

10) If the process is not finished, re-add it to the queue; otherwise, calculate completion, turnaround, waiting, and response times.

11) Repeat from step 3 until all processes are completed.

5)Conclusion:

The Round Robin (RR) scheduling algorithm was successfully implemented in this assignment. The program takes input for time quantum and the number of processes along with their arrival times and burst times . It then sorts the processes based on their arrival times and executes them in a time-sharing manner using a fixed time quantum.

The key metrics calculated include:

- **Completion Time (CT):** The time at which a process completes its execution.
- **Turnaround Time (TAT):** The total time taken from the arrival of the process to its completion.
- **Waiting Time (WT):** The time a process spends waiting in the queue before it starts executing.
- **Response Time (RT):** The time from process arrival to its first execution.

The output clearly shows the execution order in the Gantt chart and provides detailed scheduling results for each process. The results demonstrate that the Round Robin algorithm ensures fair CPU allocation by giving each process a fixed time slice, reducing starvation. However, selecting an appropriate time quantum is crucial to balancing responsiveness and efficiency.