SVKM'S NMIMS

MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT& ENGINEERING

(Campus Name)

Academic Year: 2020-2021

Practical 4 – Program to Demonstrate the Round Robin Algorithm

Name Arjun Mehta	Roll_No K036	SAP-ID 70102300018

Dear all,

Kindly complete the following task with your name in output file also attach the C program with the file.

Find the Turnaround time and Average Turnaround time.

Find the Waiting time and Average Waiting time.

1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time.
   Waiting Time = Turn Around Time – Burst Time

Problem – 1: 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

If the CPU scheduling policy is Round Robin with time quantum = 2 unit, calculate the average waiting time and average turn around time

Problem 2:

Consider the set of 6 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

If the CPU scheduling policy is Round Robin with time quantum = 2, calculate the average waiting time and average turn around time.

Code:

```
from collections import deque


def round_robin(processes, time_quantum):
    processes = sorted(processes, key=lambda x: x['arrival_time'])
    queue = deque()
    current_time = 0
    next_process = 0
    total_tat = 0
    total_wt = 0

    # Initialize remaining_time for all processes
    for p in processes:
        p['remaining_time'] = p['burst_time']
        p['completion_time'] = None
```

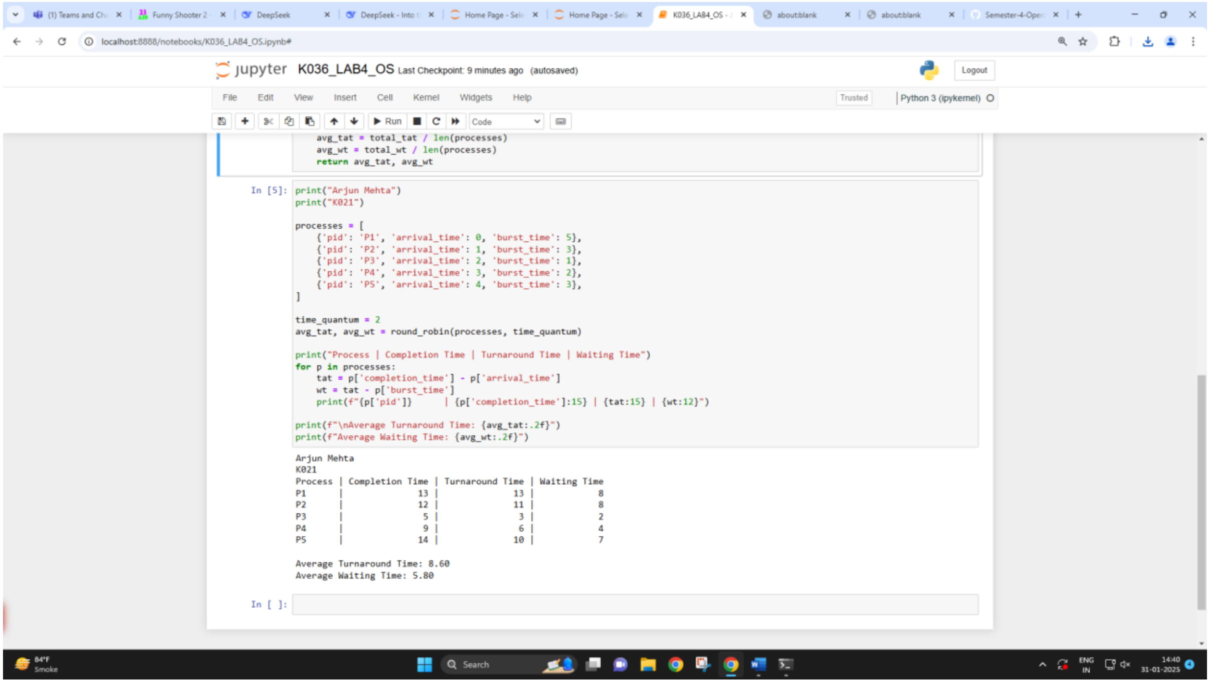
```
    # Add initial processes to the queue

    while next_process < len(processes) and processes[next_process]['arrival_time'] <=
current_time:

        queue.append(processes[next_process])

        next_process += 1



    while queue:

        current_process = queue.popleft()

        execute_time = min(time_quantum, current_process['remaining_time'])

        current_time += execute_time

        current_process['remaining_time'] -= execute_time



        # Add newly arrived processes to the queue

        while next_process < len(processes) and processes[next_process]['arrival_time'] <=
current_time:

            queue.append(processes[next_process])

            next_process += 1



        if current_process['remaining_time'] == 0:

            current_process['completion_time'] = current_time

            tat = current_process['completion_time'] - current_process['arrival_time']

            wt = tat - current_process['burst_time']

            total_tat += tat

            total_wt += wt

        else:
```

```
        queue.append(current_process)

    avg_tat = total_tat / len(processes)
    avg_wt = total_wt / len(processes)
    return avg_tat, avg_wt

# Problem 1 data
processes = [
    {'pid': 'P1', 'arrival_time': 0, 'burst_time': 5},
    {'pid': 'P2', 'arrival_time': 1, 'burst_time': 3},
    {'pid': 'P3', 'arrival_time': 2, 'burst_time': 1},
    {'pid': 'P4', 'arrival_time': 3, 'burst_time': 2},
    {'pid': 'P5', 'arrival_time': 4, 'burst_time': 3},
]

time_quantum = 2
avg_tat, avg_wt = round_robin(processes, time_quantum)

# Print results
print("Process | Completion Time | Turnaround Time | Waiting Time")
for p in processes:
    tat = p['completion_time'] - p['arrival_time']
    wt = tat - p['burst_time']
    print(f"{p['pid']}      | {p['completion_time']:15} | {tat:15} | {wt:12}")
```

print(f"\nAverage Turnaround Time: {avg_tat:.2f}")

print(f"Average Waiting Time: {avg_wt:.2f}")

OUTPUT:



Conclusion: -

Successfully learnt and implemented round robin.

The round-robin algorithm has several advantages over other scheduling algorithms. First, it is easy to implement and understand, as it only requires a simple queue and a timer. Second, it is fair and equitable, as it gives each task an equal share of the CPU time and prevents starvation.