Year:-	Academic Year- 2025-26	Semester:-

Experiment # 6

Roll No: D084	Name: Somish Jain
Class: BTech CE-B	Batch: 2
Date of Experiment: 25th August 2025	Date of Submission:25th August 2025

Study / Implementation details:

```
import matplotlib.pyplot as plt
import pandas as pd
import requests
import random


def rr_execute(process_set, q_time):
    """
    Round Robin CPU Scheduling Simulation
    process_set: [(pid, arrival, burst), ...]
    q_time: quantum slice
    """
    remaining = {p: bt for p, at, bt in process_set}
    start_times, end_times = {}, {}
    log = []
    clock = min(at for _, at, _ in process_set)
    ready_q, idx = [], 0
    sorted_proc = sorted(process_set, key=lambda x: x[1])
```

```
while True:
    # Admit new arrivals
    while idx < len(sorted_proc) and sorted_proc[idx][1] <= clock:
        ready_q.append(sorted_proc[idx][0])
        idx += 1

    if not ready_q:
        if idx < len(sorted_proc):
            clock = sorted_proc[idx][1]
            continue
        else:
            break

    current = ready_q.pop(0)
    if current not in start_times:
        start_times[current] = clock

    exec_start = clock
    if remaining[current] > q_time:
        clock += q_time
        remaining[current] -= q_time
    else:
        clock += remaining[current]
        remaining[current] = 0
        end_times[current] = clock
```

```
    exec_end = clock

    log.append((current, exec_start, exec_end))

    while idx < len(sorted_proc) and sorted_proc[idx][1] <= clock:
        ready_q.append(sorted_proc[idx][0])
        idx += 1
    if remaining[current] > 0:
        ready_q.append(current)

results = []
for pid, at, bt in process_set:
    st = start_times[pid]
    ct = end_times[pid]
    tat = ct - at
    wt = tat - bt
    rt = st - at
    results.append([pid, at, bt, st, ct, tat, wt, rt])

df = pd.DataFrame(results, columns=[
    "PID", "Arrive", "Burst", "FirstExec",
    "Complete", "TAT", "Wait", "Response"
])
avg = {
    "Mean Waiting": round(df["Wait"].mean(), 2),
```

```
        "Mean Turnaround": round(df["TAT"].mean(), 2),
        "Mean Response": round(df["Response"].mean(), 2)
    }
    return log, df, avg
def timeline_chart(executions):
    fig, ax = plt.subplots(figsize=(11, 3))
    for i, (pid, st, en) in enumerate(executions):
        color = (random.random(), random.random(), random.random())
        ax.barh(0, en - st, left=st, color=color, edgecolor="black", height=0.6)
        ax.text((st + en) / 2, 0, pid, ha="center", va="center", color="white", fontsize=9,
weight="bold")
        ax.text(st, 0.4, str(st), ha="center", fontsize=7)
    ax.text(executions[-1][2], 0.4, str(executions[-1][2]), ha="center", fontsize=7)
    ax.set_yticks([])
    ax.set_title("Round Robin Execution Flow", fontsize=13, fontweight="bold")
    ax.set_xlabel("Timeline")
    plt.tight_layout()
    plt.show()
def from_user():
    print("\nManual Input Mode Selected")
    proc_list = []
    for i in range(1, 6):
        at = float(input(f"Arrival time for P{i}: "))
        bt = float(input(f"Burst time for P{i}: "))
        proc_list.append((f"P{i}", at, bt))
```

```
    tq = float(input("Enter Quantum Value: "))
    return proc_list, tq
def from_api():
    print("\nAPI Mode Activated")
    url = "https://dummyjson.com/users"
    data = requests.get(url).json()["users"][:5]
    proc_list = []
    for i, user in enumerate(data):
        pid = f"P{i+1}"
        arr = i * 3
        bt = (len(user["firstName"]) % 6) + 2
        proc_list.append((pid, arr, bt))
    tq = random.randint(2, 5)
    print(f"Assigned Time Slice (random): {tq}")
    return proc_list, tq
if __name__ == "__main__":
    print("=== CPU Scheduler (Round Robin) ===")
    print("Select data source:")
    print("1 → Keyboard Entry")
    print("2 → Online API (auto)")
    choice = input("Your option: ").strip()
    if choice == "1":
        processes, quantum = from_user()
    elif choice == "2":
        processes, quantum = from_api()
```

```
else:
    print("Invalid input, stopping program.")
    exit()
timeline, table, summary = rr_execute(processes, quantum)
print("\n--- Results Table ---")
print(table.to_string(index=False))
print("\n--- Average Metrics ---")
for k, v in summary.items():
    print(f"{k}: {v}")
timeline_chart(timeline)
```

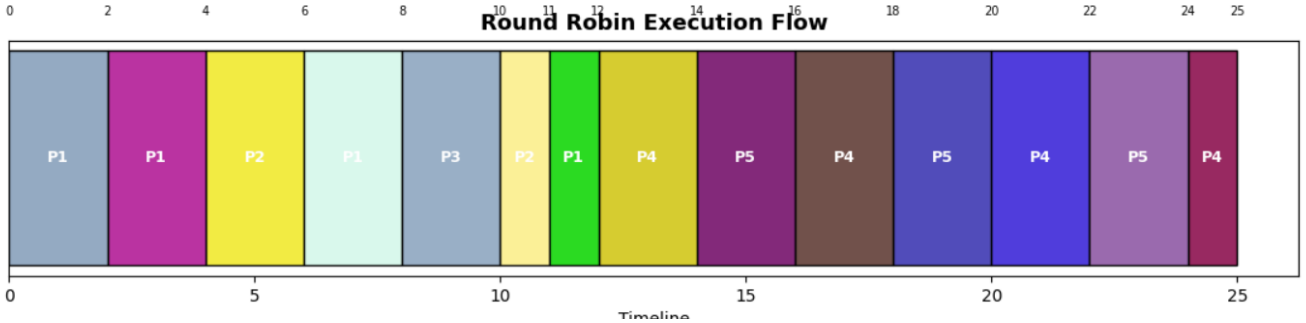
OUTPUT:-

```
2 → Online API (auto)
Your option: 2

API Mode Activated
Assigned Time Slice (random): 2

--- Results Table ---
PID  Arrive  Burst  FirstExec  Complete  TAT  Wait  Response
 P1       0      7          0        12   12     5         0
 P2       3      3          4        11    8     5         1
 P3       6      2          8        10    4     2         2
 P4       9      7         12        25   16     9         3
 P5      12      6         14        24   12     6         2

--- Average Metrics ---
Mean Waiting: 5.4
Mean Turnaround: 10.4
Mean Response: 1.6
```



Questions for Assessment (QA)

1. Difference between RR, FCFS, and SJF

- RR: Uses time quantum, cycles through processes → fair allocation.
- FCFS: Runs in arrival order → long jobs may delay short ones.
- SJF: Chooses shortest job → long jobs risk starvation.
- Key: RR ensures fairness; FCFS/SJF may cause imbalance.

---

2. Significance of Time Quantum

- Defines max CPU time per process.
- Too large: Acts like FCFS; poor response for short jobs.
- Too small: Excessive context switching; low efficiency.
- Best: Balanced quantum for responsiveness + efficiency.

---

3. Response vs Waiting Time in RR

- Response Time: First execution delay = Start – Arrival.
- Waiting Time: Time in queue = Turnaround – Burst.
- In RR: Response is quick; Waiting may grow due to repeated turns.

---

4. Handling Frequent Arrivals

- New processes join queue immediately.
- Scheduler continues current job till quantum ends, then cycles in new arrivals → fairness maintained.

---

5. Starvation in RR

- Cannot occur.
- Every process gets CPU after limited turns → all treated equally.

---

6. Applying RR to Real Data

- Needs:
    - Dynamic queue for arrivals.
    - Efficient structures (deque).
    - I/O handling and re-entry.
    - Non-zero context switch cost.
    - Scalable Gantt visualization.
    - Adaptive quantum based on workload.