Experiment-2
Name: Veeransh Shah
Reg Id: 221070063

Aim: To create CPU scheduling algorithms

Theory:

This Python code implements four different CPU scheduling algorithms: First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling. It also includes a function to plot Gantt charts for visualization of the scheduling.

Here's a brief explanation of each scheduling algorithm:

1. **First-Come-First-Serve (FCFS):**
   - Processes are scheduled based on their arrival times.
   - The waiting time for each process is calculated as the sum of the burst times of all previous processes.
   - The turnaround time is the sum of waiting time and burst time.
2. **Shortest Job First (SJF):**
   - Processes are scheduled based on their burst times.
   - The waiting time for each process is calculated similarly to FCFS.
   - Turnaround time is the sum of waiting time and burst time.
3. **Round Robin (RR):**
   - Each process is assigned a fixed time quantum (here, 2 seconds).
   - Processes are scheduled in a circular order, allowing each process to execute for the time quantum before moving to the next process in the queue.
   - This continues until all processes are completed.
4. **Priority Scheduling:**
   - Processes are scheduled based on their priority.
   - The waiting time and turnaround time are calculated similarly to the previous algorithms.

Code:

```
import matplotlib.pyplot as plt;
from matplotlib.patches import Rectangle
def fcfs(processes):
   # Sort processes based on arrival time
   processes.sort(key=lambda x: x[0])

   # Initialize waiting time and turnaround time arrays
   n = len(processes)
   timeline = []

   current_time = 0
```

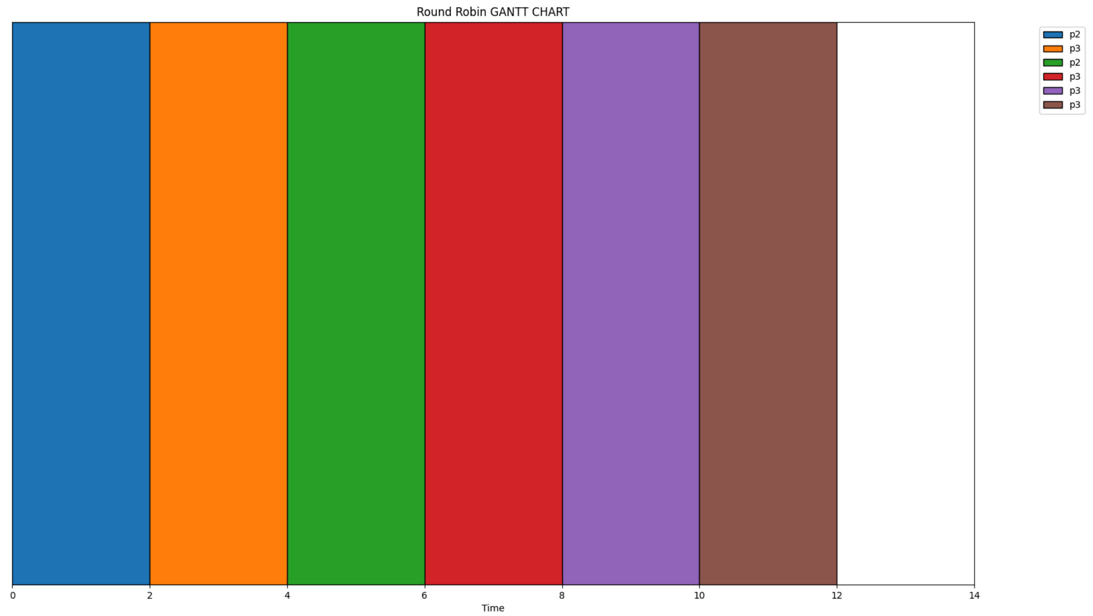
```python
    for process in processes:
        timeline.append((process[0], current_time,
process[1],process[3]))
        current_time += process[1]

    waiting_time = [0] * n
    turnaround_time =[0]*n
    waiting_time[0] = 0
    turnaround_time[0] = processes[0][1]

    for i in range(1, n):
        waiting_time[i] = turnaround_time[i - 1]
        turnaround_time[i] = waiting_time[i] + processes[i][1]


    avg_waiting_time = sum(waiting_time) / n
    avg_turnaround_time = sum(turnaround_time) / n

    # Print the result
    print("FCFS Scheduling")
    print("Process\t\tArrival Time\t\tBurst Time\t\tWaiting
Time\t\tTurnaround Time")
    for i in range(n):

print(f"{processes[i][3]}\t\t{processes[i][0]}\t\t\t{processes[i][1]}\t
\t\t{waiting_time[i]}\t\t\t{turnaround_time[i]}")

    print(f"\nAverage Waiting Time: {avg_waiting_time}")
    print(f"Average Turnaround Time: {avg_turnaround_time}")

    return timeline


def sjf(processes):
    processes.sort(key=lambda x: (x[0],x[1]))
    n = len(processes)
    timeline = []

    current_time = 0
    for process in processes:
        timeline.append((current_time, current_time + process[1],
process[1],process[3]))
        current_time += process[1]
```

```
    waiting_time = [0] * n
    turnaround_time = [0] * n

    waiting_time[0] = 0
    turnaround_time[0] = processes[0][1]

    for i in range(1, n):
        waiting_time[i] = turnaround_time[i - 1]
        turnaround_time[i] = waiting_time[i] + processes[i][1]

    avg_waiting_time = sum(waiting_time) / n
    avg_turnaround_time = sum(turnaround_time) / n

    print("SJF Scheduling:")
    print("Process\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time")
    for i in range(n):

print(f"{processes[i][3]}\t\t{processes[i][1]}\t\t\t{waiting_time[i]}\t
\t\t{turnaround_time[i]}")

    print(f"\nAverage Waiting Time: {avg_waiting_time}")
    print(f"Average Turnaround Time: {avg_turnaround_time}\n")

    return timeline

def round_robin(processes):
    time_quantum = 2
    n = len(processes)
    rem_time = [process[1] for process in processes]
    process_name=[process[3] for process in processes]
    waiting_time = [0] * n
    turnaround_time = [0] * n
    total_executed_time = 0
    current_time = 0
    timeline=[]

    while any(rem_time):
        for i in range(n):
            if rem_time[i] > 0:
                execution_time = min(rem_time[i], time_quantum)
                timeline.append((i + 1, current_time,
execution_time,process_name[i]))
```

```
                current_time += execution_time
                rem_time[i] -= execution_time

    return timeline


def priority_scheduling(processes):
    processes.sort(key=lambda x: (x[0],x[2]))   # Sort processes based on
priority (assuming lower values indicate higher priority)
    n = len(processes)
    waiting_time = [0] * n
    turnaround_time = [0] * n

    waiting_time[0] = 0
    turnaround_time[0] = processes[0][1]

    for i in range(1, n):
        waiting_time[i] = turnaround_time[i - 1]
        turnaround_time[i] = waiting_time[i] + processes[i][1]

    avg_waiting_time = sum(waiting_time) / n
    avg_turnaround_time = sum(turnaround_time) / n

    print("Priority Scheduling:")
    print("Process\t\tPriority\t\tBurst Time\t\tWaiting
Time\t\tTurnaround Time")
    for i in range(n):

print(f"{processes[i][3]}\t\t{processes[i][2]}\t\t\t{processes[i][1]}\t
\t\t{waiting_time[i]}\t\t\t{turnaround_time[i]}")

    print(f"\nAverage Waiting Time: {avg_waiting_time}")
    print(f"Average Turnaround Time: {avg_turnaround_time}\n")

def plot_gantt_chart(timeline,title):
    fig, ax = plt.subplots(figsize=(10, 1))

    for i,entry in enumerate(timeline):
        rect = Rectangle((entry[1], 0), entry[2], 1, edgecolor='black',
facecolor=f'C{i}', label=f'{entry[3]}')
        ax.add_patch(rect)

    ax.set_xlim(0, max(entry[1] + entry[2] for entry in timeline) + 2)
```

```python
    ax.set_ylim(0, 1)
    ax.set_yticks([])

    plt.title(title)
    plt.xlabel('Time')
    plt.legend(loc='upper right', bbox_to_anchor=(1.12, 1))
    plt.show()

if __name__ == "__main__":
    processes=[]
    p=int(input("Enter number of processes-"))
    for x in range(1,p+1):
        process_name=input("Enter process name-")
        arr_time=int(input("Arrrival Time for the process-"))
        burst_time=int(input("Execution time for the process in sec-"))
        priority=int(input("Priority of the process(Between 1 and "+
str(p) + ")"))
        processes.append((arr_time,burst_time,priority,process_name))
    model=int(input("Which Model do you want to
use?\n1-FCFS\n2-SJF\n3-RR\n4-Priority\nEnter Option-"))
    if(model==1):
        timeline=fcfs(processes)
        plot_gantt_chart(timeline,'FCFS GANTT CHART')
    elif(model==2):
        timeline=sjf(processes)
        plot_gantt_chart(timeline,'SJF GANTT CHART')
    elif(model==3):
        timeline=round_robin(processes)
        plot_gantt_chart(timeline,'Round Robin GANTT CHART')
    elif(model==4):
        timeline=priority_scheduling(processes)
        plot_gantt_chart(timeline,'PRIORITY GANTT CHART')
    else:
        print("No other method available")
```

Output:



Conclusion:

First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling. Each algorithm comes with its own set of rules and criteria for determining the order in which processes are executed, influencing the overall efficiency of the system.