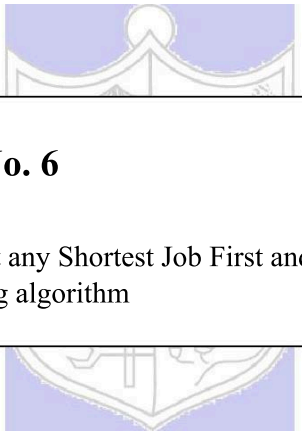


Experiment No. 6

Title: Implement any Shortest Job First and Round Robin scheduling algorithm

Batch: B-2 Roll No: 16010422234 Name: Chandana Galgali Date: 13/09/2024

Experiment No: 6

Aim: Implement any Shortest Job First and Round Robin scheduling algorithm

---

Resources needed: Any Java/C/C++/Python editor and compiler

Theory:

Pre lab/Prior concepts:

CPU scheduling algorithms are used by operating systems to manage how processes are assigned to the CPU for execution. The primary objective of these algorithms is to maximize CPU utilization and ensure that processes are completed efficiently. Here are the most common CPU scheduling algorithms:

1. First-Come, First-Served (FCFS) Scheduling

- **Description**: The simplest scheduling algorithm, where processes are executed in the order they arrive in the ready queue.
- **Advantages**: Easy to understand and implement.
- **Disadvantages**: Can lead to the **convoy effect**, where short processes have to wait a long time for longer processes to finish (i.e., high average waiting time).

2. Shortest Job Next (SJN) or Shortest Job First (SJF)

- **Description**: The process with the shortest estimated execution time is selected next. It can be preemptive (Shortest Remaining Time First - SRTF) or non-preemptive.
- **Advantages**: Minimizes average waiting time.
- **Disadvantages**: It requires knowledge of the execution time of processes in advance, which is not always possible. It may also suffer from **starvation** (long processes might never get scheduled).

3. Round-Robin (RR) Scheduling

- **Description**: Each process is assigned a small time slice (called a **quantum**), and the CPU switches to the next process in the queue after this time. This is a preemptive algorithm.
- **Advantages**: Fair allocation of CPU time to all processes and good for time-sharing systems.
- **Disadvantages**: Performance depends on the size of the quantum. Too small a quantum leads to frequent context switching, while too large can make it behave like FCFS.

4. Priority Scheduling

- **Description**: Each process is assigned a priority, and the CPU is allocated to the process with the highest priority. Can be preemptive or non-preemptive.
- **Advantages**: Flexibility in handling different types of tasks (urgent processes can be prioritized).
- **Disadvantages**: It can lead to **starvation** for lower-priority processes. This can be mitigated using techniques like **aging** (increasing the priority of processes the longer they wait).

Key Metrics to Evaluate CPU Scheduling Algorithms:

- **Throughput**: Number of processes completed per unit of time.
- **CPU Utilization**: Percentage of time the CPU is actively processing tasks.
- **Turnaround Time**: Total time taken from submission to completion of a process.
- **Waiting Time**: Time a process spends waiting in the ready queue.
- **Response Time**: Time between a process's request and its first execution.

Each algorithm has trade-offs, and the best one depends on the specific requirements and characteristics of the operating system and applications being run.

Shortest Job First Algorithm:

1) Create a process with PID and CPU Burst time.
2) Sort the processes according to CPU Burst time and put them in the Ready queue.
3) Take one by one process for execution from the ready queue
4) Show execution of processes (Gantt Chart)
5) Calculate Average waiting time.
6) Calculate Average Turnaround time.

Activities:

1. Students are required to study and implement the RR (Round Robin) and SJF (Shortest Job First) scheduling algorithms.

- Create a process with PID and CPU Burst time and Quantum.
- Put in the Ready queue.
- Take one by one process for execution from the ready queue
- Show execution of processes (Gantt Chart)
- Calculate Average waiting time.
- Calculate Average Turnaround time.

2. Calculate the average waiting time and average turnaround time for each algorithm.

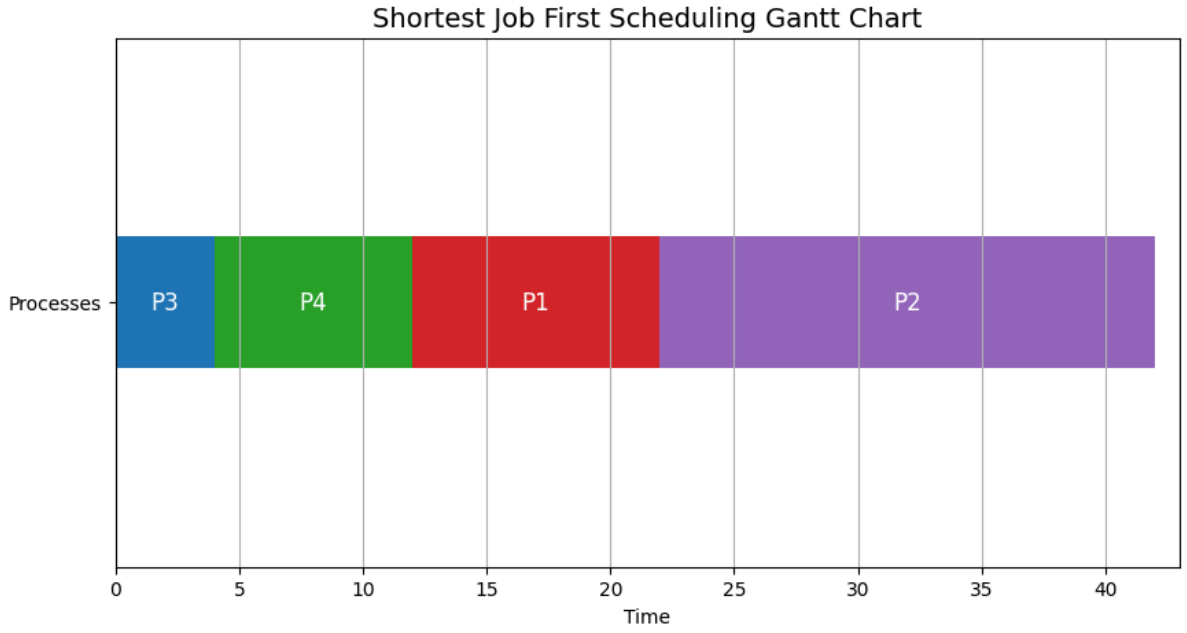
- Create a process with PID and CPU Burst time.
- Sort the processes according to CPU Burst time and put them in the Ready queue.
- Take one by one process for execution from the ready queue
- Show execution of processes (Gantt Chart)
- Calculate Average waiting time.
- Calculate Average Turnaround time.

Results:

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Function to calculate the average waiting time and turnaround time
for SJF
def sjf_scheduling(processes):
    n = len(processes)
    processes.sort(key=lambda x: x[1])  # Sort by burst time

    wait_time = [0] * n
    turnaround_time = [0] * n
    start_time = [0] * n
    current_time = 0
    gantt_chart = []

    # Calculate waiting time and turnaround time
    for i in range(n):
        pid, burst_time = processes[i]
        wait_time[i] = current_time
        turnaround_time[i] = current_time + burst_time
        start_time[i] = current_time
        current_time += burst_time
        gantt_chart.append((pid, start_time[i], current_time))

    avg_wait_time = sum(wait_time) / n
    avg_turnaround_time = sum(turnaround_time) / n

    print(f"SJF Average Waiting Time: {avg_wait_time}")
    print(f"SJF Average Turnaround Time: {avg_turnaround_time}")

    return gantt_chart

# Function to display the Gantt Chart with improved visualization
def display_gantt_chart(gantt_chart, title):
    fig, gnt = plt.subplots(figsize=(10, 5))

    gnt.set_ylim(0, 2)
    gnt.set_xlim(0, max([end for _, _, end in gantt_chart]) + 1)
    gnt.set_xlabel('Time')
    gnt.set_yticks([1])
    gnt.set_yticklabels(['Processes'])
```

```
    gnt.grid(True, axis='x')

    colors = ['tab:blue', 'tab:green', 'tab:red', 'tab:purple',
'tab:orange']

    for i, process in enumerate(gantt_chart):
        pid, start, end = process
        gnt.broken_barh([(start, end - start)], (0.75, 0.5),
facecolors=(colors[i % len(colors)]))
        gnt.text((start + end) / 2, 1, f"P{pid}", ha='center',
va='center', fontsize=12, color='white')

    plt.title(title, fontsize=14)
    plt.show()

# Taking input from the user
n = int(input("Enter number of processes for SJF: "))
processes = []
for i in range(n):
    pid = int(input(f"Enter process ID for process {i + 1}: "))
    burst_time = int(input(f"Enter CPU burst time for process {pid}:
"))
    processes.append((pid, burst_time))

gantt_chart_sjf = sjf_scheduling(processes)
display_gantt_chart(gantt_chart_sjf, "Shortest Job First Scheduling
Gantt Chart")
```

```
Enter number of processes for SJF: 4
Enter process ID for process 1: 1
Enter CPU burst time for process 1: 10
Enter process ID for process 2: 2
Enter CPU burst time for process 2: 20
Enter process ID for process 3: 3
Enter CPU burst time for process 3: 4
Enter process ID for process 4: 4
Enter CPU burst time for process 4: 8
SJF Average Waiting Time: 9.5
SJF Average Turnaround Time: 20.0
```

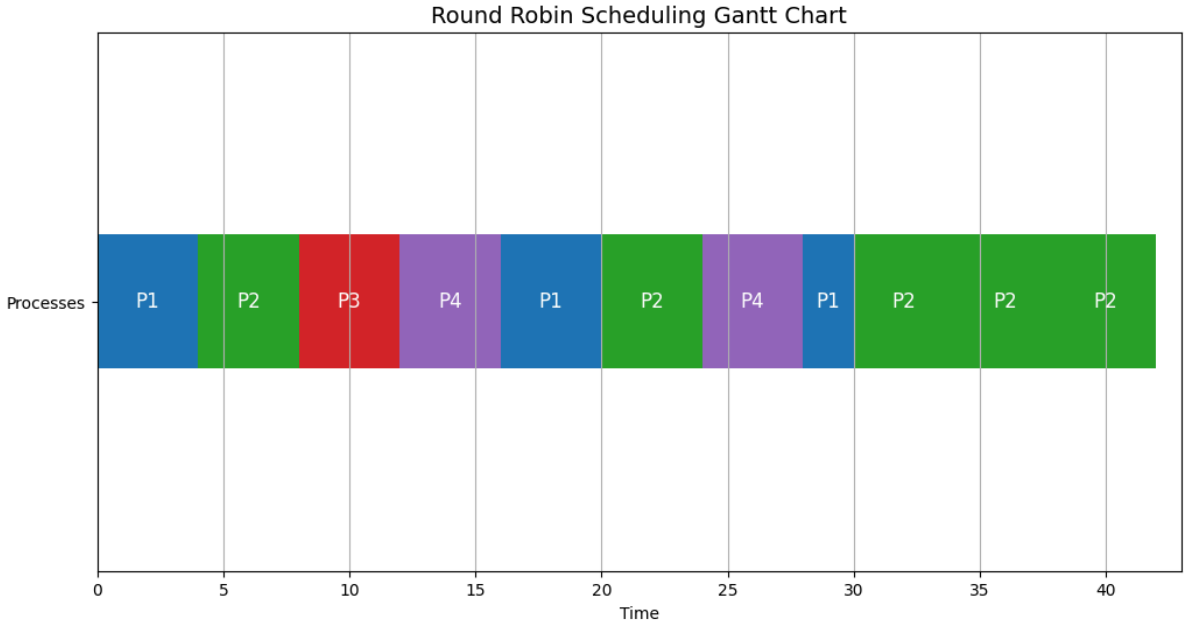



```python
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Function to calculate the average waiting time and turnaround time
for RR
def rr_scheduling(processes, quantum):
    n = len(processes)
    wait_time = [0] * n
    turnaround_time = [0] * n
    remaining_time = [burst_time for _, burst_time in processes]
    current_time = 0
    gantt_chart = []
    process_queue = []

    while any(remaining_time):
        for i in range(n):
            if remaining_time[i] > 0:
                pid, burst_time = processes[i]
                start_time = current_time
                execution_time = min(quantum, remaining_time[i])
                current_time += execution_time
                remaining_time[i] -= execution_time
                gantt_chart.append((pid, start_time, current_time))

                if remaining_time[i] == 0:
                    turnaround_time[i] = current_time
                    wait_time[i] = turnaround_time[i] - burst_time
```

```
    avg_wait_time = sum(wait_time) / n
    avg_turnaround_time = sum(turnaround_time) / n

    print(f"RR Average Waiting Time: {avg_wait_time}")
    print(f"RR Average Turnaround Time: {avg_turnaround_time}")

    return gantt_chart

# Function to display the Gantt Chart with distinct colors per process
def display_rr_gantt_chart(gantt_chart, title):
    fig, gnt = plt.subplots(figsize=(12, 6))

    gnt.set_ylim(0, 2)
    gnt.set_xlim(0, max([end for _, _, end in gantt_chart]) + 1)
    gnt.set_xlabel('Time')
    gnt.set_yticks([1])
    gnt.set_yticklabels(['Processes'])
    gnt.grid(True, axis='x')

    # Create a color map for each process
    color_map = {
        'P1': 'tab:blue',
        'P2': 'tab:green',
        'P3': 'tab:red',
        'P4': 'tab:purple',
        'P5': 'tab:orange'
    }

    # Adding the Gantt chart bars
    for process in gantt_chart:
        pid, start, end = process
        gnt.broken_barh([(start, end - start)], (0.75, 0.5),
facecolors=(color_map[f'P{pid}']))
        gnt.text((start + end) / 2, 1, f"P{pid}", ha='center',
va='center', fontsize=12, color='white')

    plt.title(title, fontsize=14)
    plt.show()

# Taking input from the user for RR scheduling
n_rr = int(input("Enter number of processes for RR: "))
quantum = int(input("Enter quantum time: "))
processes_rr = []
```

```
for i in range(n_rr):
    pid = int(input(f"Enter process ID for process {i + 1}: "))
    burst_time = int(input(f"Enter CPU burst time for process {pid}:
"))
    processes_rr.append((pid, burst_time))

gantt_chart_rr = rr_scheduling(processes_rr, quantum)
display_rr_gantt_chart(gantt_chart_rr, "Round Robin Scheduling Gantt
Chart")
```

```
Enter number of processes for RR: 4
Enter quantum time: 4
Enter process ID for process 1: 1
Enter CPU burst time for process 1: 10
Enter process ID for process 2: 2
Enter CPU burst time for process 2: 20
Enter process ID for process 3: 3
Enter CPU burst time for process 3: 4
Enter process ID for process 4: 4
Enter CPU burst time for process 4: 8
RR Average Waiting Time: 17.5
RR Average Turnaround Time: 28.0
```



Outcomes: CO2 - Demonstrate use of inter process communication

---

Questions:

Manually draw the gantt charts and show the average waiting time and turnaround time of the process.

Given Processes:

P1: Burst Time = 10

P2: Burst Time = 15

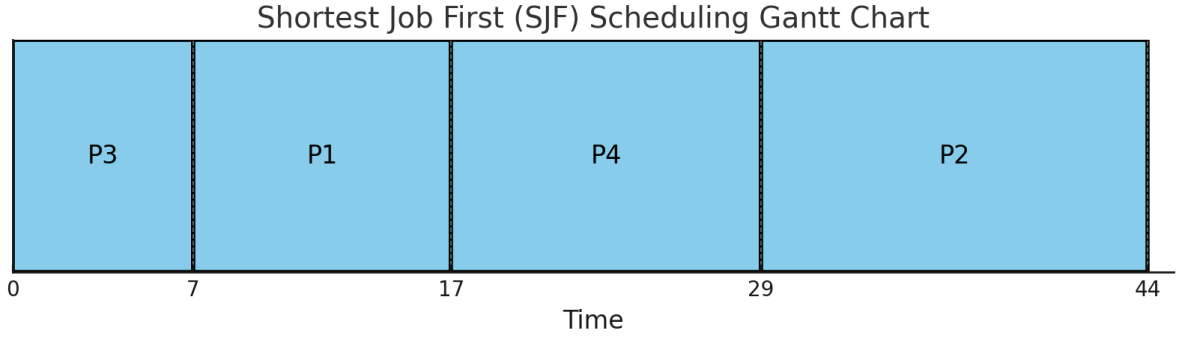
P3: Burst Time = 7

P4: Burst Time = 12

1. Shortest Job First (SJF):

   Processes (sorted by burst time):

Process	Burst Time
P3	7
P1	10
P4	12
P2	15



   Waiting Time Calculation for SJF:

   P3: 0 (since it starts first)

   P1: 7 (starts after P3 finishes)

   P4: 17 (starts after P1 finishes)

   P2: 29 (starts after P4 finishes)

   Average Waiting Time: $\frac{(0 + 7 + 17 + 29)}{4} = 13.25$

   Turnaround Time Calculation for SJF:

   P3: 7 - 0 = 7

   P1: 17 - 0 = 17

   P4: 29 - 0 = 29

   P2: 44 - 0 = 44

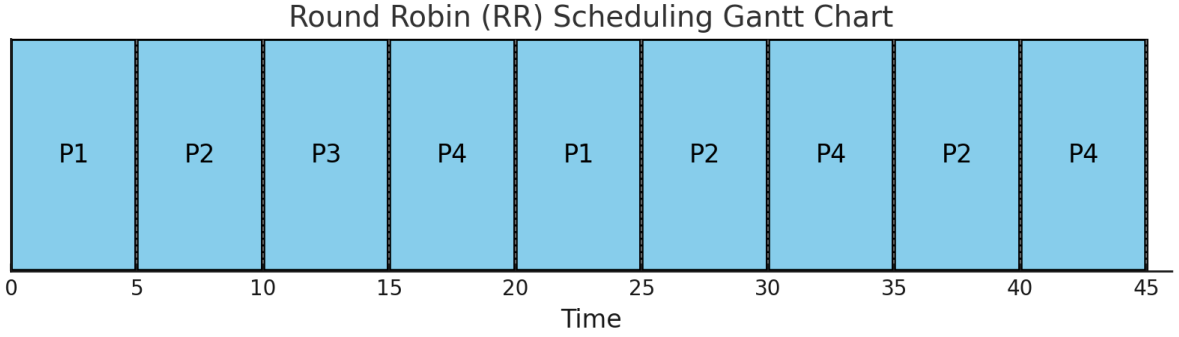
   Average Turnaround Time: $\frac{(7 + 17 + 29 + 44)}{4} = 24.25$

2. Round Robin (RR):

   Quantum Time: 5

   Processes:

Process	Burst Time
P1	10
P2	15
P3	7
P4	12



Waiting Time Calculation for RR:
P1: (25 - 5) + 0 = 20
P2: (30 - 10) + (40 - 35) + 5 = 30
P3: (30 - 15) + 10 = 25
P4: (35 - 20) + 20 = 25
Average Waiting Time: $\frac{(20 + 30 + 25 + 25)}{4} = 25$
Turnaround Time Calculation for RR:
P1: 25 - 0 = 25
P2: 45 - 0 = 45
P3: 30 - 0 = 30
P4: 40 - 0 = 40
Average Turnaround Time:
$\frac{(25 + 45 + 30 + 40)}{4} = 35$

Summary:

Metric	SJF	RR
Average Waiting Time	13.25	25
Average Turnaround Time	24.25	35

Comparison:

- SJF results in a lower average waiting and turnaround time because it prioritizes the shortest jobs, ensuring quicker completion of processes with smaller burst times.
- RR gives each process equal CPU time in a fair, cyclical manner, but results in a higher average waiting and turnaround time, especially for longer processes.

Conclusion:
This implementation allows for the simulation of the Shortest Job First (SJF) and Round Robin (RR) scheduling algorithms. We calculated and visualized the average waiting time and turnaround time for both algorithms, and the results showed that SJF minimizes average waiting time but may cause starvation for longer processes, while RR ensures fairness but requires careful tuning of the quantum size to balance efficiency and context-switching overhead.

Grade: AA/AB/BB/BC/CC/CD/DD

Signature of faculty in-charge with date

References:
Books/ Journals/ Websites:

1. Silberschatz A., Galvin P., Gagne G, “Operating Systems Concepts”, VIIIth Edition, Wiley, 2011.
