📆 Day 3: Scheduling Algorithms (CSIS 2260 Preview)

🎯 Goals

- Understand the purpose of CPU scheduling in multitasking environments.
- Learn three major scheduling algorithms: FCFS, SJF, and Round Robin.
- Visualize how scheduling affects CPU efficiency and process wait times.
- Practice building Gantt charts and simulations.

📘 Topics to Cover

✅ Why Scheduling Matters

- The OS decides **which process gets CPU time** and **when**.
- A good scheduler:
    - Maximizes CPU utilization.
    - Minimizes waiting and turnaround time.
    - Ensures fairness among processes.

✅ Key Scheduling Algorithms

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

    - Processes are handled in the order they arrive.
    - Simple, but can cause **convoy effect** (long waits behind big jobs).

2. **Shortest Job First (SJF)**

    - Executes the process with the shortest burst time first.

    - Optimal in terms of average waiting time, but hard to predict burst time.

3. **Round Robin (RR)**
    - Each process gets a fixed time slice (quantum).
    - Preemptive scheduling—great for time-sharing systems.

▶️ Watch (Optional, 15–20 min)

- CPU Scheduling Basics
- Shortest Job First Scheduling (Solved Problem 1)

✍️ Do

📊 1. Gantt Chart Practice

- Draw a Gantt chart for each algorithm using the following sample:

    1. FCFS

```
Process | Arrival| Burst | Turnaround | Waiting
--------|--------|-------|------------|---------
P1      |   0    |   5   | 5 - 0 = 5  | 5 - 5 = 0
P2      |   1    |   3   | 8 - 1 = 7  | 7 - 3 = 4
P3      |   2    |   1   | 9 - 2 = 7  | 7 - 1 = 6
P4      |   3    |   2   | 11 - 3 = 8 | 8 - 2 = 6

Gantt Chart
0 P1 5 P2 8 P3 9 P4 11
Average waiting time = ( 0 + 4 + 6 + 6 ) / 4 = 3.5
```

    2. SRTF - Shortest Remaining Time First, Preemptive SJF

- It’s definitely worth understanding, even if not common in Windows/Linux kernels.

```
Process | Arrival| Burst | Turnaround | Waiting
--------|--------|-------|------------|---------
P1      |   0    |   5   | 11 - 0 = 11| 11 - 5 = 6
P2      |   1    |   3   | 7 - 1 = 6  | 6 - 3 = 3
P3      |   2    |   1   | 3 - 2 = 1  | 1 - 1 = 0
P4      |   3    |   2   | 5 - 3 = 2  | 2 - 2 = 0

Gantt Chart
0 P1 1 P2 2 P3 3 P4 5 P2 7 P1 11
Average waiting time = ( 6 + 3 + 0 + 0 ) / 4 = 2.25
```

3. SJF - Non-Preemptive

```
Process | Arrival| Burst | Turnaround | Waiting
--------|--------|-------|------------|---------
P1      |   0    |   5   | 5 - 0 = 5  | 5 - 5 = 0
P2      |   1    |   3   | 11 - 1 = 10| 10 - 3 = 7
P3      |   2    |   1   | 6 - 2 = 4  | 4 - 1 = 3
P4      |   3    |   2   | 8 - 3 = 5  | 5 - 2 = 3

Gantt Chart
0 P1 5 P3 6 P4 8 P2 11
Average waiting time = ( 0 + 7 + 3 + 3 ) / 4 = 3.25
```

4. RR

```
Process | Arrival| Burst | Turnaround | Waiting
--------|--------|-------|------------|---------
P1      |   0    |   5   | 11 - 0 = 11| 11 - 5 = 6
P2      |   1    |   3   | 10 - 1 = 9 | 9 - 3 = 6
```

```
P3       |   2    |   1   | 5 - 2 = 3  | 3 - 1 = 2
P4       |   3    |   2   | 9 - 3 = 6  | 6 - 2 = 4

Gantt Chart
0 P1 2 P2 4 P3 5 P1 7 P4 9 P2 10 P1 11
Average waiting time = ( 6 + 6 + 2 + 4 ) / 4 = 4.5
```

- Try FCFS, SJF (non-preemptive), and Round Robin (quantum = 2).
- Calculate:
  - Turnaround Time = Completion Time – Arrival Time
  - Waiting Time = Turnaround Time – Burst Time

🔧 2. Spreadsheet or Code Simulation (Optional)

- Use Excel, Google Sheets, or a Python/JavaScript snippet to simulate Round Robin logic.
- Practice it with function
- Practice it with Class

```python
class Process:
    def __init__(self, pid, arrival_time, burst):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst = burst
        self.remaining_time = burst
        self.is_visited = False
        self.completion_time = 0
        self.turnaround_time = 0
        self.waiting_time = 0



class RoundRobinScheduler:
    def __init__(self, processes, quantum):
        self.processes = processes
        self.quantum = quantum
        self.time = 0
        self.queue = []
        self.gantt_chart = []

    def is_finished(self):
        return all(p.remaining_time == 0 for p in
        self.processes)

    def is_newly_arrived(self, p):
        return not p.is_visited and p.arrival_time <=
        self.time

    def enqueue_arrived_processes(self):
        for process in self.processes:
            if self.is_newly_arrived(process):
                self.queue.append(process)
                process.is_visited = True

    def run(self) -> None:
        while not self.is_finished():
            # enqueue
            self.enqueue_arrived_processes()
            # wait
            if not self.queue:
                self.time += 1
                continue
            # pop first queue
            current = self.queue.pop(0)
            # CPU running
            self.gantt_chart.append((self.time, current.pid))
            # Update
            if current.remaining_time > self.quantum:
                self.time += self.quantum
                current.remaining_time -= self.quantum
            else:
                self.time += current.remaining_time
                current.remaining_time = 0
                current.completion_time = self.time
            # Refresh : enqueue current process after new arrival
            self.enqueue_arrived_processes()
            if current.remaining_time > 0:
                self.queue.append(current)
        # wrap up
        self.gantt_chart.append((self.time, "Finished"))
```

✅ Checkpoint

- I can describe how each scheduling algorithm works.
- I can draw a correct Gantt chart for FCFS, SJF, and RR.
- I calculated waiting and turnaround time for a process.
- I understand how different algorithms affect performance.