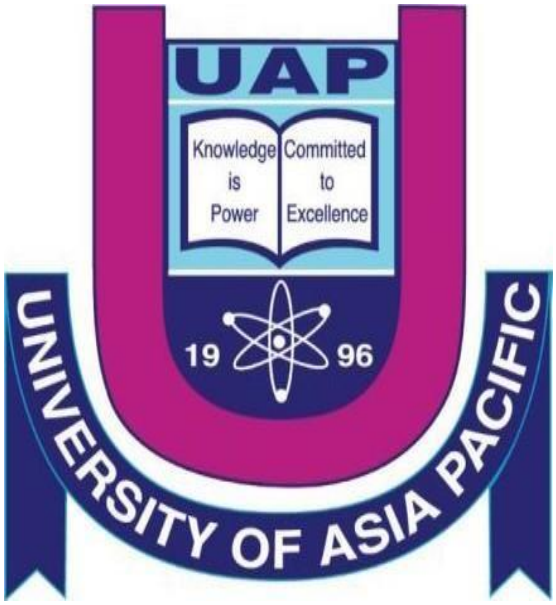

Department of Computer Science & Engineering

Course Title: Operating System Lab

Course Code: CSE 406

Lab Report No: 02

Lab Report: Shortest Job First (SJF) Scheduling

Submission Date: 19-02-2025

Submitted To:
Atia Rahman Orthi
Lecturer,
Department of CSE, UAP

Submitted By:
Name: Amirul Islam Papon
Reg No: 21201076
Sec: B

Problem Statement: The Shortest Job First (SJF) scheduling algorithm selects the process with the shortest CPU burst time first, reducing the overall waiting time and improving system efficiency. We aim to implement the Non-Preemptive SJF Scheduling Algorithm, calculate the completion, turnaround, and waiting times, and analyze its performance.

Steps:

1. **Define the Process Data:**
   - Input consists of process ID, arrival time, and burst time.
2. **Sort the Processes:**
   - Select the process with the shortest burst time among available processes (those that have arrived at or before the current time).
   - If multiple processes have the same burst time, select the one that arrived first.
3. **Schedule Execution:**
   - Assign the selected process to the CPU and update the current time.
   - Record the completion time, then calculate the turnaround and waiting times.
4. **Repeat Until All Processes Are Scheduled:**
   - Continue selecting and executing the shortest available job until all processes are completed.
5. **Display Results:**
   - Print process scheduling order.
   - Compute and display completion time, turnaround time, and waiting time for each process.

Source Code:

shortest_first_job.py > calculate_sjf

```python
def calculate_sjf():
    n = len(process_data)

    current_time = 0
    completed = []
    execution_order = []
    available = []

    while len(completed) < n:
        # Add newly arrived processes
        for process in process_data:
            if process[1] <= current_time and process not in completed and process not in available:
                available.append(process)

        if available:
            (variable) execution_order: list  # Sort by burst time
            execution_order.append(current_process[0])
            current_time += current_process[2]
            completed.append(current_process)
        else:
            current_time += 1            # If no process is available, increment time

    # Calculate Completion, Turnaround, and Waiting times
    completion_times = {}
    waiting_times = {}
    turnaround_times = {}

    current_time = 0
    for process in execution_order:
        for p, at, bt in process_data:
            if p == process:
                if current_time < at:
                    current_time = at
                completion_times[p] = current_time + bt
                turnaround_times[p] = completion_times[p] - at
                waiting_times[p] = turnaround_times[p] - bt
                current_time = completion_times[p]
                break

    # Print results
    print("Process\tArrival\tBurst\tCompletion\tTurnaround\tWaiting")
    for p, at, bt in process_data:
        print(f"{p}\t\t{at}\t\t{bt}\t\t{completion_times[p]}\t\t\t{turnaround_times[p]}\t\t{waiting_times[p]}")

# Input data
process_data = [
    ('P1', 2, 6),
    ('P2', 5, 2),
    ('P3', 1, 8),
    ('P4', 0, 3),
    ('P5', 4, 4)
]
```

Output:

```
Process Arrival Burst   Completion  Turnaround  Waiting
P1      2       6       9           7       1
P2      5       2       11          6       4
P3      1       8       23          22      14
P4      0       3       3           3       0
P5      4       4       15          11      7

[Done] exited with code=0 in 0.06 seconds
```

Discussion and Conclusion: The SJF scheduling algorithm efficiently reduces the average waiting time and turnaround time, making it an optimal scheduling technique for batch processing systems. However, SJF has limitations:

- Starvation Issue: Longer processes may suffer from indefinite waiting if shorter jobs keep arriving.
- Non-Preemptive Limitation: Once a process starts execution, it cannot be interrupted, which may not be ideal for real-time systems.