📅 Week 7 – CSIS 2260 (Operating Systems) – Survey Week with Practice

Day 3 – Processes & Threads

🔍 Concepts

- **Process**: An instance of a program in execution, with its own memory space.
- **Thread**: A smaller unit of execution within a process, sharing the same memory.
- **Process States**: new → ready → running → waiting → terminated.
- **Process Control Block (PCB)**: Data structure storing process info (ID, state, registers, memory pointers).
- **Multithreading vs Multiprocessing**:
  - *Multithreading*: Multiple threads within the same process.
  - *Multiprocessing*: Multiple processes running independently.

📖 Theory Notes

- **Context Switching**: Saving the state of a process and loading the state of another.
- **Advantages of Threads**:
  - Faster creation than processes.
  - Shared memory between threads.
- **Disadvantages**:
  - Bugs in one thread can affect others in the same process.

✍️ Pseudocode Test

```
PROCESS start → NEW
NEW → READY
READY → RUNNING
RUNNING → WAITING or TERMINATED
```

🧪 Code Simulation (Python Example)

```python
states = ["NEW", "READY", "RUNNING", "WAITING", "TERMINATED"]
current_state = 0

transitions = {
    "NEW": "READY",
    "READY": "RUNNING",
    "RUNNING": "WAITING",
    "WAITING": "READY",
    "RUNNING_END": "TERMINATED"
}

def move_state(state, end=False):
    if end and state == "RUNNING":
        return transitions["RUNNING_END"]
    return transitions.get(state, state)

# Simulate process
state = "NEW"
print(f"Initial: {state}")
state = move_state(state)
print(f"After allocation: {state}")
state = move_state(state)
print(f"Process starts: {state}")
state = move_state(state)  # simulate waiting
```

```
print(f"Waiting: {state}")
state = move_state(state)
print(f"Back to: {state}")
state = move_state("RUNNING", end=True)
print(f"Final: {state}")
```

🧪 Code Simulation (JavaScript Example)

```
const transitions = {
  NEW: "READY",
  READY: "RUNNING",
  RUNNING: "WAITING",
  WAITING: "READY",
  RUNNING_END: "TERMINATED"
};

function moveState(state, end = false) {
  if (end && state === "RUNNING") {
    return transitions.RUNNING_END;
  }
  return transitions[state] || state;
}

let state = "NEW";
console.log(`Initial: ${state}`);
state = moveState(state);
console.log(`After allocation: ${state}`);
state = moveState(state);
console.log(`Process starts: ${state}`);
state = moveState(state);
console.log(`Waiting: ${state}`);
state = moveState(state);
console.log(`Back to: ${state}`);
```

```
state = moveState("RUNNING", true);
console.log(`Final: ${state}`);
```

🧪 Code Simulation (C# Example)

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        var transitions = new Dictionary<string, string>
        {
            {"NEW", "READY"},
            {"READY", "RUNNING"},
            {"RUNNING", "WAITING"},
            {"WAITING", "READY"},
            {"RUNNING_END", "TERMINATED"}
        };

        string MoveState(string state, bool end = false)
        {
            if (end && state == "RUNNING")
                return transitions["RUNNING_END"];
            return transitions.ContainsKey(state) ?
        transitions[state] : state;
        }

        string state = "NEW";
        Console.WriteLine($"Initial: {state}");
        state = MoveState(state);
        Console.WriteLine($"After allocation: {state}");
```

```
        state = MoveState(state);
        Console.WriteLine($"Process starts: {state}");
        state = MoveState(state);
        Console.WriteLine($"Waiting: {state}");
        state = MoveState(state);
        Console.WriteLine($"Back to: {state}");
        state = MoveState("RUNNING", true);
        Console.WriteLine($"Final: {state}");
    }
}
```

❓ Quiz

1. What's the difference between a process and a program?
2. Which state means "waiting for I/O"?
3. What is context switching and why is it needed?

✅ Deliverable for Day 3

- Notes on processes, threads, and states.
- Completed pseudocode test.
- Working code simulations in Python, JavaScript, and C#.
- Quiz answers recorded.