20CYS402 – Distributed Systems & Cloud Computing

Lab 4 - Edge-Chasing Distributed Deadlock Detection Algorithm

---

Name: Chitra Harini **Date:** 06/08/2025

Roll no: CH.EN.U4CYS22010 **Lab - 4**

Github Link: [20CYS402-Distributed-Systems-Cloud-Computing/LAB4 at main · Harini-chitra/20CYS402-Distributed-Systems-Cloud-Computing]

Objective:

The objective of this lab is to **implement the Edge-Chasing Distributed Deadlock Detection Algorithm** for detecting deadlocks in distributed systems.
The algorithm works by sending **probe messages** across the **Wait-For Graph (WFG)**. If a probe returns to the **initiating process**, it means that a cycle exists in the graph, and thus a **deadlock is detected**.

Code Implementation:

```
# Edge-Chasing Distributed Deadlock Detection Algorithm
class Process:
    def __init__(self, pid):
        self.pid = pid
        self.waiting_for = []  # list of processes this process waits for
    def add_dependency(self, process):
        """Process waits for another process (edge in wait-for graph)."""
        self.waiting_for.append(process)
class EdgeChasingDeadlockDetector:
    def __init__(self, processes):
        self.processes = processes
        self.deadlock_detected = False
    def send_probe(self, initiator, current, visited):
        """
        Send probe messages recursively.
        If probe returns to initiator, deadlock exists.
        """
        if current in visited:
            return  # avoid infinite recursion in traversal
        visited.add(current)
        for neighbor in current.waiting_for:
            print(f"Probe: {initiator.pid} -> {neighbor.pid} (from {current.pid})")
            # Deadlock detected if probe reaches initiator again
            if neighbor == initiator:
                print(f"Deadlock detected! Cycle found at process {initiator.pid}.")
                self.deadlock_detected = True
                return
            # Continue probing
            self.send_probe(initiator, neighbor, visited.copy())
    def detect_deadlock(self):
        """Initiate probe from each process."""
        for process in self.processes:
            print(f"\nInitiating probe from Process {process.pid}")
            self.send_probe(process, process, set())
```

```
        if not self.deadlock_detected:
            print("\nNo deadlock detected.")

# ------------------- Example Input -------------------
if __name__ == "__main__":
    # Create processes
    p1 = Process(1)
    p2 = Process(2)
    p3 = Process(3)
    p4 = Process(4)

    # Define dependencies (Wait-For Graph edges)
    p1.add_dependency(p2)   # P1 waits for P2
    p2.add_dependency(p3)   # P2 waits for P3
    p3.add_dependency(p1)   # P3 waits for P1  (Cycle formed here: P1 -> P2 -> P3 -> P1)
    p4.add_dependency(p2)   # P4 waits for P2 (No cycle with P4)

    # Run Deadlock Detection
    detector = EdgeChasingDeadlockDetector([p1, p2, p3, p4])
    detector.detect_deadlock()
```

Working of the Algorithm:

1. Wait-For Graph (WFG) is created:
    - Nodes represent processes.
    - Edges represent dependency (P1 → P2 means P1 is waiting for P2).
2. Probe messages are initiated by each process to detect cycles.
3. If the probe returns to the initiating process, it means there is a cycle → deadlock detected.
4. If no such cycle is detected, the system is deadlock-free.

Input:

Processes: P1, P2, P3, P4

Dependencies:

- P1 → P2
- P2 → P3
- P3 → P1 (Cycle)
- P4 → P2

Output:

Initiating probe from Process 1
Probe: 1 -> 2 (from 1)
Probe: 1 -> 3 (from 2)
Probe: 1 -> 1 (from 3)
Deadlock detected! Cycle found at process 1.

Initiating probe from Process 2
Probe: 2 -> 3 (from 2)
Probe: 2 -> 1 (from 3)
Probe: 2 -> 2 (from 1)
Deadlock detected! Cycle found at process 2.

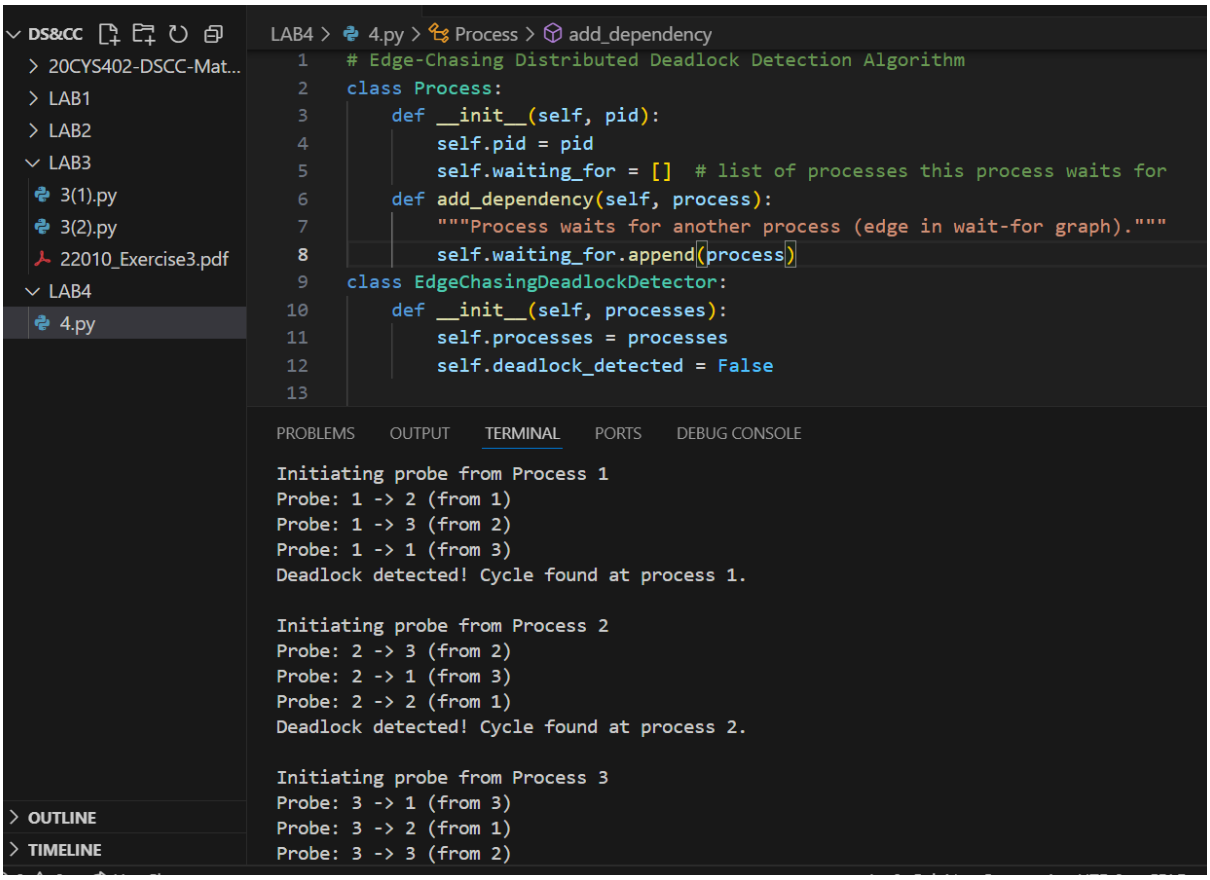
Initiating probe from Process 3
Probe: 3 -> 1 (from 3)
Probe: 3 -> 2 (from 1)
Probe: 3 -> 3 (from 2)
Deadlock detected! Cycle found at process 3.

Initiating probe from Process 4
Probe: 4 -> 2 (from 4)

Probe: 4 -> 3 (from 2)
Probe: 4 -> 1 (from 3)
Probe: 4 -> 2 (from 1)
Deadlock detected! Cycle found at process 4.

Deadlock detected because of the cycle: P1 → P2 → P3 → P1.

Screenshot:



Conclusion:

The **Edge-Chasing Distributed Deadlock Detection Algorithm** is a **decentralized approach** where processes send **probe messages** through the **wait-for graph** to detect cycles.

- If a probe returns to the initiator, it confirms a **deadlock**.
- This algorithm is efficient for distributed systems as it does not rely on a central coordinator.
- Our simulation successfully detected a cycle (deadlock) among processes P1, P2, and P3.