20CYS402 — *Distributed Systems and Cloud Computing*

Lab Exercise – 4: Edge-Chasing Distributed Deadlock Detection Algorithm

Student Name: R Subramanian

Roll No.: CH.EN.U4CYS22043

Date: 06/08/2025

Title: Chandy–Misra–Hass Deadlock Detection Algorithm

Objective of the Exercise

The objective of this lab exercise is to **implement the Chandy–Misra–Hass distributed deadlock detection algorithm**, which detects deadlocks in a distributed system using a **probe-based edge-chasing technique**.

This decentralized algorithm allows processes to detect cycles by sending probe messages without maintaining or sharing a global resource allocation graph.

Code Implementation

```
class Process:
    def __init__(self, pid):
        self.pid = pid
        self.waiting_for = []
        self.blocked = False

    def receive_probe(self, initiator, sender, receiver):
        print(f"Probe received at P{self.pid} → (initiator={initiator}, sender={sender}, receiver={receive
r})")
        if self.pid == initiator:
            print(f"🔴 Deadlock detected! Probe returned to initiator P{self.pid}")
            return True
        if not self.blocked or not self.waiting_for:
            return False
        for next_receiver in self.waiting_for:
            if processes[next_receiver].receive_probe(initiator, self.pid, next_receiver):
                return True
        return False

    def initiate_detection(self):
        print(f"\n[Initiating Deadlock Detection from Process P{self.pid}]")
        deadlock = False
        if self.blocked:
            for next_receiver in self.waiting_for:
                if processes[next_receiver].receive_probe(self.pid, self.pid, next_receiver):
                    deadlock = True
            break
    if not deadlock:
        print("✅ No deadlock detected.")


processes = {}
num_processes = int(input("Enter the number of processes: "))
for i in range(num_processes):
    processes[i] = Process(i)

print("Enter the number of wait-for relations: ")
num_edges = int(input())
print("Enter waiting pairs (A B): Process A is waiting for process B")
for _ in range(num_edges):
    a, b = map(int, input().split())
    processes[a].waiting_for.append(b)
    processes[a].blocked = True

initiator_id = int(input("Enter the initiator process ID: "))
processes[initiator_id].initiate_detection()
```

Explanation of the Code

1. **Process Class**
   - Represents each distributed process with a unique ID.
   - Maintains a list of other processes it is waiting for.
   - The `receive_probe()` method simulates the propagation of a probe to detect a possible cycle.
2. **Probe Message**
   - Each probe is represented as a triplet: **(initiator, sender, receiver)**.
   - If the probe returns to the **initiator**, it confirms the presence of a **deadlock**.
3. **Initiator**
   - A **blocked process** initiates detection by sending probe messages to all processes it is waiting for.
   - The probe travels along the **wait-for chain** across the system.
4. **Detection Logic**
   - If a probe completes a cycle and returns to the initiator, a **deadlock** is detected.
   - If no cycle is found, the system reports **"No deadlock detected."**

Input / Output Examples

Example 1

Input:

```
Enter the number of processes: 3
Enter the number of wait-for relations: 3
Enter waiting pairs (A B):
```

```
0 1
1 2
2 0
Enter the initiator process ID: 0
```

Output:

```
[Initiating Deadlock Detection from Process P0]
Probe received at P1 → (initiator=0, sender=0, receiver=1)
Probe received at P2 → (initiator=0, sender=1, receiver=2)
Probe received at P0 → (initiator=0, sender=2, receiver=0)
🔴 Deadlock detected! Probe returned to initiator P0
```

Example 2

Input:

```
Enter the number of processes: 3
Enter the number of wait-for relations: 2
Enter waiting pairs (A B):
0 1
1 2
Enter the initiator process ID: 0
```

Output:

```
[Initiating Deadlock Detection from Process P0]
Probe received at P1 → (initiator=0, sender=0, receiver=1)
Probe received at P2 → (initiator=0, sender=1, receiver=2)
✅ No deadlock detected.
```

Conclusion

This lab demonstrated the **Chandy–Misra–Hass algorithm** for detecting distributed deadlocks using **probe-based edge chasing**.

The algorithm effectively identifies cycles in a distributed environment **without requiring centralized control or global state knowledge**.

Through simulation, we observed how probe messages are propagated and how returning probes signify the presence of a deadlock.
