

**Vidyavardhini’s College of Engineering & Technology**

Department of Computer Engineering Academic Year : 2024-25

|  |  |  |  |
| --- | --- | --- | --- |
| **Class:** | **BE** | **Semester:** | **VIII** |
| **Course Code:** | **CSL801** | **Course Name:** | **Distributed Computing Lab** |

|  |  |
| --- | --- |
| **Name of Student:** | **Pratima Dinkar Bombe** |
| **Roll No. :** | **07** |
| **Division:** | **-** |
| **Experiment No.:** | **07** |
| **Title of Experiment:** | **Deadlock Management in Distributed Systems** |
| **Date of Submission:** | **25/02/2025** |
| **Date of Correction:** | **04/03/2025** |

Evaluation

|  |  |  |
| --- | --- | --- |
| **Performance Indicator** | **Max. Marks** | **Marks Obtained** |
| Performance | 5 |  |
| Understanding | 5 |  |
| Journal work and timely submission | 10 |  |
| Total | 20 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Performance Indicator** | **Exceed Expectations (EE)** | **Meet Expectations (ME)** | **Below Expectations (BE)** |
| Performance | 4-5 | 2-3 | 1 |
| Understanding | 4-5 | 2-3 | 1 |
| Journal work and timely submission | 8-10 | 5-8 | 1-4 |

**Checked by**

**Name of Faculty : Ms. Swati Varma**

**Signature :**

**Date :**

EXPERIMENT 7

**AIM:** To implement deadlock management in Distributed System

**Objective:** Develop a program to implement deadlock management in distributed system

**THEORY:**

The algorithm allows a process to request for multiple resources at a time. Special messages or probes circulated along edges of WFG. Deadlock exists if the probe is received back by the initiator.

Working:

1. When a process that requests for a resource, fails to get it and times out,
2. It generates a special probe message and sends it to the process (or processes) holding the requested resource.
   * The probe message (acts as a unique id) contains the following fields:
   * The id of the process just blocked
   * The id of the process sending this message
   * The id of the process to whom this message is being sent
3. On receipt of the probe message, the receiver checks to see if it itself is waiting for any resource.
4. If not, this means that the recipient is using the resource requested by the process that sent the probe message to it.
5. In this case, the recipient simply ignores the probe message.
6. On the other hand, if the recipient is itself waiting for any resource, it passes the probe message to the resource holding process (or processes)
7. But, before the probe is forwarded, the recipient does following modifications
8. The first field is left unchanged
9. The recipient changes the second field to its own process id.
10. The third field is changed to the id of the process that will be the new recipient of the probe.
11. Every new recipient of the probe message repeats this procedure.
12. If the probe message returns back to the original sender,( the process whose id is the first field of the message), a cycle exists and, the system is deadlocked.

**Code and output**:

class Process:

    def \_\_init\_\_(self, pid):

        self.pid = pid  # Process ID

        self.waiting\_for = []  # List of processes it is waiting for

    def request\_resource(self, resource\_holders, initiator):

        """Initiates a probe if the resource request fails."""

        if not resource\_holders:

            print(f"Process {self.pid} found no resource holders.")

            return False  # No resource holders, no deadlock

        for holder in resource\_holders:

            probe = [initiator, self.pid, holder.pid]  # Probe message

            print(f"Process {self.pid} sending probe {probe} to {holder.pid}")

            if holder.receive\_probe(probe):

                print(f"Deadlock detected involving process {initiator}!")

                return True

        return False

    def receive\_probe(self, probe):

        """Handles received probe messages."""

        initiator, sender, receiver = probe

        print(f"Process {self.pid} received probe {probe}")

        if self.pid == initiator:  # Cycle detected

            print(f"Cycle detected! Process {self.pid} is in a deadlock.")

            return True

        if not self.waiting\_for:  # Not waiting for any process

            print(f"Process {self.pid} is not waiting for any resource. Ignoring probe.")

            return False  # No deadlock

        for next\_holder in self.waiting\_for:

            new\_probe = [initiator, self.pid, next\_holder.pid]  # Modify probe

            print(f"Process {self.pid} forwarding probe {new\_probe} to {next\_holder.pid}")

            if next\_holder.receive\_probe(new\_probe):

                return True

        return False

# Example Usage

print("--- Deadlock Example ---")

p1 = Process(1)

p2 = Process(2)

p3 = Process(3)

p4 = Process(4)

# Defining dependencies (waits-for relationships) - Deadlock case

p1.waiting\_for = [p2]

p2.waiting\_for = [p3]

p3.waiting\_for = [p4]

p4.waiting\_for = [p1]  # Creates a cycle (deadlock)

# Initiate probe from p1

if p1.request\_resource([p2], p1.pid):

    print("Deadlock confirmed.")

else:

    print("No deadlock detected.")

print("\n--- No Deadlock Example ---")

p5 = Process(5)

p6 = Process(6)

p7 = Process(7)

p8 = Process(8)

# Defining dependencies - No cycle

p5.waiting\_for = [p6]

p6.waiting\_for = [p7]

p7.waiting\_for = [p8]

p8.waiting\_for = []  # No cycle

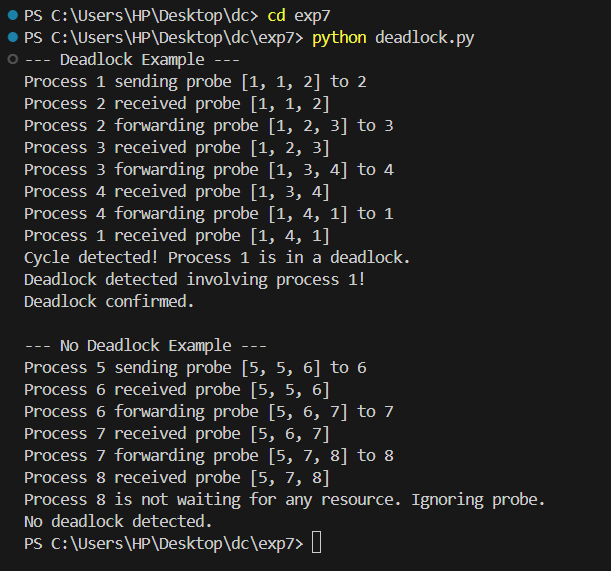
# Initiate probe from p5

if p5.request\_resource([p6], p5.pid):

    print("Deadlock confirmed.")

else:

    print("No deadlock detected.")



**Conclusion**: The algorithm detects deadlocks by circulating probe messages along the Wait-For Graph (WFG). If a probe returns to its initiator, a cycle is detected, confirming deadlock. Otherwise, if the probe reaches a process that is not waiting for any resource, it is ignored, indicating no deadlock.