|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

BSA Activity

|  |  |
| --- | --- |
| Semester | B.E. Semester VIII – Computer Engineering |
| Subject | Distributed Computing Lab |
| Subject Professor In-charge | Dr. Umesh Kulkarni |
| Assisting Professor | Prof. Prakash Parmar |
| Academic Year | 2024-25 |

|  |  |
| --- | --- |
| Student Name | Deep Salunkhe |
| Roll Number | 21102A0014 |

**Title:** Answers based on videos

**No-Token-Based Algorithm in Distributed Computing**

No-token-based algorithms are used for mutual exclusion in distributed systems without relying on a token. They work based on timestamps or logical ordering. Two main types of no-token-based algorithms are:

1. **Ricart-Agrawala Algorithm (RA Algorithm)**
   * Each process that wants to enter the critical section (CS) sends a request message to all other processes.
   * Each request message contains a **timestamp** and the process ID.
   * A process can enter the CS only when it has received permission (REPLY message) from all other processes.
   * If a process receives a request while in the CS, it delays the response until it exits.
   * If it’s not in the CS but has already requested it, it compares timestamps and grants permission to the process with the smaller timestamp.
2. **Maekawa’s Algorithm**
   * It reduces the number of messages by dividing all processes into **quorum sets**.
   * A process requesting CS needs permission from only its quorum members.
   * Deadlocks can occur if two processes wait for each other in overlapping quorums, requiring a timeout mechanism.

**Advantages**:

* No token loss issue.
* Fair access to the CS.

**Disadvantages**:

* High message complexity in RA (2(N−1) messages per CS request).
* Maekawa’s algorithm may lead to deadlocks.

**Methods of Deadlock Detection and Prevention**

Deadlocks in distributed systems occur when processes wait indefinitely for resources held by others. There are three primary methods:

**1. Deadlock Prevention**

* Prevents deadlocks before they occur by breaking one of the necessary conditions for deadlock:
  + **Mutual Exclusion:** Use sharable resources where possible.
  + **Hold and Wait:** Ensure processes request all needed resources at once.
  + **No Preemption:** Allow resources to be preempted if needed.
  + **Circular Wait:** Impose a global ordering of resource requests.

**2. Deadlock Detection and Recovery**

* Allows deadlocks to occur but detects and resolves them.
* Uses **Wait-For Graphs (WFG)**, where a cycle in the graph indicates deadlock.
* Recovery methods include:
  + **Process Termination** (abort processes to break the cycle).
  + **Resource Preemption** (forcefully take resources from some processes).

**3. Deadlock Avoidance**

* Uses algorithms like **Banker’s Algorithm** to ensure the system remains in a safe state.
* Requires each process to declare its maximum resource need in advance.

**Correction Issues in Deadlock Methods**

When handling deadlocks, various challenges arise:

1. **Incorrect Deadlock Detection**
   * If message delays occur, the system may mistakenly detect a deadlock.
   * Solution: Use **timeout-based confirmation** before terminating a process.
2. **Overhead in Deadlock Prevention**
   * Preventing deadlocks entirely may reduce system utilization (e.g., forcing processes to request all resources at once).
   * Solution: Use a hybrid approach, where critical resources use deadlock avoidance, while others use detection.
3. **Process Starvation**
   * In recovery-based approaches, certain processes may always be selected for termination.
   * Solution: Ensure fairness by using priority-based selection.
4. **Difficulty in Distributed Deadlock Detection**
   * Due to the **absence of a central coordinator**, tracking deadlocks across multiple nodes is complex.
   * Solution: Use **hierarchical or distributed detection** methods.

**Different Models in Deadlocks**

Deadlocks can occur in different system models:

**1. Centralized Deadlock Model**

* A single process or coordinator maintains a **global resource allocation table**.
* It detects cycles in the **Wait-For Graph** to identify deadlocks.
* **Example**: A central resource manager in a cloud-based database.

**2. Distributed Deadlock Model**

* Each process maintains its own **partial information** about dependencies.
* Deadlock detection messages are exchanged between nodes.
* **Example**: A multi-server transaction system where different servers hold different parts of a database.

**3. Hierarchical Deadlock Model**

* The system is structured in levels, with **local** and **global** deadlock detection.
* Local detectors check for deadlocks in their domains and report to the global detector.
* **Example**: Distributed banking systems where different banks handle local transactions but report conflicts to a central authority.