Assignment No. 05

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: Token-Based Algorithms and Deadlock Handling

Introduction

Token-based algorithms are widely used in distributed systems to ensure mutual exclusion in resource allocation. These algorithms employ a unique token that grants the holder the right to access a critical section, thereby preventing conflicts in a multi-process environment.

This assignment explores two key token-based algorithms—**Suzuki-Kasami's Broadcast Algorithm** and **Raymond's Tree-Based Algorithm**—and analyzes how deadlocks might arise in such scenarios. Furthermore, it discusses deadlock detection and resolution strategies using the **Chandy-Misra-Haas Algorithm**.

---

1. Token-Based Algorithms

1.1 Suzuki-Kasami's Broadcast Algorithm

Overview

Suzuki-Kasami's algorithm is a token-based mutual exclusion algorithm that efficiently handles multiple requests in a distributed system. The key idea is to use a **single token** to grant access to a critical section. Processes send requests to all other processes in the system, and the token is passed based on request timestamps.

Working Mechanism

1. **Requesting Access:**
   - A process sends a request to all other processes.
   - The request includes the requesting process's ID and sequence number.
2. **Receiving Requests:**
   - Each process maintains a request queue.
   - If a process does not have the token, it forwards the request to the token holder.
3. **Granting Access:**
   - When a process holding the token receives a request, it checks its request queue.
   - If another process has a higher priority (based on sequence number), it forwards the token to that process.
4. **Releasing the Token:**
   - After completing execution in the critical section, the process updates its request queue and passes the token to the next eligible process.

Advantages

- Efficient in systems with **high contention**.
- Reduces message complexity **to O(N)** (where N is the number of processes).

Disadvantages

- If the token is **lost**, recovery mechanisms are required.
- Broadcasting requests to all processes may introduce overhead.

1.2 Raymond's Tree-Based Algorithm

Overview

Raymond's algorithm optimizes Suzuki-Kasami's approach by structuring processes into a logical **tree hierarchy**. Instead of broadcasting requests, it routes them along the tree, reducing the number of messages exchanged.

Working Mechanism

1. **Tree Structure:**
   - Each process knows its **parent** and **children** in the tree.
   - The token resides at a particular node in the tree.
2. **Requesting Access:**
   - If a process requires access, it sends a request to its parent.
   - Requests are forwarded up the tree until they reach the token holder.
3. **Token Passing:**
   - The token moves down the request path until it reaches the requesting process.
4. **Releasing the Token:**
   - The process completes execution and checks if any other requests exist.
   - If yes, the token is passed to the next requesting process.
   - If no, the process holds onto the token until a new request arrives.

Advantages

- **Reduces message complexity** compared to Suzuki-Kasami's algorithm (**O(log N)** in ideal conditions).
- The structured tree eliminates unnecessary broadcasts.

Disadvantages

- **Single failure point:** If a parent node crashes, it may cause delays.
- The structure must be dynamically maintained if processes join or leave.

---

2. Deadlocks in Token-Based Systems

A deadlock occurs when two or more processes wait indefinitely for a condition that will never be satisfied. In token-based algorithms, deadlocks can arise due to:

1. **Token Loss:** If a process holding the token crashes, other processes remain blocked indefinitely.
2. **Circular Wait:** If requests form a cyclic dependency, processes wait for each other, leading to a deadlock.
3. **Incorrect Token Forwarding:** If a token is misrouted or an incorrect request is served first, the system can enter a deadlock state.

---

3. Deadlock Detection and Resolution

3.1 Chandy-Misra-Haas Algorithm

The **Chandy-Misra-Haas Algorithm** is used for deadlock detection in distributed systems. It is based on a **wait-for graph (WFG)**, where nodes represent processes and directed edges indicate waiting relationships.

Working Mechanism

1. **Constructing the WFG:**
   - Each process records dependencies (which process is waiting for whom).
   - Processes periodically exchange this information.
2. **Deadlock Detection Using Probe Messages:**
   - A process initiates a deadlock check if it is waiting for a resource for an extended period.
   - It sends a **probe message** to the process it is waiting for.
   - The recipient forwards the probe to its own dependency.
   - If a probe returns to the initiator, a **cycle** is detected, indicating a deadlock.
3. **Resolving Deadlocks:**
   - The system selects a victim process (e.g., based on priority, time waited, or minimal disruption) and **forces it to release the token**.

    - The token is then reassigned or regenerated to restore operation.
    - Recovery mechanisms ensure processes can rejoin the system after failure.

Advantages

- Efficient detection without global synchronization.
- Works well in **dynamic environments** with changing process dependencies.

Disadvantages

- Requires extra messaging overhead for probe circulation.
- Delays in detection if probe messages are lost or delayed.

---

4. Conclusion

Token-based algorithms like **Suzuki-Kasami's Broadcast Algorithm** and **Raymond's Tree-Based Algorithm** provide structured ways to handle mutual exclusion in distributed systems. However, these approaches can suffer from **deadlocks**, especially due to token loss or cyclic dependencies.

The **Chandy-Misra-Haas Algorithm** provides an effective method to detect and resolve deadlocks through **probe-based cycle detection**. By integrating such deadlock-handling mechanisms, distributed systems can maintain efficiency and avoid indefinite blocking situations.