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

Experiment No. 07

|  |  |
| --- | --- |
| 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 Mutual Exclusion Using Raymond's Tree Algorithm

**1. Introduction**

Mutual exclusion is a fundamental problem in distributed systems, ensuring that only one process accesses a critical section (CS) at a time. Raymond's Tree Algorithm is a **token-based mutual exclusion algorithm** that organizes processes in a logical tree structure, reducing the number of messages required to request and pass the token.

**2. Objective**

The objective of this lab is to implement Raymond's Tree Algorithm in Java and demonstrate how processes request and receive the token in a **distributed system**.

**3. Theory**

Raymond’s Tree Algorithm is an **optimized version** of the token-based mutual exclusion approach. It minimizes message complexity by using a **logical tree structure** where:

* Each process (node) maintains a **request queue**.
* There is only **one token** in the system.
* The process **holding the token** can enter the critical section.
* If a process **does not have the token**, it forwards the request **toward the token holder** via its parent in the tree.

**Key Features:**

1. **Logical Tree Structure**:
   * Each node maintains a **parent pointer**, pointing toward the token holder.
2. **Request Handling**:
   * A process requesting the token forwards its request **upward** until it reaches the **token holder**.
   * The token then moves **downward** through the queue.
3. **Token Passing**:
   * Once a process **finishes using the token**, it **passes it** to the next requestor in its queue.

**Advantages of Raymond’s Algorithm:**

* **Lower message complexity** (**O(log N)** in a balanced tree).
* **Efficient token forwarding**, reducing redundant requests.
* **Prevents starvation** by processing requests in FIFO order.

**4. Working of the Algorithm**

**Step 1: Initial Setup**

* A **logical tree structure** is created, with a single token present at one node.
* Each node maintains:
  + **Parent ID** (the process to forward requests).
  + **Queue** (FIFO order of requests).

**Step 2: Token Request Process**

1. A process that needs access to the **critical section (CS)** checks if it **already has the token**:
   * If **yes**, it enters the CS.
   * If **no**, it adds itself to the **request queue** and forwards the request **to its parent**.
2. The request **propagates up** the tree until it reaches the **token holder**.

**Step 3: Token Forwarding**

1. The **token holder** sends the token **downward** to the first process in its request queue.
2. The token moves **process by process** until it reaches the requester.

**Step 4: Releasing the Token**

1. When a process **finishes using the token**, it:
   * Checks its request queue.
   * If the queue is **not empty**, it **sends the token** to the first requestor.
   * If the queue is **empty**, it keeps the token.

**5. Implementation Details**

* **Data Structures Used**:
  + **Queue**: To manage token requests.
  + **Map (HashMap in Java)**: To store process nodes and their states.
* **Functions Implemented**:
  + requestToken(): A process requests the token.
  + receiveRequest(): Handles incoming token requests.
  + sendToken(): Passes the token to the next process.
  + receiveToken(): Processes token reception and forwards it if needed.

**Conclusion**

Raymond's Tree Algorithm efficiently solves mutual exclusion in distributed systems using a logical tree structure. It significantly reduces message overhead compared to traditional token-based algorithms by limiting requests to logically related nodes. The experiment successfully demonstrated how a process requests, receives, and passes the token, ensuring mutual exclusion in a distributed environment.

**Code:  
  
import** **java.util.\***;

**class** Node {

**int** id;

**boolean** hasToken;

**int** parentId;

**Queue**<**Integer**> requestQueue;

**public** Node(**int** id, **boolean** hasToken, **int** parentId) {

**this**.id **=** id;

**this**.hasToken **=** hasToken;

**this**.parentId **=** parentId;

**this**.requestQueue **=** **new** **LinkedList**<>();

    }

**public** **void** requestToken(**DistributedSystem** system) {

**if** (**!**hasToken) {

**if** (requestQueue.isEmpty()) {

                requestQueue.add(id);

                system.sendRequest(parentId, id);

            }

        }

    }

**public** **void** receiveRequest(**int** requesterId, **DistributedSystem** system) {

        requestQueue.add(requesterId);

**if** (hasToken) {

            sendToken(system);  // ✅ Fixed: Pass system

        } **else** **if** (requestQueue.size() **==** 1) {

            system.sendRequest(parentId, id);

        }

    }

**public** **void** sendToken(**DistributedSystem** system) {

**if** (**!**requestQueue.isEmpty()) {

**int** nextProcess **=** requestQueue.poll();

            System.out.println("Process " **+** id **+** " sends token to Process " **+** nextProcess);

            system.sendToken(nextProcess);

            hasToken **=** **false**;

        }

    }

**public** **void** receiveToken(**DistributedSystem** system) {  // ✅ Fixed: Pass system

        hasToken **=** **true**;

        System.out.println("Process " **+** id **+** " received the token.");

        sendToken(system);  // ✅ Pass system reference

    }

}

**class** DistributedSystem {

**Map**<**Integer**, **Node**> nodes;

**public** DistributedSystem() {

        nodes **=** **new** **HashMap**<>();

    }

**public** **void** addNode(**int** id, **boolean** hasToken, **int** parentId) {

        nodes.put(id, **new** Node(id, hasToken, parentId));

    }

**public** **void** sendRequest(**int** to, **int** from) {

        System.out.println("Process " **+** from **+** " requests token from " **+** to);

        nodes.get(to).receiveRequest(from, **this**);

    }

**public** **void** sendToken(**int** to) {

        nodes.get(to).receiveToken(**this**);  // ✅ Pass 'this' to keep system reference

    }

**public** **void** startRequest(**int** id) {

        nodes.get(id).requestToken(**this**);

    }

}

**public** **class** RaymondTree {

**public** **static** **void** main(**String**[] args) {

**DistributedSystem** system **=** **new** DistributedSystem();

        system.addNode(0, **true**, **-**1);

        system.addNode(1, **false**, 0);

        system.addNode(2, **false**, 1);

        system.addNode(3, **false**, 1);

        system.addNode(4, **false**, 2);

        System.out.println("\nProcess 4 initiates token request...\n");

        system.startRequest(4);

    }

}

**Output:  
  
A black screen with green and blue text

AI-generated content may be incorrect.**