|  |  |
| --- | --- |
| **Name**: Shivam Tiwari | **Roll No**: 5117060 |

**Aim:** Implementation of mutual exclusion algorithm.

**Theory:**

**Mutual exclusion** is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process cannot enter its critical section while another concurrent process is currently present or executing in its critical section i.e. only one process is allowed to execute the critical section at any given instance of time.

**Mutual exclusion in single computer system vs. distributed system:**   
In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved.

In Distributed systems, we neither have shared memory nor a common physical clock and there for we cannot solve mutual exclusion problem using shared variables. To eliminate the mutual exclusion problem in distributed system approach based on message passing is used.

A site in distributed system does not have complete information of state of the system due to lack of shared memory and a common physical clock.

**Requirements of Mutual exclusion Algorithm:**

* **No Deadlock :**   
  Two or more site should not endlessly wait for any message that will never arrive.
* **No Starvation :**  
  Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
* **Fairness:**  
  Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.
* **Fault Tolerance:**  
  In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

**Solution to distributed mutual exclusion:**  
As we know shared variables or a local kernel can not be used to implement mutual exclusion in distributed systems. Message passing is a way to implement mutual exclusion. Below are the three approaches based on message passing to implement mutual exclusion in distributed systems:

1. **Token Based Algorithm:**
   * A unique **token** is shared among all the sites.
   * If a site possesses the unique token, it is allowed to enter its critical section
   * This approach uses sequence number to order requests for the critical section.
   * Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.
   * This approach insures Mutual exclusion as the token is unique
   * **Example:**
   * Suzuki-Kasami’s Broadcast Algorithm
2. **Non-token based approach:**
   * A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successive round of messages among sites.
   * This approach use timestamps instead of sequence number to order requests for the critical section.
   * Whenever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.
   * All algorithm which follows non-token based approach maintains a logical clock. Logical clocks get updated according to Lamport’s scheme
   * **Example:**
   * Lamport's algorithm, Ricart–Agrawala algorithm
3. **Quorum based approach:**
   * Instead of requesting permission to execute the critical section from all other sites, Each site requests only a subset of sites which is called a **quorum**.
   * Any two subsets of sites or Quorum contains a common site.
   * This common site is responsible to ensure mutual exclusion
   * **Example:**
   * Maekawa’s Algorithm

**Program:**

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.OutputStreamWriter;

import java.io.PrintWriter;

import java.net.InetAddress;

import java.net.MulticastSocket;

import java.net.ServerSocket;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

public class suzuki\_kasami {

public static void exitCS(site localsite, int thissiteNumber, int number\_of\_nodes, String[] ip\_addr, int[] port,

int no\_of\_sites) {

localsite.LN[thissiteNumber - 1] = localsite.RN[thissiteNumber - 1];

// Send updated LN value to all sites

String message = "ln," + thissiteNumber + "," + localsite.LN[thissiteNumber - 1];

for (int i = 0; i < no\_of\_sites; i++) {

if(i==thissiteNumber-1) {

continue;

}

try {

Socket skt = new Socket(ip\_addr[i], port[i]);

// System.out.println(skt.getPort());

OutputStream os = skt.getOutputStream();

OutputStreamWriter osw = new OutputStreamWriter(os);

BufferedWriter bw = new BufferedWriter(osw);

bw.write(message);

bw.flush();

skt.close();

} catch (UnknownHostException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

for (int i = 0; i < number\_of\_nodes; i++) {

if (localsite.RN[i] == localsite.LN[i] + 1) {

if (!localsite.token\_queue.contains(i + 1)) {

localsite.token\_queue.add(i + 1);

}

}

}

if (localsite.token\_queue.size() > 0) {

localsite.sendToken(localsite.token\_queue.poll());

}

}

public static void main(String[] args) {

// Read config of nodes

BufferedReader br = null;

File fFile = new File("");

String cwd = fFile.getAbsolutePath();

File nodes = new File(cwd + "\\nodes.config");

try {

br = new BufferedReader(new FileReader(nodes));

int number\_of\_nodes = 0;

String node\_addr = br.readLine();

ArrayList<String> node\_table = new ArrayList<String>();

while (node\_addr != null) {

node\_table.add(node\_addr);

number\_of\_nodes++;

node\_addr = br.readLine();

}

//System.out.println(number\_of\_nodes);

int[] siteNumber = new int[number\_of\_nodes];

String[] ip\_addr = new String[number\_of\_nodes];

int[] port = new int[number\_of\_nodes];

String[] tmpAddress = null;

for (int counter = 0; counter < number\_of\_nodes; counter++) {

tmpAddress = node\_table.get(counter).split(" ");

siteNumber[counter] = Integer.parseInt(tmpAddress[0]);

ip\_addr[counter] = tmpAddress[1];

port[counter] = Integer.parseInt(tmpAddress[2]);

}

// preparing this site

Scanner scan = new Scanner(System.in);

int thissiteNumber = 0;

int inFlag = 0;

do {

System.out.print("Enter site number (1-" + number\_of\_nodes + "): ");

thissiteNumber = Integer.parseInt(scan.nextLine());

if (thissiteNumber >= 1 && thissiteNumber <= number\_of\_nodes) {

inFlag = 1;

} else {

System.out.println("Please enter the correct site number i.e. from 1 to " + number\_of\_nodes);

}

} while (inFlag == 0);

int hasToken = 0;

if (thissiteNumber == 1) {

hasToken = 1;

}

site localsite = new site(number\_of\_nodes, thissiteNumber, hasToken, ip\_addr, port);

// Open a socket

listenToBroadcast listenBcst = new listenToBroadcast(localsite, port[thissiteNumber - 1]);

listenBcst.start();

String input\_query = "";

while (!input\_query.equalsIgnoreCase("quit")) {

System.out.println("Press ENTER to enter CS: ");

Scanner scan\_query = new Scanner(System.in);

input\_query = scan\_query.nextLine();

// System.out.println("Query from keyboard:" + input\_query);

if (localsite.token == 1) {

localsite.processingCS = 1;

System.out.println("Site has token. Executing CS.");

Thread.sleep(15000);

localsite.processingCS = 0;

System.out.println("Exiting CS.");

exitCS(localsite, thissiteNumber, number\_of\_nodes, ip\_addr, port, number\_of\_nodes);

} else {

System.out.println("Requesting token");

localsite.reqCS();

System.out.println("Waiting for token..");

localsite.processingCS = 1;

while (localsite.token == 0) {

Thread.sleep(100);

}

System.out.println("Site has received token. Executing CS.");

Thread.sleep(15000);

localsite.processingCS = 0;

System.out.println("Exiting CS.");

exitCS(localsite, thissiteNumber, number\_of\_nodes, ip\_addr, port, number\_of\_nodes);

}

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

class site {

String[] ip\_addr = null;

int[] port = null;

int number\_of\_sites = 0;

int site\_number = 0;

int token = 0;

int seq\_number = 0;

int processingCS = 0;

Queue<Integer> token\_queue = new LinkedList<>();

int RN[];

int LN[];

site(int numberofsites, int siteNumber, int hasToken, String[] ipAddr, int[] portno) {

this.number\_of\_sites = numberofsites;

this.site\_number = siteNumber;

this.token = hasToken;

this.ip\_addr = ipAddr;

this.port = portno;

RN = new int[number\_of\_sites];

LN = new int[number\_of\_sites];

for (int i = 0; i < numberofsites; i++) {

RN[i] = 0;

LN[i] = 0;

}

}

void print() {

System.out.println(number\_of\_sites + " " + site\_number + " " + token);

}

void updateLN(int thissiteNumber, int value) {

LN[thissiteNumber-1]=value;

}

void reqCS() {

RN[site\_number - 1]++;

String message = "request," + site\_number + "," + RN[site\_number - 1];

System.out.println("Broadcasting request to " + (number\_of\_sites - 1) + " sites.");

for (int i = 0; i < number\_of\_sites; i++) {

if (i != site\_number - 1) {

Socket skt = null;

try {

// InetAddress address= InetAddress.getByName(ip\_addr[i]);

skt = new Socket(ip\_addr[i], port[i]);

System.out.println(skt.getPort());

OutputStream os = skt.getOutputStream();

OutputStreamWriter osw = new OutputStreamWriter(os);

BufferedWriter bw = new BufferedWriter(osw);

bw.write(message);

bw.flush();

// InputStream is = skt.getInputStream();

// InputStreamReader isr = new InputStreamReader(is);

// BufferedReader br = new BufferedReader(isr);

// String message1 = br.readLine();

// System.out.println(message1);

os.close();

osw.close();

bw.close();

} catch (UnknownHostException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} finally {

try {

skt.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

}

}

void processCSReq(int site, int sn) {

if (RN[site - 1] < sn) {

RN[site - 1] = sn;

}

if (processingCS == 0 && token == 1) {

sendToken(site);

} else {

token\_queue.add(site);

}

}

void sendToken(int site) {

if (this.token == 1) {

if (RN[site - 1] == LN[site - 1] + 1) {

System.out.println("Sending token to site " + site);

try {

Socket skt = new Socket(ip\_addr[site - 1], port[site - 1]);

String message = "token";

int tokenQueuelen=token\_queue.size();

for(int i=0;i<tokenQueuelen;i++) {

message+=","+token\_queue.poll();

}

OutputStream os = skt.getOutputStream();

OutputStreamWriter osw = new OutputStreamWriter(os);

BufferedWriter bw = new BufferedWriter(osw);

bw.write(message);

bw.flush();

skt.close();

this.token = 0;

} catch (UnknownHostException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

} // -end sending

}

}

}

class listenToBroadcast extends Thread {

int port = 0;

site localSite = null;

public listenToBroadcast(site thisSite, int port) {

this.port = port;

this.localSite = thisSite;

}

public void run() {

try {

ServerSocket serverSckt = new ServerSocket(port);

while (true) {

Socket skt = serverSckt.accept();

new processRq(skt, localSite).start();

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

class processRq extends Thread {

Socket lSocket = null;

site localSite = null;

public processRq(Socket lSocket, site localSitePass) {

this.lSocket = lSocket;

this.localSite = localSitePass;

}

public void run() {

BufferedReader in = null;

PrintWriter out = null;

// System.out.println(lSocket.getInetAddress().getHostAddress());

try {

in = new BufferedReader(new InputStreamReader(lSocket.getInputStream()));

// out = new PrintWriter(

// new OutputStreamWriter(lSocket.getOutputStream()));

// out.println("Welcome");

// out.flush();

String command = "";

String[] message = null;

command = in.readLine();

System.out.println(command);

if (null != command) {

if (command.charAt(0) == 'r') {

message = command.split(",");

localSite.processCSReq(Integer.parseInt(message[1]), Integer.parseInt(message[2]));

// synchronized (localSite) {

//

// localSite.sendToken(Integer.parseInt(message[1]));

// }

}

if (command.charAt(0) == 't') {

message = command.split(",");

localSite.token\_queue.clear();

int length=message.length;

for(int i=1;i<length;i++) {

localSite.token\_queue.add(Integer.parseInt(message[i]));

}

localSite.token = 1;

}

if (command.charAt(0) == 'l') {

message = command.split(",");

localSite.updateLN(Integer.parseInt(message[1]), Integer.parseInt(message[2]));

}

}

// while (!command.equalsIgnoreCase("quit")) {

// command = in.readLine();

// System.out.println(command);

// if (null != command) {

// if (command.charAt(0) == 'r') {

// message = command.split(",");

// // localSite.processCSReq(Integer.parseInt(message[1]),

// // Integer.parseInt(message[2]));

//

// }

// }else {

// command="";

// }

// }

//

// out.println("Bye");

// out.flush();

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

in.close();

// out.close();

lSocket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

**Output:**

Press ENTER to enter CS:

request,2,1

Requesting token

Broadcasting request to 3 sites.

3450

3451

3453

Waiting for token..

request,4,1

ln,1,0

request,1,1

ln,2,1

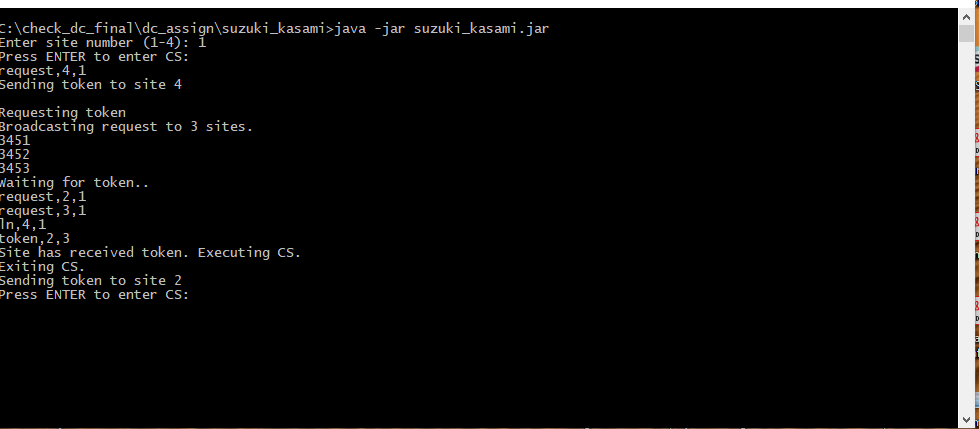
token,4,1

Site has recieved token. Executing CS.

Exiting CS.

Sending token to site 4

Press ENTER to enter CS:



**Conclusion:**

Thus we have successfully studied and implemented mutual exclusion algorithm in a distributed system.