**Department of Computer Engineering**

**Academic Term: Jan-May 2021**

**Class: *B.E. Computer*  Subject Name: *DC***

**Semester: VIII Subject Code: *CSC802***

|  |  |
| --- | --- |
| **Practical No:** | **7** |
| **Title:** | **Mutual Exclusion Algorithm.** |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Roll No:** | **8364** |
| **Name of Student:** | **Vedant Sahai** |

**Evaluation:**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Rubric** | **Grade** |
| **1** | **On time Completion & Submission** |  |
| **2** | **Coding Standards** |  |
| **3** | **Output/Test Cases Used** |  |
| **4** | **Conclusion & Post Lab Assignment** |  |

**Signature of the Teacher:**

**Mutual Exclusion Algorithm.**

1. **Aim:** To implement a Bulletin Board using Vector Timestamps for Event ordering.
2. Code:

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();

}

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();

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 {

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();

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()));

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]));

}

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]));

}

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

in.close();

lSocket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

1. Output:

**Node 1:**

C:\Users\Vedant \OneDrive\Desktop\delete later\temp>java -jar suzuki\_kasami.jar

Enter site number (1-4): 1

Press ENTER to enter CS:

Site has token. Executing CS.

request,4,1

request,2,1

request,3,1

Exiting CS.

Sending token to site 4

Press ENTER to enter CS:

ln,4,1

ln,2,1

ln,3,1

**Node 4:**

C:\Users\Vedant \OneDrive\Desktop\delete later\temp>java -jar suzuki\_kasami.jar

Enter site number (1-4): 4

Press ENTER to enter CS:

Requesting token

Broadcasting request to 3 sites.

3450

3451

3452

Waiting for token..

request,2,1

request,3,1

ln,1,0

token,2,3

Site has received token. Executing CS.

Exiting CS.

Sending token to site 2

Press ENTER to enter CS:

ln,2,1

ln,3,1

**Node 2:**

C:\Users\Vedant \OneDrive\Desktop\delete later\temp>java -jar suzuki\_kasami.jar

Enter site number (1-4): 2

Press ENTER to enter CS:

request,4,1

Requesting token

Broadcasting request to 3 sites.

3450

3452

3453

Waiting for token..

request,3,1

ln,1,0

ln,4,1

token,3

Site has received token. Executing CS.

Exiting CS.

Sending token to site 3

Press ENTER to enter CS:

ln,3,1

**Node 3:**

C:\Users\Vedant \OneDrive\Desktop\delete later\temp>java -jar suzuki\_kasami.jar

Enter site number (1-4): 3

Press ENTER to enter CS:

request,4,1

request,2,1

Requesting token

Broadcasting request to 3 sites.

3450

3451

3453

Waiting for a token.

ln,1,0

ln,4,1

ln,2,1

token

The site has received a token. Executing CS.

Exiting CS.

Press ENTER to enter CS:

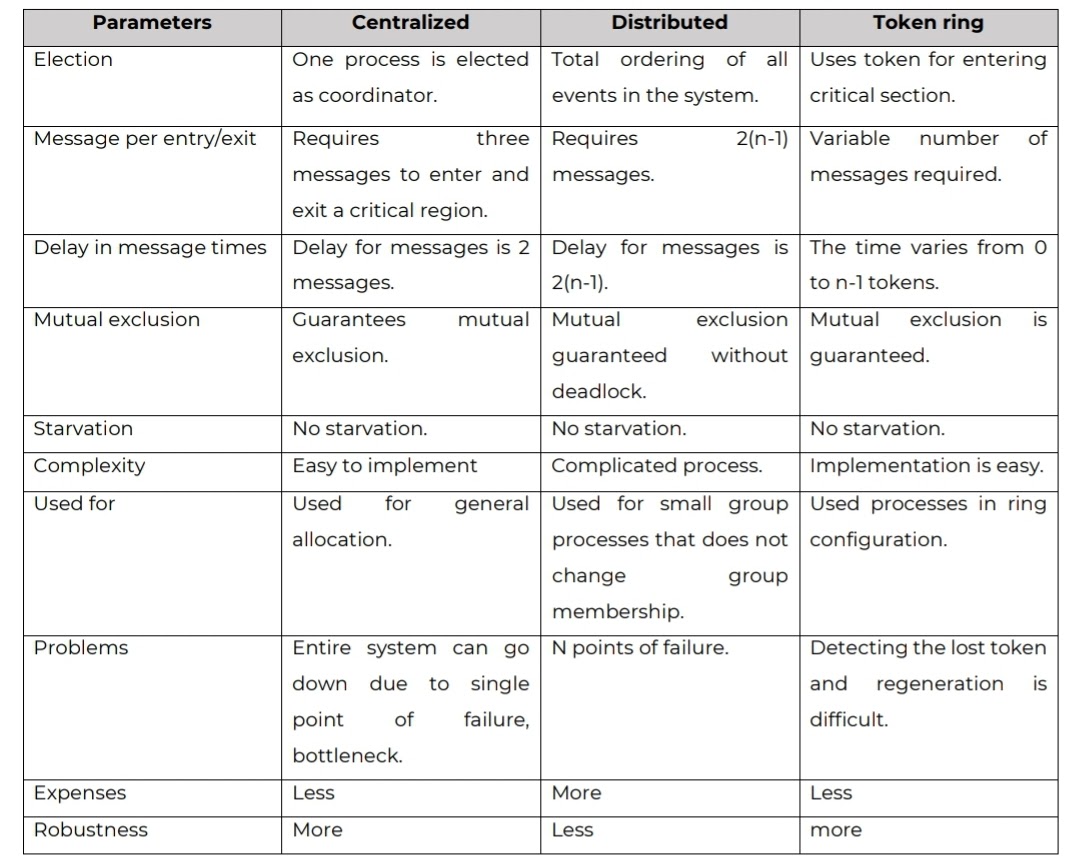
1. Conclusion:

Mutual exclusion is a concurrency control property that 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.

In Distributed Systems, we neither have shared memory nor a common physical clock and therefore we cannot solve the 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 the distributed system does not have complete information of the state of the system due to a lack of shared memory and a common physical clock.

1. Post Lab Assignments:
2. Compare the three (Centralized, distributed and token-based) algorithms.

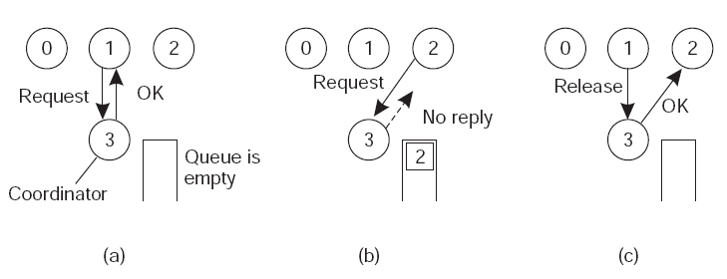
Ans: -



1. What are the disadvantages of centralised algorithms?

Ans: -

In Centralized Algorithm, one process is elected as the coordinator which may be the machine with the highest network address.



Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission. If no other process is currently in that critical region, the coordinator sends back a reply granting permission, as shown in Fig (a). When the reply arrives, the requesting process enters the critical region.

Suppose another process 2 shown in Fig (b), asks for permission to enter the same critical region. Now the coordinator knows that a different process is already in the critical region, so it cannot grant permission. The coordinator just refrains from replying, thus blocking process 2, which is waiting for a reply or it could send a reply saying ‘permission denied.’

When process 1 exits the critical region, it sends a message to the coordinator releasing its exclusive access as shown in Fig (c).

The coordinator takes the first item off the queue of deferred requests and sends that process a grant message. If the process was still blocked it unblocks and enters the critical region.

If an explicit message has already been sent denying permission, the process will have to poll for incoming traffic or block later. When it sees the grant, it can enter the critical region.

**Disadvantages**:

* The coordinator is a single point of failure, the entire system may go down if it crashes
* In a large system, a single coordinator can become a performance bottleneck.
* If processes normally block after making a request, they cannot distinguish a dead coordinator from ‘‘permission denied’’ since no message comes back.

**Department of Computer Engineering**

**Academic Term: Jan-May 2021**

**Class: *B.E. Computer*  Subject Name: *DC***

**Semester: VIII Subject Code: *CSC802***

|  |  |
| --- | --- |
| **Practical No:** | **8** |
| **Title:** | **Clock Synchronization algorithms.** |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Roll No:** | **8364** |
| **Name of Student:** | **Vedant Sahai** |

**Evaluation:**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Rubric** | **Grade** |
| **1** | **On time Completion & Submission** |  |
| **2** | **Coding Standards** |  |
| **3** | **Output/Test Cases Used** |  |
| **4** | **Conclusion & Post Lab Assignment** |  |

**Signature of the Teacher:**

**Clock Synchronization Algorithms.**

1. **Aim:** To implement a clock synchronization algorithm (Berkley algorithm).
2. **Code:**

**SERVER**

from functools import reduce

from dateutil import parser

import threading

import datetime

import socket

import time

# datastructure used to store client address and clock data

client\_data = {}

# nested thread function used to receive clock time from a connected client

def startRecieveingClockTime(connector, address):

while True:

# receive clock time

clock\_time\_string = connector.recv(1024).decode()

clock\_time = parser.parse(clock\_time\_string)

clock\_time\_diff = datetime.datetime.now() - \clock\_time

client\_data[address] = {

"clock\_time" : clock\_time,

"time\_difference" : clock\_time\_diff,

"connector" : connector

}

print("Client Data updated with: "+ str(address), end = "\n\n")

time.sleep(5)

# master thread function used to open portal for accepting clients over given port

def startConnecting(master\_server):

# fetch clock time at slaves / clients

while True:

# accepting a client / slave clock client

master\_slave\_connector, addr = master\_server.accept()

slave\_address = str(addr[0]) + ":" + str(addr[1])

print(slave\_address + " got connected successfully")

current\_thread = threading.Thread(target = startRecieveingClockTime,

args = (master\_slave\_connector, slave\_address))

current\_thread.start()

# subroutine function used to fetch average clock difference

def getAverageClockDiff():

current\_client\_data = client\_data.copy()

time\_difference\_list = list(client['time\_difference'] for client\_addr, client in client\_data.items())

sum\_of\_clock\_difference = sum(time\_difference\_list, \datetime.timedelta(0, 0))

average\_clock\_difference = sum\_of\_clock\_difference \/ len(client\_data)

return average\_clock\_difference

# master sync thread function used to generate cycles of clock synchronization in the network

def synchronizeAllClocks():

while True:

print("New synchroniztion cycle started.")

print("Number of clients to be synchronized: " + \str(len(client\_data)))

if len(client\_data) > 0:

average\_clock\_difference = getAverageClockDiff()

for client\_addr, client in client\_data.items():

try:

synchronized\_time = \datetime.datetime.now() + \average\_clock\_difference

client['connector'].send(str(synchronized\_time).encode())

except Exception as e:

print("Something went wrong while " + \"sending synchronized time " + \"through " + str(client\_addr))

else :

print("No client data." + \" Synchronization not applicable.")

print("\n\n")

time.sleep(5)

# function used to initiate the Clock Server / Master Node

def initiateClockServer(port = 8080):

master\_server = socket.socket()

master\_server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

print("Socket at master node created successfully\n")

master\_server.bind(('', port))

# Start listening to requests

master\_server.listen(10)

print("Clock server started...\n")

# start making connections

print("Starting to make connections...\n")

master\_thread = threading.Thread(target = startConnecting, args = (master\_server, ))

master\_thread.start()

# start synchronization

print("Starting synchronization parallely...\n")

sync\_thread = threading.Thread(target = synchronizeAllClocks, args = ())

sync\_thread.start()

if \_name\_ == '\_main\_':

initiateClockServer(port = 8080)

**CLIENT**

from timeit import default\_timer as timer

from dateutil import parser

import threading

import datetime

import socket

import time

# client thread function used to send time at client side

def startSendingTime(slave\_client):

while True:

# provide server with clock time at the client

slave\_client.send(str(datetime.datetime.now()).encode())

print("Recent time sent successfully", end = "\n\n")

time.sleep(5)

# client thread function used to receive synchronized time

def startReceivingTime(slave\_client):

while True:

# receive data from the server

Synchronized\_time = parser.parse(slave\_client.recv(1024).decode())

print("Synchronized time at the client is: " + \str(Synchronized\_time), end = "\n\n")

# function used to Synchronize client process time

def initiateSlaveClient(port = 8080):

slave\_client = socket.socket()

# connect to the clock server on local computer

slave\_client.connect(('127.0.0.1', port))

# start sending time to server

print("Starting to receive time from server\n")

send\_time\_thread = threading.Thread(target = startSendingTime, args = (slave\_client, ))

send\_time\_thread.start()

# start receiving synchronized from server

print("Starting to receiving " + \"synchronized time from server\n")

receive\_time\_thread = threading.Thread(target = startReceivingTime, args = (slave\_client, ))

receive\_time\_thread.start()

if \_name\_ == '\_main\_':

initiateSlaveClient(port = 8080)

1. **Output:**

**Client 1:**

C:\Users\Vedant \Downloads\clock-sync>python client.py

Starting to receive time from server

Starting to receiving synchronized time from server

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:18.444959

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:23.459324

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:28.488060

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:33.504850

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:38.521801

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:43.548204

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:48.570888

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:53.598538

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:58.626058

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:03.641591

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:08.653717

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:13.669800

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:18.678029

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:23.694137

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:28.714945

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:33.739309

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:38.771646

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:43.796787

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:48.815745

Recent time sent successfully

Recent time sent successfully

**Client 2:**

C:\Users\Vedant \Downloads\clock-sync>python client.py

Starting to receive time from server

Starting to recieving synchronized time from server

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:18.444959

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:23.459324

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:28.488060

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:33.504850

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:38.521801

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:43.548204

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:48.570888

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:53.598538

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:14:58.626058

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:03.641591

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:08.653717

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:13.669800

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:18.678029

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:23.694137

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:28.714945

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:33.739309

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:38.771646

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:43.796787

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:15:48.815745

Recent time sent successfully

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:00.702122

Recent time sent successfully

Recent time sent successfully

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:15.399806

Recent time sent successfully

Recent time sent successfully

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:29.039524

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:34.046638

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:39.057704

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:44.074257

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:49.086424

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:54.105769

Recent time sent successfully

Synchronized time at the client is: 2021-04-19 15:16:59.124179

Recent time sent successfully

**Server:**

C:\Users\Vedant \Downloads\clock-sync>python server.py

Socket at master node created successfully

Clock server started...

Starting to make connections...

Starting synchronization parallely...

New synchroniztion cycle started.

Number of clients to be synchronized: 0

No client data. Synchronization not applicable.

127.0.0.1:60895 got connected successfully

Client Data updated with: 127.0.0.1:60895

New synchroniztion cycle started.

Number of clients to be synchronized: 1

Client Data updated with: 127.0.0.1:60895

New synchroniztion cycle started.

Number of clients to be synchronized: 1

127.0.0.1:60898 got connected successfully

Client Data updated with: 127.0.0.1:60898

Client Data updated with: 127.0.0.1:60895

New synchroniztion cycle started.

Number of clients to be synchronized: 2

Client Data updated with: 127.0.0.1:60898

Client Data updated with: 127.0.0.1:60895

New synchroniztion cycle started.

Number of clients to be synchronized: 2

Client Data updated with: 127.0.0.1:60898

Client Data updated with: 127.0.0.1:60895

New synchronization cycle started.

Number of clients to be synchronized: 2

1. **Conclusions:**

In the Distributed Systems (DS) the nodes are communicating with each other using message passing. In many real-time applications such as banking systems, reservation systems that are implemented on distributed systems, it is important to execute each transaction/event in an ordered manner. Ordering of events is essential for proper allocation of available resources and mutual allocation. This can be implemented using clock synchronization algorithms.

1. **Post labs:**
2. Give the example synchronization algorithm with a) an active server, b) a passive server.

Ans: -

a.] Active Time Server Centralized Algorithm: In this approach, the time server periodically broadcasts its clock time (“time = T”). The other nodes receive the broadcast message and use the clock time in the message for correcting their clocks. Each node has a priori knowledge of the approximate time (Ta) required for the propagation of the message “time = T” from the time server node to its node, Therefore, when a broadcast message is received at a node, the node’s clock is readjusted to the time T+Ta. A major drawback of this method is that it does not fault-tolerant. If the broadcast message reaches too late at a node due to some communication fault, the clock of that node will be readjusted to an incorrect value. Another disadvantage of this approach is that it requires the broadcast facility to be supported by the network.

b.] Passive Time Server Centralized Algorithm: In this method, each node periodically sends a message to the time server. When the time server receives the message, it quickly responds with a message (“time = T”), where T is the current time in the clock of the time server node. Assume that when the client node sends the “time =?” message, its clock time is T0, and when it receives the “time = T” message, its clock time is T1. Since T0 and T1 are measured using the same clock, in the absence of any other information, the best estimate of the time required for the propagation of the message “time = T” from the time server node to the client’s node is (T1-T0)/2. Therefore, when the reply is received at the client’s node, its clock is readjusted to T + (T1-T0)/2.

1. What are the disadvantages of Cristian’s algorithm (Mention 3)

Ans: -

Disadvantages of Cristian’s algorithm

* Major problem: time must never run backwards

ν If the sender’s clock is faster

ο Sol. Add 9 msec rather than 10 msec at each interrupt

* Minor problem: it takes a nonzero amount of time for a reply

ο Sol.

ν Add (T1 -T0 )/2

ν Add (T1 -T0 -I)/2 (I: Interrupt handling time)

ν Measure more than one time

* Prone to failure of the central server

1. Difference between physical clock and logical clock synchronization

Ans: -

Physical clock

* It is a physical process and also a method of measuring that process to record the passage of time. For example, the rotation of the Earth measured in solar days. Most of the physical clocks are based on cyclic processes such as celestial rotation.
* Tied to the notion of real-time
* Can be used to order events, find the time difference between two events.

Logical clock

* It is a mechanism for capturing causal and chronological relationships in a distributed system. A physically synchronous global clock may not be present in a distributed system. In such systems, a logical clock allows global ordering of events from different processes.
* Derived from the notion of potential cause-effect between events
* Not tied to the notion of real-time
* Can be used to order events

**Department of Computer Engineering**

**Academic Term: Jan-May 2021**

**Class: *B.E. Computer*  Subject Name: *DC***

**Semester: VIII Subject Code: *CSC802***

|  |  |
| --- | --- |
| **Practical No:** | **9** |
| **Title:** | **Group Communication** |
| **Date of Performance:** |  |
| **Date of Submission:** |  |
| **Roll No:** | **8364** |
| **Name of Student:** | **Vedant Sahai** |

**Evaluation:**

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Rubric** | **Grade** |
| **1** | **On time Completion & Submission** |  |
| **2** | **Coding Standards** |  |
| **3** | **Output/Test Cases Used** |  |
| **4** | **Conclusion & Post Lab Assignment** |  |

**Signature of the Teacher:**

**Group Communication**

1. **Aim:** To implement Lamport algorithm for logical clock synchronization
2. **Code:**

**Master.java** import java.io.\*; import java.util.\*; import java.net.\*; public class Master

{

public static void main(String args[])throws Exception

{

String send="", r="";

Socket MyClient = new Socket("192.168.0.106",25); System.out.println("Connected as Master");

DataInputStream din=new DataInputStream(MyClient.getInputStream()); DataOutputStream dout = new DataOutputStream(MyClient.getOutputStream()); Scanner sc = new Scanner(System.in);

do

{

System.out.print("Message('close' to stop): "); send = sc.nextLine();

dout.writeUTF(send); dout.flush();

}while(!send.equals("stop")); dout.close();

din.close(); MyClient.close();

}

}

**Slave.java** import java.io.\*; import java.util.\*; import java.net.\*; public class Slave

{

public static void main(String args[])throws Exception

{

String r="";

Socket MyClient = new Socket("192.168.0.106",25); System.out.println("Connected as Slave");

DataInputStream din=new DataInputStream(MyClient.getInputStream());

do

{

r=din.readUTF(); System.out.println("Master says: " + r);

}while(!r.equals("stop"));

din.close(); MyClient.close();

}

}

**Server.java** import java.util.\*; import java.io.\*; import java.net.\*;

public class Server {

static ArrayList<ClientHandler> clients;

public static void main(String args[]) throws Exception{

//Server server = new Server();

ServerSocket MyServer = new ServerSocket(25); clients = new ArrayList<ClientHandler>(); Socket ss = null;

Message msg = new Message(); int count = 0;

while(true) {

ss = null; try {

ss = MyServer.accept();

DataInputStream din =new DataInputStream(ss.getInputStream()); DataOutputStream dout=new DataOutputStream(ss.getOutputStream()); ClientHandler chlr = new ClientHandler(ss, din, dout, msg);

Thread t = chlr; if (count > 0)

clients.add(chlr); count++;

//System.out.println(threads); t.start();

}

catch(Exception E){

continue;

}

}

}

}

class Message{

String msg;

public void set\_msg(String msg){ this.msg = msg;

}

public void get\_msg(){

System.out.println("\nNEW GROUP MESSAGE: " + this.msg); for(int i = 0; i < Server.clients.size(); i++){

try{

System.out.print("Client: " + Server.clients.get(i).ip + "; "); Server.clients.get(i).out.writeUTF(this.msg); Server.clients.get(i).out.flush();

}

catch(Exception e){

System.out.print(e);

}

}

}

}

class ClientHandler extends Thread{ DataInputStream in; DataOutputStream out; Socket socket;

int sum; float res;

boolean conn; Message msg; String ip;

public ClientHandler(Socket s, DataInputStream din, DataOutputStream dout, Message msg) { this.socket = s;

this.in = din; this.out = dout; this.conn = true; this.msg = msg;

this.ip = (((InetSocketAddress) this.socket.getRemoteSocketAddress()).getAddress()).toString().replace("/","");

}

public void run(){

while(conn == true){ try{

String input = this.in.readUTF();

// System.out.println("From host " + this.ip + ':' + input);

// String msg = "From host " + this.ip + ':' + input; this.msg.set\_msg(input);

this.msg.get\_msg();

}

catch(Exception E){

conn = false;

System.out.println(E);

}

}

closeConn();

}

public void closeConn(){

try{ this.out.close(); this.in.close(); this.socket.close();

}

catch(Exception E){

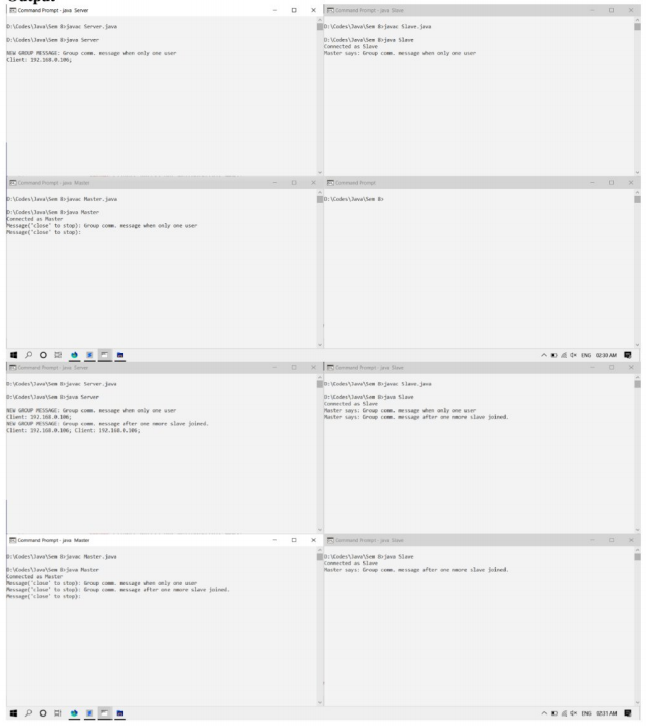
System.out.println(E);

}

}

}

1. **Output:**



1. **Conclusion:**

We have identified 6 criteria that are important design issues for group communication: addressing, reliability, ordering, delivery semantics, response semantics, and group structure. We have discussed each of these criteria and the choices that have been made for the Amoeba distributed system. The Amoeba interface for group communication is simple, powerful, and easy to understand. Its main properties are:

* Messages are totally-ordered per group.
* Programmers can trade performance against fault tolerance.

Based on our experience with distributed programming we believe that these properties are essential in building efficient distributed applications. We have described in detail the group communication interface and its implementation. In addition, we have provided extensive performance measurements on 30 processors. The delay for a null broadcast to a group of 30 processes running on 20MHz MC68030s connected by 10 Mbit/s Ethernet is 2.8 msec. The maximum throughput per group is 815 broadcasts per group. With multiple groups, the maximum number of broadcasts per second has been measured at 3175.

1. **Post lab:**
2. What are the three types of Group communication?

Ans: -

Three types of group communication:

* One to many (single sender and multiple receivers)

 In this scheme, there are multiple receivers for a message sent by a single sender. The one-to-many scheme is also known as multicast communication. A special case of multicast communication is broadcast communication, in which the message is sent to all processors connected to a network.

* Many to one (multiple senders and single receiver)

- multiple senders send messages to a single receiver.

- The single receiver may be selective or nonselective.

- A *selective receiver* specifies a unique sender; a message exchange takes place only if that sender sends a message.

- A nonselective receiver specifies a set of senders, and if anyone sender in the set sends a message to this receiver, a message exchange takes place - an important issue related to the many-to-one communication scheme is nondeterminism

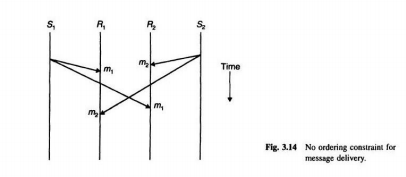
* Many too many (multiple senders and multiple receivers)

- multiple senders send messages to multiple receivers.

- an important issue related to many-to-many communication scheme is that of ordered message delivery

- Ordered message delivery ensures that all messages are delivered to all receivers in an order acceptable to the application. This property is needed by many applications for its correct functioning.

- Ordered message delivery requires message sequencing.



- The commonly used semantics for ordered delivery of multicast messages are absolute ordering, consistent ordering, and causal ordering.

1. What is the primary issue with many-to-one communications?

Ans: -

Many to one is the reverse of one to one many. In a many to one model several different points of information are received by a single reception.

An example of this could be a company sending out surveys to its customers or else reading feedback on their social media account

1. What is the primary issue with many-to-many communications?

Ans: -

In this model where information is generated from multiple sources and is received by multiple sources. The many to many information sharing are often found on modern networked platforms such as social media and other internet-based forms of communication. An e.g., of this would be several different people speaking about a current event of cultural importance in social media and these statements being shared and responded by another different group of readers.

Note that there is no core singular source when this information sharing begins as several people start sharing info wholly unaware and independent of others statement.