**Experiment-1: Implementation FTP Client:**

**FTP Client:**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.net.\*;

import java.io.\*;

class One extends JFrame implements ActionListener

{/\* ctrl space \*/

public JButton b,b1;

public JLabel l;

public JLabel l1,lmsg1,lmsg2;

One(){

b=new JButton("Upload");

l=new JLabel("Uplaod a file : ");

lmsg1=new JLabel("");

b1=new JButton("Download");

l1=new JLabel("Downlaod a file");

lmsg2=new JLabel("");

setLayout(new GridLayout(2,3,10,10));

add(l);add(b);add(lmsg1);add(l1);add(b1);add(lmsg2);

b.addActionListener(this);

b1.addActionListener(this);

setVisible(true);

setSize(600,500);

}

public void actionPerformed(ActionEvent e)

{

// TODO Auto-generated method stub

try {/\* String s=e.getActionCommand();

if(s.equals("Upload"))\*/

if (b.getModel().isArmed())

{Socket s=new Socket("localhost",1010);

System.out.println("Client connected to server");

JFileChooser j=new JFileChooser();

int val;

val=j.showOpenDialog(One.this);

String filename=j.getSelectedFile().getName();

String path=j.getSelectedFile().getPath();

PrintStream out=new PrintStream(s.getOutputStream());

out.println("Upload");

out.println(filename);

FileInputStream fis=new FileInputStream(path);

int n=fis.read();

while (n!=-1)

{out.print((char)n);n=fis.read();

} fis.close(); out.close();lmsg1.setText(filename+"is uploaded");

//s.close();

repaint();

} if (b1.getModel().isArmed())

{

Socket s=new Socket("localhost",1010);

System.out.println("Client connected to server");

String remoteadd=s.getRemoteSocketAddress().toString();

System.out.println(remoteadd);

JFileChooser j1=new JFileChooser(remoteadd);

int val;

val=j1.showOpenDialog(One.this);

String filename=j1.getSelectedFile().getName();

String filepath=j1.getSelectedFile().getPath();

System.out.println("File name:"+filename);

PrintStream out=new PrintStream(s.getOutputStream());

out.println("Download");

out.println(filepath);

FileOutputStream fout=new FileOutputStream(filename);

DataInputStream fromserver=new DataInputStream(s.getInputStream());

int ch;

while ((ch=fromserver.read())!=-1)

{

fout.write((char) ch);

}

fout.close();//s.close();

lmsg2.setText(filename+"is downlaoded");

repaint();

}}

catch (Exception ee)

{// TODO: handle exception

System.out.println(ee);

}}}

public class FTPClient

{ public static void main(String[] args)

{new One();

}}

**FTP Server:**

import java.io.DataInputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.PrintStream;

import java.net.ServerSocket;

import java.net.Socket;

public class FTPServer {

public static void main(String[] args)

{

try {

while (true)

{

ServerSocket ss=new ServerSocket(1010);

Socket sl=ss.accept();

System.out.println("Server scoket is created....");

System.out.println(" test1");

DataInputStream fromserver=new DataInputStream(sl.getInputStream());

System.out.println(" test2");

String option=fromserver.readLine();

if (option.equalsIgnoreCase("upload"))

{

System.out.println("upload test");

String filefromclient=fromserver.readLine();

File clientfile=new File(filefromclient);

FileOutputStream fout=new FileOutputStream(clientfile);

int ch;

while ((ch=fromserver.read())!=-1)

{

fout.write((char)ch);

}

fout.close();

}

if (option.equalsIgnoreCase("download"))

{

System.out.println("download test");

String filefromclient=fromserver.readLine();

File clientfile=new File(filefromclient);

FileInputStream fis=new FileInputStream(clientfile);

PrintStream out=new PrintStream(sl.getOutputStream());

int n=fis.read();

while (n!=-1)

{

out.print((char)n);

n=fis.read();

}fis.close();

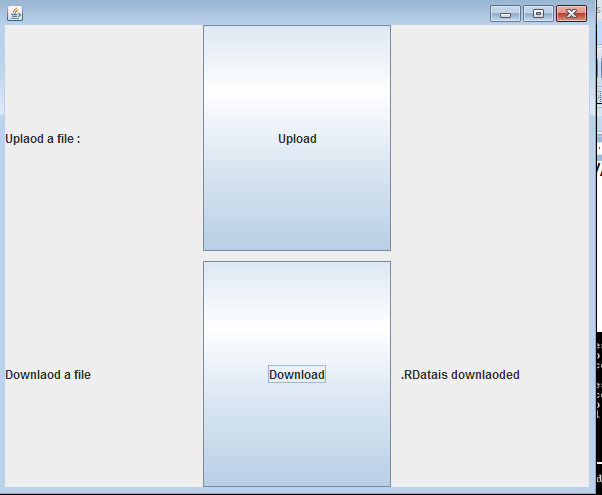
out.close();

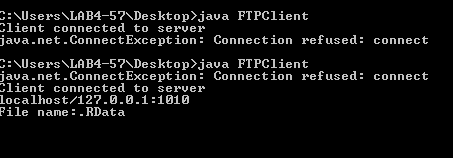
} //while} }

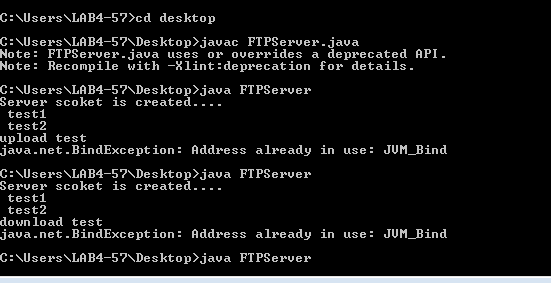
catch (Exception e)

{System.out.println(e);

}}}







**Experiment-2: Implementation of Name Server**

**Program:**

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class DNS

{public static void main(String[] args)

{

int n;

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

do{

System.out.println("\n Menu: \n 1. DNS 2. Reverse DNS 3. Exit \n");

System.out.println("\n Enter your choice");

n = Integer.parseInt(System.console().readLine());

if(n==1){

try {

System.out.println("\n Enter Host Name ");

String hname=in.readLine();

InetAddress address;

address = InetAddress.getByName(hname);

System.out.println("Host Name: " + address.getHostName());

System.out.println("IP: " + address.getHostAddress());

} catch(IOException ioe)

{ioe.printStackTrace();

}}

if(n==2){

try {

System.out.println("\n Enter IP address");

String ipstr = in.readLine();

InetAddress ia = InetAddress.getByName(ipstr);

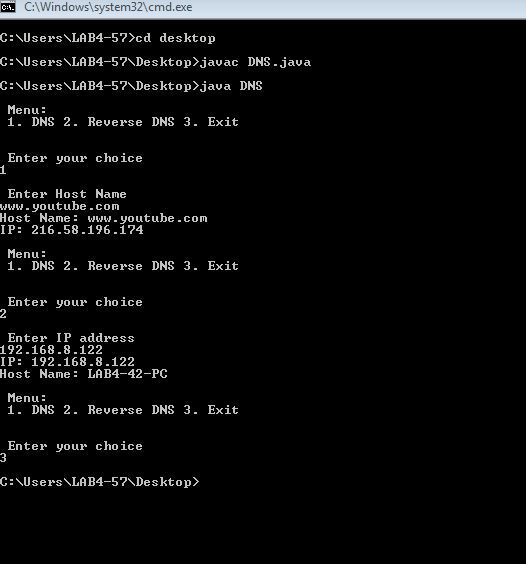
System.out.println("IP: "+ipstr);

System.out.println("Host Name: " +ia.getHostName());

} catch(IOException ioe)

{ioe.printStackTrace();

}}}while(!(n==3));}}



**Experiment-3: Implementation of Chat Server**

**CCLogin.java**

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

import java.awt.GridLayout;

public class CCLogin implements ActionListener

{

JFrame frame1; JTextField tf,tf1; JButton button;

JLabel heading; JLabel label,label1;

public static void main(String[] paramArrayOfString)

{

new CCLogin();

}

public CCLogin()

{

this.frame1 = new JFrame("Login Page");

this.tf = new JTextField(10);

this.button = new JButton("Login");

this.heading = new JLabel("Chat Server");

this.heading.setFont(new Font("Impact", 1, 40));

this.label = new JLabel("Enter you Login Name");

this.label.setFont(new Font("Serif", 0, 24));

JPanel localJPanel = new JPanel();

this.button.addActionListener(this);

localJPanel.add(this.heading); localJPanel.add(this.label);

localJPanel.add(this.tf);

localJPanel.add(this.button);

this.heading.setBounds(30, 20, 280, 50);

this.label.setBounds(20, 100, 250, 60);

this.tf.setBounds(50, 150, 150, 30);

this.button.setBounds(70, 190, 90, 30);

this.frame1.add(localJPanel);

localJPanel.setLayout(null);

this.frame1.setSize(300,300);

this.frame1.setVisible(true);

this.frame1.setDefaultCloseOperation(3);

} public void actionPerformed(ActionEvent paramActionEvent)

{

String str = "";

try

{ str = this.tf.getText();

this.frame1.dispose();

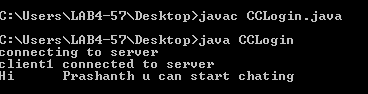
Client1 c1= new Client1(str);

c1.main(null);}

catch(Exception localIOException)

{

}}}



**ChatMultiServer:**

import java.net.\*;

import java.io.\*;

class A implements Runnable

{

Thread t;

Socket s;

A(Socket x)

{

s=x;

t=new Thread(this);

t.start();

}

public void run()

{

try

{ /\* Reading data from client \*/

InputStream is=s.getInputStream();

byte data[]=new byte[50];

is.read(data);

String mfc=new String(data);

mfc=mfc.trim();

System.out.println(mfc);

/\* Sending message to the server \*/

//System.out.println("Hi"+name+"u can start chating");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

String n=br.readLine();

OutputStream os=s.getOutputStream();

os.write(n.getBytes());

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

class ChatMultiServer

{

static int c=0;

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

{

System.out.println("ServerSocket is creating");

ServerSocket ss=new ServerSocket(1010);

System.out.println("ServerSocket is created");

System.out.println("waiting for the client from the client");

while(true)

{

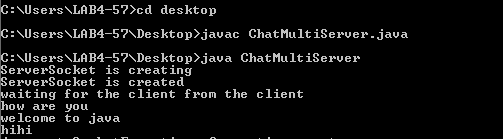
Socket s=ss.accept();

new A(s);

}

}

}



**Client1.java**

import java.net.\*;

import java.io.\*;

class Client1

{

static String name="";

public Client1(String n)

{

name=n;

}

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

{

System.out.println("connecting to server");

System.out.println("client1 connected to server");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

/\* Sending message to the server \*/

System.out.println("Hi\t"+name+" u can start chating");

while(true)

{

Socket s=new Socket("localhost",1010);

String n=br.readLine();

OutputStream os=s.getOutputStream();

os.write(n.getBytes());

/\* Reading data from client \*/

InputStream is=s.getInputStream();

byte data[]=new byte[50];

is.read(data);

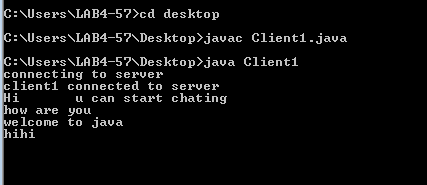
String mfc=new String(data);

mfc=mfc.trim();

System.out.println(mfc);

}

} }



**Experiment-6: Implement a word count application which counts the number of occurrences of each words a large collection of documents Using Map Reduce model**

package PackageDemo;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

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

{

Configuration c=new Configuration();

String[] files=new GenericOptionsParser(c,args).getRemainingArgs();

Path input=new Path(files[0]);

Path output=new Path(files[1]);

Job j=new Job(c,"wordcount");

j.setJarByClass(WordCount.class);

j.setMapperClass(MapForWordCount.class);

j.setReducerClass(ReduceForWordCount.class);

j.setOutputKeyClass(Text.class);

j.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(j, input);

FileOutputFormat.setOutputPath(j, output);

System.exit(j.waitForCompletion(true)?0:1);

}

public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{

public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException

{

String line = value.toString();

String[] words=line.split(",");

for(String word: words )

{

Text outputKey = new Text(word.toUpperCase().trim());

IntWritable outputValue = new IntWritable(1);

con.write(outputKey, outputValue);

}

}}

public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>

{

public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException

{

int sum = 0;

for(IntWritable value : values)

{

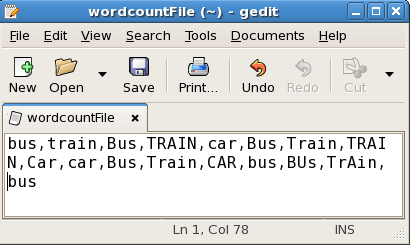
sum += value.get();

}con.write(word, new IntWritable(sum));

}}}

**Output:**

**1.** **Take a text file and move it into HDFS format:**



To move this into Hadoop directly, open the terminal and enter the following commands:

[training@localhost ~]$ hadoop fs -put wordcountFile wordCountFile

**2. Run the jar file: (***Hadoop jar jarfilename.jar packageName.ClassName  PathToInputTextFile PathToOutputDirectry)*

[training@localhost ~]$ hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile MRDir1

**3. Open the result:**[training@localhost ~]$ hadoop fs -ls MRDir1

Found 3 items

-rw-r--r-- 1 training supergroup 0 2016-02-23 03:36 /user/training/MRDir1/\_SUCCESS

drwxr-xr-x - training supergroup 0 2016-02-23 03:36 /user/training/MRDir1/\_logs

-rw-r--r-- 1 training supergroup 20 2016-02-23 03:36 /user/training/MRDir1/part-r-00000

[training@localhost ~]$ hadoop fs -cat MRDir1/part-r-00000

BUS 7

CAR 4

TRAIN 6

**5. RPC HELLOWORLD**

**Publisher.java**

package rpc\_helloworld;

import javax.xml.ws.Endpoint;

public class Publisher {

public static void main(String[] args) {

Endpoint.publish("http://localhost:7779/ws/hello", new HelloWorldImpl());

}

}

**HelloWorld.java**

package rpc\_helloworld;

import javax.jws.WebMethod;

import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import javax.jws.soap.SOAPBinding.Style;

@WebService

@SOAPBinding(style = Style.RPC)

public interface HelloWorld {

@WebMethod String getHelloWorld(String name);

}

**Experient 8:Implementation of Berkeley Algorithm**

**Client.cpp**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <iostream>

#include <stdlib.h> /\* srand, rand \*/

#include <cstdlib>

#include <ctime>

#include <vector>

#define PORT 8080

using namespace std;

// function for string delimiter

vector<string> split(string s, string delimiter) {

size\_t pos\_start = 0, pos\_end, delim\_len = delimiter.length();

string token;

vector<string> res;

while ((pos\_end = s.find (delimiter, pos\_start)) != string::npos) {

token = s.substr (pos\_start, pos\_end - pos\_start);

pos\_start = pos\_end + delim\_len;

res.push\_back (token);

}

res.push\_back (s.substr (pos\_start));

return res;

}

int main(int argc, char const \*argv[])

{

srand((unsigned int)time(NULL)); // avoid always same output of rand()

float client\_local\_clock = rand() % 10; // range from 0 to 9

printf("Client starts. Client pid is %d \n", getpid());

printf("Client local clock is %f \n\n", client\_local\_clock);

int client\_socket\_fd, valread;

char client\_read\_buffer[1024] = {0};

struct sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

// server\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]); // hardcode to 127.0.0.1

server\_addr.sin\_port = htons(PORT);

// Creating socket file descriptor (IPv4, TCP, IP)

if ((client\_socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0)

{

printf("\n Client: Socket creation error \n");

return -1;

}

// Converting IPv4 and IPv6 addresses from text to binary form,

// from character string src into a network

// address structure in the af address family, then copies the

// network address structure to dst.

if(inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr)<=0)

{printf("\nClient: Invalid address/ Address not supported \n");

return -1;

}// Connecting server, return 0 with success, return -1 with error

if (connect(client\_socket\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

{printf("\nClient: Connection Failed \n");

return -1;

}char server\_ip[INET\_ADDRSTRLEN]="";

inet\_ntop(AF\_INET, &server\_addr.sin\_addr, server\_ip, INET\_ADDRSTRLEN);

printf("Client: connected server(%s:%d). \n", server\_ip, ntohs(server\_addr.sin\_port));

printf("\n\n");

//

// first round communication

// receiving form server

valread = read( client\_socket\_fd , client\_read\_buffer, 1024);

printf("Client: read: '%s'\n",client\_read\_buffer );

// convert char array to string

string recv\_msg = string(client\_read\_buffer);

// reply according to what client receive

if (strcmp(client\_read\_buffer, "Hello from server, please tell me your local clock value.") == 0) {

// prepare msg

string msg\_str = "Hello from client, my local clock value is " + to\_string(client\_local\_clock);

char msg\_char\_array[msg\_str.length() + 1];

strcpy(msg\_char\_array, msg\_str.c\_str());

// sending a message to server

send(client\_socket\_fd , &msg\_char\_array , strlen(msg\_char\_array) , 0 );

printf("Client: sent message: '%s'\n", msg\_char\_array); }

//

// second round communicattion

//

// receiving form server

valread = read( client\_socket\_fd , client\_read\_buffer, 1024);

printf("Client: read: '%s'\n",client\_read\_buffer );

// convert char array to string

recv\_msg = string(client\_read\_buffer);

if (recv\_msg.find("From server, your clock adjustment offset is") != string::npos){ // if latter is a substring of former

string substr\_after\_lastbutone\_space;

string substr\_after\_last\_space;

vector<string> split\_str = split(recv\_msg, " ");

substr\_after\_lastbutone\_space = split\_str[ split\_str.size() - 2 ];

substr\_after\_last\_space = split\_str[ split\_str.size() - 1 ];

cout << "Client: received local clock adjustment offset (string) is " << substr\_after\_lastbutone\_space << " " << substr\_after\_last\_space << endl;

float substr\_after\_last\_space\_f = stof(substr\_after\_last\_space);

cout << "Client: received local clock adjustment offset (float) is " << substr\_after\_lastbutone\_space << " " << substr\_after\_last\_space\_f << endl;

char oper\_char\_array[substr\_after\_lastbutone\_space.length() + 1];

strcpy(oper\_char\_array, substr\_after\_lastbutone\_space.c\_str());

if (strcmp(oper\_char\_array, "add") == 0 ){

client\_local\_clock += substr\_after\_last\_space\_f;

}else if (strcmp(oper\_char\_array, "minus") == 0 ){

client\_local\_clock -= substr\_after\_last\_space\_f;

} printf("Client local clock is %f \n\n", client\_local\_clock);

}

close(client\_socket\_fd);

return 0;}

**SERVER.CPP**

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#include <arpa/inet.h>

#include <vector>

#include <cstdlib>

#include <ctime>

#define PORT 8080

using namespace std;

// function for string delimiter

vector<string> split(string s, string delimiter) {

size\_t pos\_start = 0, pos\_end, delim\_len = delimiter.length();

string token;

vector<string> res;

while ((pos\_end = s.find (delimiter, pos\_start)) != string::npos) {

token = s.substr (pos\_start, pos\_end - pos\_start);

pos\_start = pos\_end + delim\_len;

res.push\_back (token);

}

res.push\_back (s.substr (pos\_start));

return res;

}

int main(int argc, char \*argv[])

{

// /\* deal with input arguments\*/

// std::cout << "print arguments:\nargc == " << argc << '\n';

// for(int ndx{}; ndx != argc; ++ndx) {

// std::cout << "argv[" << ndx << "] == " << argv[ndx] << '\n';

// }

// std::cout << "argv[" << argc << "] == "

// << static\_cast<void\*>(argv[argc]) << '\n';

srand((unsigned int)time(NULL)); // avoid always same output of rand()

float server\_local\_clock = rand() % 10; // range from 0 to 9

vector<float> clients\_local\_clocks;

printf("Sever starts. Server pid is %d \n", getpid());

printf("Server local clock is %f \n\n", server\_local\_clock);

// Socket Cite: https://www.geeksforgeeks.org/socket-programming-cc/?ref=lbp

int server\_socket\_fd, new\_socket, valread;

vector<int> client\_sockets;

vector<string> client\_ips;

vector<int> client\_ports;

struct sockaddr\_in server\_address;

server\_address.sin\_family = AF\_INET; // IPv4

server\_address.sin\_addr.s\_addr = INADDR\_ANY; // localhost

server\_address.sin\_port = htons( PORT ); // 8080

int opt = 1; // for setsockopt

// Creating socket file descriptor (IPv4, TCP, IP)

if ((server\_socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0)

{

perror("Server: socket failed");

exit(EXIT\_FAILURE);

}

// Optional: it helps in reuse of address and port. Prevents error such as: “address already in use”.

if (setsockopt(server\_socket\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT,

&opt, sizeof(opt)))

{

perror("Server: setsockopt");

exit(EXIT\_FAILURE);

}

// Forcefully attaching socket to the port 8080

if (bind(server\_socket\_fd, (struct sockaddr \*)&server\_address,

sizeof(server\_address))<0)

{

perror("Server: bind failed");

exit(EXIT\_FAILURE);

}

// Putting the server socket in a passive mode, waiting for the client to approach the server to make a connection

// The backlog=7, defines the maximum length to which the queue of pending connections for sockfd may grow.

// If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

if (listen(server\_socket\_fd, 7) < 0)

{

perror("Server: listen");

exit(EXIT\_FAILURE);

}

printf("Server: server is listening ...\n\nYou can open one or multiple new terminal windows now to run ./client\n");

int clients\_ctr = 0;

// Setting up buffer for receiving msg

char recv\_buf[65536];

memset(recv\_buf, '\0', sizeof(recv\_buf));

int in\_client\_enough = 0;

while ( in\_client\_enough == 0) { // block on accept() until positive fd or error

struct sockaddr\_in client\_addr;

socklen\_t length = sizeof(client\_addr);

// Extracting the first connection request on the queue of pending connections for the listening socket (server\_socket\_fd)

// Creates a new connected socket, and returns a new file descriptor referring to that socket

if ((new\_socket = accept(server\_socket\_fd, (struct sockaddr \*)&client\_addr,

(socklen\_t\*)&length))<0)

{

perror("Server: accept");

exit(EXIT\_FAILURE);

}

clients\_ctr ++;

printf("\nYou have connected %d client(s) now.", clients\_ctr);

// converting the network address structure src in the af address family into a character string.

char client\_ip[INET\_ADDRSTRLEN] = "";

inet\_ntop(AF\_INET, &client\_addr.sin\_addr, client\_ip, INET\_ADDRSTRLEN);

printf("Server: new client accepted. client ip and port: %s:%d\n", client\_ip, ntohs(client\_addr.sin\_port));

// store new client connection into array

client\_sockets.push\_back(new\_socket);

client\_ips.push\_back(client\_ip);

client\_ports.push\_back(ntohs(client\_addr.sin\_port));

printf("current connected clients amount is %d \n", int(client\_sockets.size()) );

cout << "Do you have enought clients? (please input '1' for yes, '0' for no):" ;

cin >> in\_client\_enough;

if (in\_client\_enough == 0){

cout << "OK. Please continute opening one or multiple new terminal windows to run ./client\n" << endl;

}else if (in\_client\_enough != 1){

cout << "Unrecognized input has been considered as 0. You can create one more client.\n" << endl;

in\_client\_enough = 0;

}}

printf("\nClients creation finished! There are totally %d connected clients.\n", int(client\_sockets.size()) );

printf("Asking all clients to report their local clock value ... \n\n\n");

for (int i = 0; i < client\_sockets.size(); i++){

// sending a message to client

const char \*msg = "Hello from server, please tell me your local clock value.";

send(client\_sockets[i] , msg , strlen(msg) , 0 );

printf("Server: sent to client(%s:%d): '%s'\n", client\_ips[i].c\_str(), client\_ports[i], msg);

// receiving

while(recv(client\_sockets[i], recv\_buf, sizeof(recv\_buf), 0) > 0 ){

printf("Server: recv from client(%s:%d): '%s' \n", client\_ips[i].c\_str(), client\_ports[i], recv\_buf);

// convert char array to string

string recv\_msg = string(recv\_buf);

if (recv\_msg.find("Hello from client, my local clock value is") != string::npos){

string substr\_after\_last\_space;

vector<string> split\_str = split(recv\_msg, " ");

substr\_after\_last\_space = split\_str[ split\_str.size() - 1 ];

cout << "Server: received client local clock (string) is " << substr\_after\_last\_space << endl;

float substr\_after\_last\_space\_f = stof(substr\_after\_last\_space);

cout << "Server: received client local clock (float) is " << substr\_after\_last\_space\_f << endl;

clients\_local\_clocks.push\_back(substr\_after\_last\_space\_f);

}

memset(recv\_buf, '\0', strlen(recv\_buf));

break;

}}

printf("\n\n");

// average clock values

float all\_clock\_sum = server\_local\_clock;

for (int i = 0; i < clients\_local\_clocks.size(); i++){

all\_clock\_sum += clients\_local\_clocks[i];

}

float avg\_clock = all\_clock\_sum / (client\_sockets.size() + 1);

// tell clients how to adjust

for (int i = 0; i < client\_sockets.size(); i++){

// prepare msg

float offset = clients\_local\_clocks[i] - avg\_clock;

string operation;

if (offset >= 0){

operation = "minus";

}else{

operation = "add";

offset = 0 - offset;

}

string msg\_str = "From server, your clock adjustment offset is " + operation + " " + to\_string(offset);

char msg\_char\_array[msg\_str.length() + 1];

strcpy(msg\_char\_array, msg\_str.c\_str());

// sending a message to client

send(client\_sockets[i] , &msg\_char\_array , strlen(msg\_char\_array) , 0 );

printf("Server: sent to client(%s:%d): '%s'\n", client\_ips[i].c\_str(), client\_ports[i], msg\_char\_array);

}

// adjust self

server\_local\_clock += avg\_clock - server\_local\_clock;

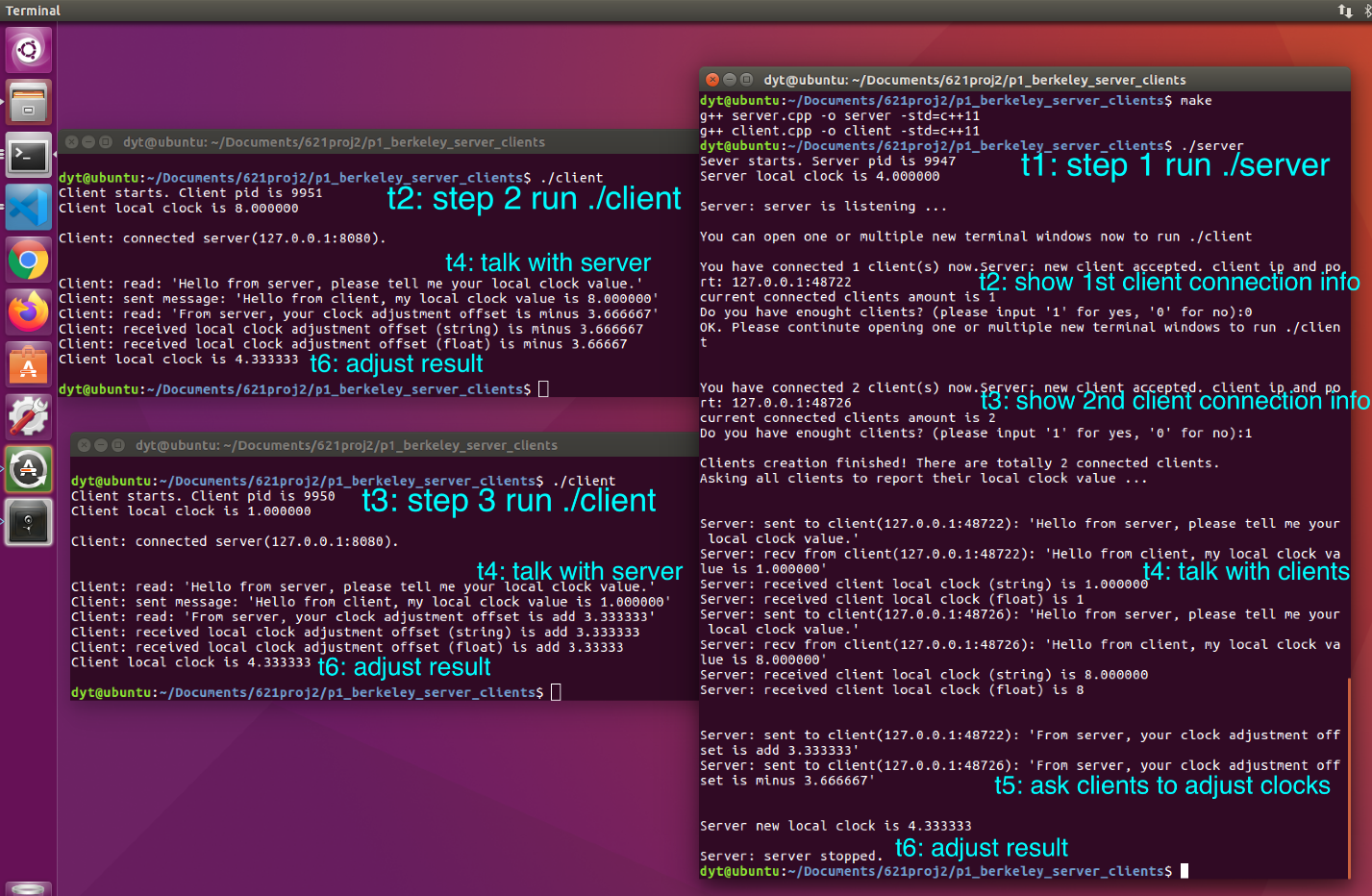
printf("\n\nServer new local clock is %f \n\n", server\_local\_clock);

printf("Server: server stopped. \n");

close(server\_socket\_fd);

return 0;

}

****

**Experient 9:Implementation of Lamport Algorithm**

**Main.java**

**import java.net.InetAddress;**

**import java.io.BufferedReader;**

**import java.io.InputStreamReader;**

**import java.util.Arrays;**

**import java.util.List;**

**public class Main {**

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

**if (args.length == 0) {**

**System.out.println("Usage: java Main (number of processses) [filename of commands]");**

**return;**

**}**

**// TODO: add support for reading commands from a file**

**String input;**

**try {**

**int n = Integer.parseInt(args[0]);**

**LamportClock[] clocks = new LamportClock[n];**

**System.setProperty("java.net.preferIPv4Stack" , "true");**

**InetAddress group = InetAddress.getByName("224.255.255.255");**

**for (int i = 0; i < n; ++i) {**

**int port = 8888;**

**LamportClock lc = new LamportClock(group, port, i);**

**lc.start();**

**clocks[i] = lc;**

**}**

**BufferedReader in = new BufferedReader(new InputStreamReader(System.in));**

**while (true) {**

**input = in.readLine();**

**if (input.equals("exit"))**

**return;**

**/\*\***

**\* A message format is of the following:**

**\* EVENT\_NAME ID\_OF\_SENDER (ID\_OF\_RECEIVER)**

**\***

**\* EVENT\_NAME is of the following:**

**\* - SEND**

**\* - LOCAL**

**\***

**\* For example:**

**\* SEND 1 2 (process 1 sends a message to process 2)**

**\* LOCAL 3 (process 3 performs a local event)**

**\*/**

**// perform a string split operation based on space**

**String[] splits = input.split(" ");**

**if (splits.length == 0) {**

**continue;**

**}**

**switch(splits[0].toUpperCase()) {**

**case "SEND":**

**int clockArrayId = Integer.parseInt(splits[1]);**

**long firstProcessId = clocks[clockArrayId].getId();**

**long secondProcessId = clocks[Integer.parseInt(splits[2])].getId();**

**String messageContent = "";**

**if (splits.length >= 3) {**

**List<String> wordsList = Arrays.asList(**

**Arrays.copyOfRange(splits, 3, splits.length));**

**messageContent = String.join(" ", wordsList);**

**}**

**Event e = new Event(1, firstProcessId, secondProcessId, messageContent);**

**clocks[clockArrayId].updateTime(e);**

**break;**

**case "LOCAL":**

**clockArrayId = Integer.parseInt(splits[1]);**

**firstProcessId = clocks[clockArrayId].getId();**

**secondProcessId = 0;**

**messageContent = "";**

**e = new Event(0, firstProcessId, secondProcessId, messageContent);**

**clocks[clockArrayId].updateTime(e);**

**break;**

**case "REQUEST":**

**clockArrayId = Integer.parseInt(splits[1]);**

**firstProcessId = clocks[clockArrayId].getId();**

**e = new Event(3, firstProcessId, -1, "");**

**clocks[clockArrayId].updateTime(e);**

**break;**

**default:**

**throw new RuntimeException("Invalid event name");**

**}**

**}**

**} catch(Exception e) {**

**System.err.println(e);**

**return;**

**}**

**}**

**}**

**EVENT.JAVA**

**public class Event {**

**public int type;**

**public long senderId;**

**public long receiverId;**

**public int localTime;**

**public String content;**

**public Event(int type, long senderId, long receiverId, String content) {**

**this.type = type;**

**this.senderId = senderId;**

**this.receiverId = receiverId;**

**this.content = content;**

**this.localTime = 0;**

**}**

**public Event(int type, long senderId,**

**long receiverId, int localTime, String content) {**

**this(type, senderId, receiverId, content);**

**this.localTime = localTime;**

**}**

**}**

**REQUEST.JAVA**

/\*\*

\* This class represents a request to be added to the priority queue

\*/

public class Request implements Comparable<Request> {

public int time;

public long processId;

public Request(int time, long processId) {

this.time = time;

this.processId = processId;

}

public int getTime() {

return this.time;

}

public long getProcessId() {

return this.processId;

}

@Override

public int compareTo(Request other) {

if (this.getTime() == other.getTime())

return 0;

else if (this.getTime() > other.getTime())

return 1;

else

return -1;

}}

**LAMPORTCLOCK.JAVA**

import java.net.DatagramPacket;

import java.net.InetAddress;

import java.net.MulticastSocket;

import java.lang.Thread;

import java.util.PriorityQueue;

import java.util.Random;

public class LamportClock extends Thread {

private MulticastSocket sock;

private InetAddress group;

private int port;

private PriorityQueue<Request> clockPQ;

// local time of a process

private int time;

// order of the process (viewed from the master's perspective)

private int order;

public LamportClock(InetAddress group, int port) throws Exception {

this.group = group;

this.port = port;

// if we don't assign an order to a process

this.order = -1;

// set local time to random

Random rand = new Random();

// this.time = rand.nextInt(10);

this.time = 0;

// initialize the priority queue

this.clockPQ = new PriorityQueue<>();

sock = new MulticastSocket(port);

sock.setTimeToLive(2);

sock.joinGroup(group);

}

public LamportClock(InetAddress group, int port, int order) throws Exception {

this(group, port);

this.order = order;

}

public int getOrder() {

return this.order;

}

public int getTime() {

return this.time;

}

public int localEvent() {

++this.time;

// System.out.println(this.getId() + " performing local event. local time is " + this.time);

return this.time;

}

public int receivedEvent(long senderId, int receivedTime) {

// System.out.println(this.getId() + " received message from "

// + senderId + ". local time is " + this.time);

return this.time;

}

public int sendEvent(String msg) throws Exception {

byte[] data = msg.getBytes();

DatagramPacket d = new DatagramPacket(data, data.length, group, port);

sock.send(d);

return this.time;

} public void updateTime(Event e) throws Exception {

int type = e.type;

switch (type) {

// LOCAL EVENT

case 0:

this.localEvent();

break;

// SEND EVENT

case 1: // extract information from the event

long senderId = e.senderId;

long receiverId = e.receiverId;

// increase the time first before sending the message

e.localTime = ++this.time;

String content = e.content;

/\*\* send a message of the following format

\* SENDER\_ID|RECEIVER\_ID|LOCAL\_TIME

\*/

String msg = Long.toString(senderId) + "-" + Long.toString(receiverId)

+ "-" + e.localTime + "-" + content;

sendEvent(msg);

break;

// RECEIVE EVENT

case 2:

// update its logical clock

this.time = Math.max(e.localTime, this.time) + 1;

break;

// REQUEST EVENT

case 3:

// update its local clock

e.localTime = ++this.time;

// add new request to the priority queue

clockPQ.add(new Request(this.time, this.getId()));

String requestContent = "REQUEST-" + this.time + "-" + this.getId();

sendEvent(requestContent);

break;

// REPLY REQUEST EVENT

case 4:

// update its local clock

++this.time;

// add new request to the priority queue

clockPQ.add(new Request(e.localTime, e.senderId));

break;

// REPLY EVENT

case 5:

e.localTime = ++this.time;

senderId = e.senderId;

break;

// ACK EVENT

case 6:

// update its local clock

++this.time;

break;

// ACK EVENT

default:

break;

}

printTime(e);

}

public void printTime(Event e) {

String logging = "-------------------------\n";

logging += "Process " + this.getId() + "\n";

logging += "Process' local time " + this.getTime() + "\n";

logging += "\tEvent type: ";

switch(e.type) {

case 0:

logging += "LOCAL EVENT\n";

break;

case 1:

logging += "SEND EVENT\n";

break;

case 2:

logging += "RECEIVE EVENT\n";

break;

case 3:

logging += "REQUEST EVENT\n";

break;

case 4:

logging += "RECEIVE REQUEST EVENT\n";

break;

case 5:

logging += "REPLY EVENT\n";

break;

case 6:

logging += "ACK EVENT\n";

break;

default:

break;

} logging += "\tEvent sender's ID: " + e.senderId + "\n";

logging += "\tEvent receiver's ID: " + e.receiverId + "\n";

logging += "\tEvent local time: " + e.localTime + "\n";

logging += "\tEvent content: " + e.content + "\n";

logging += "-------------------------\n";

System.out.print(logging);

} public void run() {

String greeting = "";

greeting = "Unique ID " + this.getId() +

" is initialized with local clock " + this.time;

if (this.order != -1)

greeting = "Process " + this.order + " " + greeting;

System.out.println(greeting);

try {

while (true) {

DatagramPacket d = new DatagramPacket(new byte[256], 256);

sock.receive(d);

String s = new String(d.getData());

// System.out.println(this.getId() + " received " + s);

String[] meta = s.trim().split("-");

// if this is a REQUEST event

if (meta[0].equals("REQUEST")) {

int requestTime = Integer.parseInt(meta[1]);

long senderId = Long.parseLong(meta[2]);

if (this.getId() != senderId) {

// create a RECEIVE event for every clock

Event e = new Event(4, senderId, this.getId(), requestTime, "");

updateTime(e);

}

// if this is a REPLY event

} else if (meta[0].equals("REPLY")) {

} else {

long senderId = Long.parseLong(meta[0]);

long receiverId = Long.parseLong(meta[1]);

int localTime = Integer.parseInt(meta[2]);

String content = "";

// if there is a message

if (meta.length >= 4)

content = meta[3];

if (this.getId() == receiverId) {

Event e = new Event(2, senderId, receiverId, localTime, content);

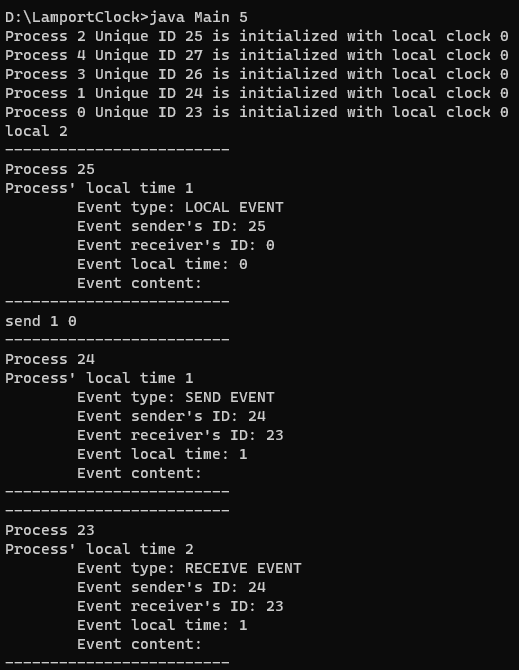
updateTime(e);

}}}

} catch (Exception e) {

System.err.println("LC Failed: " + e);

return;}}}

****