**Ex.No: 4:** **Implementation of Client-Server Full-Duplex Stop and Wait protocol for Multimodal File Transmission using UDP protocol**

**AIM:**

Write a socket program in Java for Multimodal File Transmission using UDP with Full-Duplex Stop and Wait protocol. The program/protocol should support the following properties/mechanism

1. The protocol will send any type of files
2. Each packet should consist of the file name, sequence number/Acknowledgement number
3. A log file should be generated with some information like,

List of uncommon files in server and client which are to be transferred, **Priority of each file,** Start time, If the connection is broken then the % of the file already uploaded, How many times connections were established during the complete transmission, End time (when the file is fully transmitted), How many packets are lost, How many time-outs are occurred, etc.

1. **Prioritize the files to be transferred and send the files accordingly.**

**DESCRIPTION:**

Client will mention its set of files and server will mention its set of files and then before transferring files they will sort the files according to the priority and then while exchanging the files the client/server will let the other party know that whether they have the file or not. If they are not having file then transfer of file will proceed otherwise they will be checking for the next file. Both will use the UDP protocol and stop and wait protocol to transfer only uncommon files will get transferred. Also for each and every packet an acknowledgement will be sent stating the requirement of the next sequence number.

At the end the log file will be generated along with the details of the files which are transferred from server to client or vice versa.

**ALGORITHM:**

Server

1. Initially sort the files according to the priority before sending it to the client.
2. Create two datagram socket and bind it to two different port. One socket will be used for sending the file and receiving the acknowledgement and the other socket will be used for receiving the file and sending the acknowledgement using stop and wait protocol.
3. Create a multithread program and call two threads with different port.
4. Initially send the file name along with the size and wait for the reply from the client.
5. If Client replies that it is already having the file then repeat step 3 for next file. If the client replies that it is ready to accept the file then start sending the files along with the sequence number.
6. If the Client is sending any file then check that whether server is having that file or not. If server is not having then send the message to client that the server is ready to receive otherwise deny that server is already having the file.
7. While receiving any file from client, extract the sequence number from the packet and send the acknowledgement of the next expected sequence number.
8. Follow the above steps for all the files and log the activities.
9. Close all the streams once transfer is done.
10. Ensure that you have a log file generated at the end of all transfers.
11. Stop

Client

1. Initially sort the files according to the priority before sending it to the client.
2. Create two datagram socket and bind it to two different port. One socket will be used for sending the file and receiving the acknowledgement and the other socket will be used for receiving the file and sending the acknowledgement using stop and wait protocol.
3. Create a multithread program and call two threads with different port.
4. Initially send the file name along with the size and wait for the reply from the client.
5. If server replies that it is already having the file then repeat step 3 for next file. If the server replies that it is ready to accept the file then start sending the files along with the sequence number.
6. If the server is sending any file then check that whether server is having that file or not. If server is not having then send the message to server that the client is ready to receive otherwise deny that client is already having the file.
7. While receiving any file from server, extract the sequence number from the packet and send the acknowledgement of the next expected sequence number.
8. Follow the above steps for all the files and log the activities.
9. Close all the streams once transfer is done.
10. Ensure that you have a log file generated at the end of all transfers.
11. Stop

**PROGRAM:**

// Assignment4UDPServerFTP.java

import java.io.\*;

import java.net.\*;

import java.nio.ByteBuffer;

import java.time.LocalTime;

import java.util.Arrays;

import java.util.logging.FileHandler;

import java.util.logging.Logger;

import java.util.logging.SimpleFormatter;

class ServerThread1 extends Thread {

    private static final int BUFFER\_SIZE = 1024;

    private static final int BASE\_SEQUENCE\_NUMBER = 0;

    DatagramSocket sendSocket, receiveSocket;

    String work;

    String[] srcFiles;

    Logger loggerFile;

    ServerThread1(DatagramSocket outSocket, DatagramSocket inSocket, String dowork, String[] fileNameSource,

            Logger logger) {

        sendSocket = outSocket;

        receiveSocket = inSocket;

        work = dowork;

        srcFiles = fileNameSource;

        loggerFile = logger;

    }

    public void run() {

        if (work.equals("send")) {

            boolean checkConnection = true;

            boolean timedOut = true;

            int noOfEstablishedConnections = 0;

            int totalNoOfPacketLost = 0;

            int noOfPacketLost = 0;

            for (int i = 0; i < this.srcFiles.length; i++) {

                noOfPacketLost = 0;

                try {

                    InetAddress id = InetAddress.getByName("192.168.142.129");

                    File sendingFile = new File(this.srcFiles[i]);

                    long fileSize = sendingFile.length();

                    byte[] buffer = new byte[(int) fileSize];

                    InputStream fis = new FileInputStream(sendingFile);

                    // BufferedInputStream bis = new BufferedInputStream(fis);

                    fis.read(buffer, 0, buffer.length);

                    checkConnection = true;

                    timedOut = true;

                    String acknowledgement = "";

                    while (timedOut) {

                        try {

                            // Establish the connection before sending any file

                            String conn = this.srcFiles[i] + " " + buffer.length;

                            System.out.println("Sending a file to client: " + " " + conn);

                            byte[] sendBytes = conn.getBytes();

                            DatagramPacket sendDatagramPacket = new DatagramPacket(sendBytes, conn.length(), id, 9876);

                            sendSocket.send(sendDatagramPacket);

                            byte[] receiveBytes = new byte[BUFFER\_SIZE];

                            DatagramPacket receiveDatagramPacket = new DatagramPacket(receiveBytes,

                                    receiveBytes.length);

                            sendSocket.receive(receiveDatagramPacket);

                            acknowledgement = new String(receiveDatagramPacket.getData(), 0,

                                    receiveDatagramPacket.getLength());

                            System.out.println("Client: " + acknowledgement);

                            timedOut = false;

                        } catch (SocketTimeoutException exception) {

                            System.out.println("Timeout in receiving the client's status");

                            noOfPacketLost++;

                            checkConnection = false;

                        } catch (IOException exception) {

                            System.out.println("Network is not reachable");

                            noOfPacketLost++;

                            checkConnection = false;

                        }

                    }

                    if (!checkConnection) {

                        noOfEstablishedConnections++;

                        checkConnection = true;

                        loggerFile.warning(

                                "Connection was failed but established again. No of packet lost = " + noOfPacketLost);

                        totalNoOfPacketLost += noOfPacketLost;

                        noOfPacketLost = 0;

                    }

                    if (acknowledgement.substring(0, 5).equals("Sorry")) {

                        // System.out.println("Continue: " + acknowledgement);

                        continue;

                    }

                    // Start Sending file - Full Duplex

                    Integer sequenceNumber = BASE\_SEQUENCE\_NUMBER;

                    LocalTime startTime = LocalTime.now();

                    System.out.println("startTime: " + startTime);

                    loggerFile.info("Start Time: " + startTime + " - Started to send the file: " + this.srcFiles[i]

                            + " from server to client of size: " + buffer.length);

                    int counter = 0;

                    int lastCounterVal = 0;

                    while (counter < buffer.length) {

                        sequenceNumber++;

                        lastCounterVal = counter;

                        String packetData = this.srcFiles[i] + " - Sequence Number: " + sequenceNumber;

                        // System.out.println("Sending: " + packetData);

                        byte[] fileName = this.srcFiles[i].getBytes();

                        byte[] fileNameLen = ("" + this.srcFiles[i].length()).getBytes();

                        byte[] seqNum = sequenceNumber.toString().getBytes();

                        byte[] sendData = new byte[BUFFER\_SIZE + 120];

                        System.arraycopy(fileNameLen, 0, sendData, 0, fileNameLen.length);

                        System.arraycopy(fileName, 0, sendData, 5, fileName.length);

                        System.arraycopy(seqNum, 0, sendData, 105, seqNum.length);

                        for (int j = 120; j < BUFFER\_SIZE + 120; j++) {

                            sendData[j] = buffer[counter++];

                            if (counter == buffer.length)

                                break;

                        }

                        timedOut = true;

                        while (timedOut) {

                            try {

                                DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 9876);

                                sendSocket.send(packet);

                                Thread.sleep(15);

                                byte[] receiveData = new byte[BUFFER\_SIZE];

                                DatagramPacket received = new DatagramPacket(receiveData, receiveData.length);

                                sendSocket.receive(received);

                                String msgFromServer = new String(received.getData(), 0, received.getLength());

                                // System.out.println("Client: " + msgFromServer);

                                timedOut = false;

                            } catch (SocketTimeoutException exception) {

                                System.out.println("Packet of file " + this.srcFiles[i] + " and of sequence number "

                                        + sequenceNumber + " is lost");

                                noOfPacketLost++;

                                checkConnection = false;

                            } catch (InterruptedException exception) {

                                System.out.println(exception);

                                noOfPacketLost++;

                                checkConnection = false;

                            } catch (IOException exception) {

                                System.out.println("Network is not reachable.");

                                System.out.println("Packet of file " + this.srcFiles[i] + " and of sequence number "

                                        + sequenceNumber + " is lost");

                                noOfPacketLost++;

                                checkConnection = false;

                            }

                        }

                        if (!checkConnection) {

                            double percentFileSent = (lastCounterVal / buffer.length) / 100;

                            noOfEstablishedConnections++;

                            checkConnection = true;

                            loggerFile.info("Failed Connection: " + percentFileSent + "% of " + this.srcFiles[i]

                                    + "file was successfully transferred");

                            loggerFile.warning("Connection was failed but established again. Number of Packet of file "

                                    + this.srcFiles[i] + " and of sequence number " + sequenceNumber + " lost = "

                                    + noOfPacketLost);

                            totalNoOfPacketLost += noOfPacketLost;

                            noOfPacketLost = 0;

                        }

                    }

                    timedOut = true;

                    while (timedOut) {

                        try {

                            byte[] sendData = "EOF".getBytes();

                            DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 9876);

                            sendSocket.send(packet);

                            Thread.sleep(15);

                            byte[] receiveData = new byte[BUFFER\_SIZE];

                            DatagramPacket received = new DatagramPacket(receiveData, receiveData.length);

                            sendSocket.receive(received);

                            String msgFromClient = new String(received.getData(), 0, received.getLength());

                            System.out.println("Client: " + msgFromClient);

                            timedOut = false;

                        } catch (SocketTimeoutException exception) {

                            System.out.println("EOF msg is lost");

                            noOfPacketLost++;

                            checkConnection = false;

                        } catch (InterruptedException exception) {

                            System.out.println(exception);

                            noOfPacketLost++;

                            checkConnection = false;

                        } catch (IOException exception) {

                            System.out.println("Network is not reachable.");

                            System.out.println("EOF msg is lost");

                            noOfPacketLost++;

                            checkConnection = false;

                        }

                    }

                    if (!checkConnection) {

                        noOfEstablishedConnections++;

                        checkConnection = true;

                        loggerFile.warning("Connection was failed but established again. EOF msg is lost");

                        totalNoOfPacketLost += noOfPacketLost;

                        noOfPacketLost = 0;

                    }

                    LocalTime endTime = LocalTime.now();

                    System.out.println("endTime: " + endTime);

                    loggerFile.info("End Time: " + endTime + " - File: " + this.srcFiles[i]

                            + " is fully transmitted from server to client of size: " + buffer.length);

                    fis.close();

                } catch (IOException ex) {

                    System.out.println(ex);

                }

            }

            timedOut = true;

            while (timedOut) {

                try {

                    InetAddress id = InetAddress.getByName("192.168.142.129");

                    byte[] sendData = "Bye".getBytes();

                    DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 9876);

                    sendSocket.send(packet);

                    Thread.sleep(15);

                    byte[] receiveData = new byte[BUFFER\_SIZE];

                    DatagramPacket received = new DatagramPacket(receiveData, receiveData.length);

                    sendSocket.receive(received);

                    String msgFromClient = new String(received.getData(), 0, received.getLength());

                    System.out.println("Client: " + msgFromClient);

                    timedOut = false;

                } catch (SocketTimeoutException exception) {

                    System.out.println("Bye msg is lost");

                    noOfPacketLost++;

                    checkConnection = false;

                } catch (InterruptedException exception) {

                    System.out.println(exception);

                    noOfPacketLost++;

                    checkConnection = false;

                } catch (IOException exception) {

                    System.out.println("Network is not reachable.");

                    System.out.println("Bye msg is lost");

                    noOfPacketLost++;

                    checkConnection = false;

                }

            }

            if (!checkConnection) {

                noOfEstablishedConnections++;

                checkConnection = true;

                loggerFile.warning("Connection was failed but established again. Bye msg is lost");

                totalNoOfPacketLost += noOfPacketLost;

                noOfPacketLost = 0;

            }

            loggerFile.info("Total number of Connections Established: " + noOfEstablishedConnections);

            loggerFile.info("Total number of Packets Lost: " + totalNoOfPacketLost);

        } else {

            String oldmsg = "";

            boolean moreFiles = true;

            while (moreFiles) {

                try {

                    InetAddress id = InetAddress.getByName("192.168.142.129");

                    boolean checkFilebool = true;

                    String[] fileDetails = new String[2];

                    String msg = "";

                    boolean eof = false;

                    while (checkFilebool) {

                        byte[] receiveBytes = new byte[512];

                        DatagramPacket receiveDatagramPacket = new DatagramPacket(receiveBytes, receiveBytes.length);

                        receiveSocket.receive(receiveDatagramPacket);

                        msg = new String(receiveDatagramPacket.getData(), 0, receiveDatagramPacket.getLength());

                        System.out.println("Client: " + msg);

                        fileDetails = msg.split(" ");

                        if (msg.equalsIgnoreCase("EOF")) {

                            checkFilebool = false;

                            eof = true;

                        } else if (msg.equalsIgnoreCase("Bye")) {

                            checkFilebool = false;

                            moreFiles = false;

                        } else if (msg != oldmsg && fileDetails[0].substring(0, 33)

                                .equalsIgnoreCase("/home/vmtwo/Desktop/ClientSource/")) {

                            checkFilebool = false;

                            oldmsg = msg;

                        }

                    }

                    if (!moreFiles || eof) {

                        byte[] sendBytes = "Thank You".getBytes();

                        DatagramPacket sendDatagramPacket = new DatagramPacket(sendBytes, sendBytes.length, id, 6789);

                        receiveSocket.send(sendDatagramPacket);

                        continue;

                    }

                    // Establish the connection before sending any file

                    String[] arr = fileDetails[0].split("/");

                    String checkFile = "/home/vmone/Desktop/ServerSource/" + arr[arr.length - 1];

                    String conn;

                    if (Arrays.asList(this.srcFiles).contains(checkFile)) {

                        conn = "Sorry Server is already having this file: " + checkFile;

                        System.out.println("Sending to Client: " + conn);

                        byte[] sendBytes = conn.getBytes();

                        DatagramPacket sendDatagramPacket = new DatagramPacket(sendBytes, conn.length(), id, 6789);

                        receiveSocket.send(sendDatagramPacket);

                        continue;

                    }

                    conn = "Server is ready to receive: " + msg;

                    System.out.println("Sending to Client: " + conn);

                    byte[] sendBytes = conn.getBytes();

                    DatagramPacket sendDatagramPacket = new DatagramPacket(sendBytes, conn.length(), id, 6789);

                    receiveSocket.send(sendDatagramPacket);

                    // Start Receiving file - Full Duplex

                    String newFile = "/home/vmone/Desktop/ServerDestination/" + arr[arr.length - 1];

                    OutputStream fos = new FileOutputStream(newFile);

                    int fileSize = Integer.parseInt(fileDetails[1]);

                    LocalTime startTime = LocalTime.now();

                    System.out.println("startReceiveTime: " + startTime);

                    loggerFile.info("Start Time: " + startTime + " - Started to receive the file: " + fileDetails[0]

                            + " from client to server of size: " + fileSize);

                    int counter = 0;

                    while (counter < fileSize + 1) {

                        boolean checkConnection = true;

                        int lastSeqNo = -1;

                        while (checkConnection) {

                            try {

                                byte[] receiveData = new byte[BUFFER\_SIZE + 120];

                                DatagramPacket received = new DatagramPacket(receiveData, receiveData.length);

                                receiveSocket.receive(received);

                                String msg1 = new String(received.getData(), 0, received.getLength());

                                if (msg1.equalsIgnoreCase("EOF")) {

                                    byte[] sending = "Thank You".getBytes();

                                    DatagramPacket sendPacket = new DatagramPacket(sending, sending.length, id, 6789);

                                    receiveSocket.send(sendPacket);

                                    checkConnection = false;

                                    counter = fileSize + 1;

                                    continue;

                                }

                                // Extract Data

                                byte[] fileNameLen = new byte[5];

                                byte[] seqNum = new byte[10];

                                System.arraycopy(receiveData, 0, fileNameLen, 0, fileNameLen.length);

                                int fileNmLen = Integer.parseInt(

                                        (new String(fileNameLen, 0, fileNameLen.length).replaceAll("\u0000.\*", "")));

                                byte[] fileName = new byte[fileNmLen];

                                System.arraycopy(receiveData, 5, fileName, 0, fileName.length);

                                System.arraycopy(receiveData, 105, seqNum, 0, seqNum.length);

                                String fileNm = (new String(fileName, 0, fileName.length).replaceAll("\u0000.\*", ""));

                                int seqNo = Integer

                                        .parseInt((new String(seqNum, 0, seqNum.length).replaceAll("\u0000.\*", "")));

                                if (lastSeqNo == seqNo) {

                                    String acknowledgement = "Already Received seq no " + seqNo;

                                    // System.out.println("Sending Acknowledgement: " + acknowledgement);

                                    byte[] sendData = acknowledgement.getBytes();

                                    DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 6789);

                                    receiveSocket.send(packet);

                                    continue;

                                } else {

                                    lastSeqNo = seqNo;

                                    checkConnection = false;

                                }

                                fos.write(receiveData, 120, receiveData.length - 120);

                                fos.flush();

                                // System.out.println(fileNm + " Sequence Number: " + seqNo + " received");

                                String acknowledgement;

                                if (counter < fileSize)

                                    acknowledgement = fileNm + " Waiting for Sequence Number: " + (seqNo + 1);

                                else

                                    acknowledgement = "Received whole file - " + fileNm;

                                // System.out.println("Sending Acknowledgement: " + acknowledgement);

                                byte[] sendData = acknowledgement.getBytes();

                                DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 6789);

                                receiveSocket.send(packet);

                                counter = counter + receiveData.length - 120;

                            } catch (IOException ex) {

                                checkConnection = true;

                            }

                        }

                    }

                    LocalTime endTime = LocalTime.now();

                    System.out.println("endReceiveTime: " + endTime);

                    loggerFile.info("End Time: " + endTime + " - File: " + fileDetails[0]

                            + " is successfully received from client to server of size: " + fileSize);

                    fos.close();

                } catch (IOException ex) {

                    System.out.println(ex);

                }

            }

        }

    }

}

class Assignment4UDPServerFTP {

    public static String[] fileNameSource = { "/home/vmone/Desktop/ServerSource/ServerFile.png",

            "/home/vmone/Desktop/ServerSource/ServerFile.mp4", "/home/vmone/Desktop/ServerSource/ServerFile.pdf",

            "/home/vmone/Desktop/ServerSource/ClientFile.mp4", "/home/vmone/Desktop/ServerSource/ServerFile.txt" };

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

        Assignment4UDPServerFTP obj = new Assignment4UDPServerFTP();

        Logger logger = obj.LoggingFiles();

        String[] priority = { ".txt", ".png", ".pdf", ".mp4" };

        String[] setpriority = new String[fileNameSource.length];

        int count = 0;

        for (int i = 0; i < priority.length; i++) {

            for (int j = 0; j < fileNameSource.length; j++) {

                if (fileNameSource[j].substring(fileNameSource[j].length() - 4).equals(priority[i])) {

                    setpriority[count++] = fileNameSource[j];

                }

            }

        }

        DatagramSocket sendSocket = new DatagramSocket(1234);

        DatagramSocket receiveSocket = new DatagramSocket(4321);

        sendSocket.setSoTimeout(2000);

        ServerThread1 sct = new ServerThread1(sendSocket, receiveSocket, "send", setpriority, logger);

        sct.start();

        ServerThread1 rct = new ServerThread1(sendSocket, receiveSocket, "receive", setpriority, logger);

        rct.start();

    }

    public Logger LoggingFiles() {

        Logger logger = Logger.getLogger("ServerLog");

        FileHandler fh;

        try {

            // This block configure the logger with handler and formatter

            fh = new FileHandler("/home/vmone/Desktop/ServerDestination/ServerLogFile.log");

            logger.addHandler(fh);

            SimpleFormatter formatter = new SimpleFormatter();

            fh.setFormatter(formatter);

            // the following statement is used to log any messages

            logger.setUseParentHandlers(false);

            logger.info("Uncommon Files Information");

        } catch (SecurityException e) {

            e.printStackTrace();

        } catch (IOException e) {

            e.printStackTrace();

        }

        return logger;

    }

}

// Assignment4UDPClientFTP.java

import java.io.\*;

import java.net.\*;

import java.time.LocalTime;

import java.util.Arrays;

import java.util.logging.FileHandler;

import java.util.logging.Logger;

import java.util.logging.SimpleFormatter;

class ClientThread1 extends Thread {

    private static final int BUFFER\_SIZE = 1024;

    private static final int BASE\_SEQUENCE\_NUMBER = 0;

    DatagramSocket sendSocket, receiveSocket;

    String work;

    String[] srcFiles;

    Logger loggerFile;

    ClientThread1(DatagramSocket outSocket, DatagramSocket inSocket, String dowork, String[] fileNameSource,

            Logger logger) {

        sendSocket = outSocket;

        receiveSocket = inSocket;

        work = dowork;

        srcFiles = fileNameSource;

        loggerFile = logger;

    }

    public void run() {

        if (work.equals("send")) {

            int noOfEstablishedConnections = 0;

            int totalNoOfPacketLost = 0;

            boolean checkConnection = true;

            boolean timedOut = true;

            int noOfPacketLost = 0;

            for (int i = 0; i < this.srcFiles.length; i++) {

                noOfPacketLost = 0;

                try {

                    InetAddress id = InetAddress.getByName("192.168.142.128");

                    File sendingFile = new File(this.srcFiles[i]);

                    long fileSize = sendingFile.length();

                    byte[] buffer = new byte[(int) fileSize];

                    InputStream fis = new FileInputStream(sendingFile);

                    // BufferedInputStream bis = new BufferedInputStream(fis);

                    fis.read(buffer, 0, buffer.length);

                    checkConnection = true;

                    timedOut = true;

                    String acknowledgement = "";

                    while (timedOut) {

                        try {

                            // Establish the connection before sending any file

                            String conn = this.srcFiles[i] + " " + buffer.length;

                            System.out.println("Sending a file to server: " + " " + conn);

                            byte[] sendBytes = conn.getBytes();

                            DatagramPacket sendDatagramPacket = new DatagramPacket(sendBytes, conn.length(), id, 4321);

                            sendSocket.send(sendDatagramPacket);

                            byte[] receiveBytes = new byte[512];

                            DatagramPacket receiveDatagramPacket = new DatagramPacket(receiveBytes,

                                    receiveBytes.length);

                            sendSocket.receive(receiveDatagramPacket);

                            acknowledgement = new String(receiveDatagramPacket.getData(), 0,

                                    receiveDatagramPacket.getLength());

                            System.out.println("Server: " + acknowledgement);

                            timedOut = false;

                        } catch (SocketTimeoutException exception) {

                            System.out.println("Timeout in receiving the server's status");

                            noOfPacketLost++;

                            checkConnection = false;

                        } catch (IOException exception) {

                            System.out.println("Network is not reachable");

                            noOfPacketLost++;

                            checkConnection = false;

                        }

                    }

                    if (!checkConnection) {

                        noOfEstablishedConnections++;

                        checkConnection = true;

                        loggerFile.warning(

                                "Connection was failed but established again. No of packet lost = " + noOfPacketLost);

                        totalNoOfPacketLost += noOfPacketLost;

                        noOfPacketLost = 0;

                    }

                    if (acknowledgement.substring(0, 5).equals("Sorry")) {

                        //System.out.println("Continue: " + acknowledgement);

                        continue;

                    }

                    // Start Sending file - Full Duplex

                    Integer sequenceNumber = BASE\_SEQUENCE\_NUMBER;

                    LocalTime startTime = LocalTime.now();

                    System.out.println("startTime: " + startTime);

                    loggerFile.info("Start Time: " + startTime + " - Started to send the file: " + this.srcFiles[i]

                            + " from client to server of size: " + buffer.length);

                    int counter = 0;

                    int lastCounterVal = 0;

                    while (counter < buffer.length) {

                        sequenceNumber++;

                        lastCounterVal = counter;

                        String packetData = this.srcFiles[i] + " - Sequence Number: " + sequenceNumber;

                        // System.out.println("Sending: " + packetData);

                        byte[] fileName = this.srcFiles[i].getBytes();

                        byte[] fileNameLen = ("" + this.srcFiles[i].length()).getBytes();

                        byte[] seqNum = sequenceNumber.toString().getBytes();

                        byte[] sendData = new byte[BUFFER\_SIZE + 120];

                        System.arraycopy(fileNameLen, 0, sendData, 0, fileNameLen.length);

                        System.arraycopy(fileName, 0, sendData, 5, fileName.length);

                        System.arraycopy(seqNum, 0, sendData, 105, seqNum.length);

                        for (int j = 120; j < BUFFER\_SIZE + 120; j++) {

                            sendData[j] = buffer[counter++];

                            if (counter == buffer.length)

                                break;

                        }

                        timedOut = true;

                        while (timedOut) {

                            try {

                                DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 4321);

                                sendSocket.send(packet);

                                Thread.sleep(15);

                                byte[] receiveData = new byte[BUFFER\_SIZE];

                                DatagramPacket received = new DatagramPacket(receiveData, receiveData.length);

                                sendSocket.receive(received);

                                String msgFromServer = new String(received.getData(), 0, received.getLength());

                                // System.out.println("Server: " + msgFromServer);

                                timedOut = false;

                            } catch (SocketTimeoutException exception) {

                                System.out.println("Packet of file " + this.srcFiles[i] + " and of sequence number "

                                        + sequenceNumber + " is lost");

                                noOfPacketLost++;

                                checkConnection = false;

                            } catch (InterruptedException exception) {

                                System.out.println(exception);

                                noOfPacketLost++;

                                checkConnection = false;

                            } catch (IOException exception) {

                                System.out.println("Network is not reachable");

                                System.out.println("Packet of file " + this.srcFiles[i] + " and of sequence number "

                                        + sequenceNumber + " is lost");

                                noOfPacketLost++;

                                checkConnection = false;

                            }

                        }

                        if (!checkConnection) {

                            double percentFileSent = (lastCounterVal / buffer.length) / 100;

                            noOfEstablishedConnections++;

                            checkConnection = true;

                            loggerFile.info("Failed Connection: " + percentFileSent + "% of " + this.srcFiles[i]

                                    + "file was successfully transferred");

                            loggerFile.warning("Connection was failed but established again. Number of Packet of file "

                                    + this.srcFiles[i] + " and of sequence number " + sequenceNumber + " lost = "

                                    + noOfPacketLost);

                            totalNoOfPacketLost += noOfPacketLost;

                            noOfPacketLost = 0;

                        }

                    }

                    timedOut = true;

                    while (timedOut) {

                        try {

                            byte[] sendData = "EOF".getBytes();

                            DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 4321);

                            sendSocket.send(packet);

                            Thread.sleep(15);

                            byte[] receiveData = new byte[BUFFER\_SIZE];

                            DatagramPacket received = new DatagramPacket(receiveData, receiveData.length);

                            sendSocket.receive(received);

                            String msgFromServer = new String(received.getData(), 0, received.getLength());

                            System.out.println("Server: " + msgFromServer);

                            timedOut = false;

                        } catch (SocketTimeoutException exception) {

                            System.out.println("EOF msg is lost");

                            noOfPacketLost++;

                            checkConnection = false;

                        } catch (InterruptedException exception) {

                            System.out.println(exception);

                            noOfPacketLost++;

                            checkConnection = false;

                        } catch (IOException exception) {

                            System.out.println("Network is not reachable.");

                            System.out.println("EOF msg is lost");

                            noOfPacketLost++;

                            checkConnection = false;

                        }

                    }

                    if (!checkConnection) {

                        noOfEstablishedConnections++;

                        checkConnection = true;

                        loggerFile.warning("Connection was failed but established again. EOF msg is lost");

                        totalNoOfPacketLost += noOfPacketLost;

                        noOfPacketLost = 0;

                    }

                    LocalTime endTime = LocalTime.now();

                    System.out.println("endTime: " + endTime);

                    loggerFile.info("End Time: " + endTime + " - File: " + this.srcFiles[i]

                            + " is fully transmitted from client to server of size: " + buffer.length);

                    fis.close();

                } catch (IOException ex) {

                    System.out.println(ex);

                }

            }

            timedOut = true;

            while (timedOut) {

                try {

                    InetAddress id = InetAddress.getByName("192.168.142.128");

                    byte[] sendData = "Bye".getBytes();

                    DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 4321);

                    sendSocket.send(packet);

                    Thread.sleep(15);

                    byte[] receiveData = new byte[BUFFER\_SIZE];

                    DatagramPacket received = new DatagramPacket(receiveData, receiveData.length);

                    sendSocket.receive(received);

                    String msgFromServer = new String(received.getData(), 0, received.getLength());

                    System.out.println("Server: " + msgFromServer);

                    timedOut = false;

                } catch (SocketTimeoutException exception) {

                    System.out.println("Bye msg is lost");

                    noOfPacketLost++;

                    checkConnection = false;

                } catch (InterruptedException exception) {

                    System.out.println(exception);

                    noOfPacketLost++;

                    checkConnection = false;

                } catch (IOException exception) {

                    System.out.println("Network is not reachable.");

                    System.out.println("Bye msg is lost");

                    noOfPacketLost++;

                    checkConnection = false;

                }

            }

            if (!checkConnection) {

                noOfEstablishedConnections++;

                checkConnection = true;

                loggerFile.warning("Connection was failed but established again. Bye msg is lost");

                totalNoOfPacketLost += noOfPacketLost;

                noOfPacketLost = 0;

            }

            loggerFile.info("Total number of Connections Established: " + noOfEstablishedConnections);

            loggerFile.info("Total number of Packets Lost: " + totalNoOfPacketLost);

        } else {

            String oldmsg = "";

            boolean moreFiles = true;

            while (moreFiles) {

                try {

                    InetAddress id = InetAddress.getByName("192.168.142.128");

                    boolean checkFilebool = true;

                    String[] fileDetails = new String[2];

                    String msg = "";

                    boolean eof = false;

                    while (checkFilebool) {

                        byte[] receiveBytes = new byte[512];

                        DatagramPacket receiveDatagramPacket = new DatagramPacket(receiveBytes, receiveBytes.length);

                        receiveSocket.receive(receiveDatagramPacket);

                        msg = new String(receiveDatagramPacket.getData(), 0, receiveDatagramPacket.getLength());

                        System.out.println("Server: " + msg);

                        fileDetails = msg.split(" ");

                        if (msg.equalsIgnoreCase("EOF")) {

                            checkFilebool = false;

                            eof = true;

                        } else if (msg.equalsIgnoreCase("Bye")) {

                            checkFilebool = false;

                            moreFiles = false;

                        } else if (msg != oldmsg

                                && fileDetails[0].substring(0, 33).equals("/home/vmone/Desktop/ServerSource/")) {

                            checkFilebool = false;

                            oldmsg = msg;

                        }

                    }

                    if (!moreFiles || eof) {

                        byte[] sendBytes = "Thank You".getBytes();

                        DatagramPacket sendDatagramPacket = new DatagramPacket(sendBytes, sendBytes.length, id, 1234);

                        receiveSocket.send(sendDatagramPacket);

                        continue;

                    }

                    // Establish the connection before sending any file

                    String[] arr = fileDetails[0].split("/");

                    String checkFile = "/home/vmtwo/Desktop/ClientSource/" + arr[arr.length - 1];

                    String conn;

                    if (Arrays.asList(this.srcFiles).contains(checkFile)) {

                        conn = "Sorry Client is already having this file: " + checkFile;

                        System.out.println("Sending to Server: " + conn);

                        byte[] sendBytes = conn.getBytes();

                        DatagramPacket sendDatagramPacket = new DatagramPacket(sendBytes, conn.length(), id, 1234);

                        receiveSocket.send(sendDatagramPacket);

                        continue;

                    }

                    conn = "Client is ready to receive: " + msg;

                    System.out.println("Sending to Server: " + conn);

                    byte[] sendBytes = conn.getBytes();

                    DatagramPacket sendDatagramPacket = new DatagramPacket(sendBytes, conn.length(), id, 1234);

                    receiveSocket.send(sendDatagramPacket);

                    // Start Receiving file - Full Duplex

                    String newFile = "/home/vmtwo/Desktop/ClientDestination/" + arr[arr.length - 1];

                    OutputStream fos = new FileOutputStream(newFile);

                    int fileSize = Integer.parseInt(fileDetails[1]);

                    LocalTime startTime = LocalTime.now();

                    System.out.println("startReceiveTime: " + startTime);

                    loggerFile.info("Start Time: " + startTime + " - Started to receive the file: " + fileDetails[0]

                            + " from server to client of size: " + fileSize);

                    int counter = 0;

                    while (counter < fileSize + 1) {

                        boolean checkConnection = true;

                        int lastSeqNo = -1;

                        while (checkConnection) {

                            try {

                                byte[] receiveData = new byte[BUFFER\_SIZE + 120];

                                DatagramPacket received = new DatagramPacket(receiveData, receiveData.length);

                                receiveSocket.receive(received);

                                String msg1 = new String(received.getData(), 0, received.getLength());

                                if (msg1.equalsIgnoreCase("EOF")) {

                                    byte[] sending = "Thank You".getBytes();

                                    DatagramPacket sendPacket = new DatagramPacket(sending, sending.length, id, 1234);

                                    receiveSocket.send(sendPacket);

                                    checkConnection = false;

                                    counter = fileSize + 1;

                                    continue;

                                }

                                // Extract Data

                                byte[] fileNameLen = new byte[5];

                                byte[] seqNum = new byte[10];

                                System.arraycopy(receiveData, 0, fileNameLen, 0, fileNameLen.length);

                                int fileNmLen = Integer.parseInt(

                                        (new String(fileNameLen, 0, fileNameLen.length).replaceAll("\u0000.\*", "")));

                                byte[] fileName = new byte[fileNmLen];

                                System.arraycopy(receiveData, 5, fileName, 0, fileName.length);

                                System.arraycopy(receiveData, 105, seqNum, 0, seqNum.length);

                                String fileNm = (new String(fileName, 0, fileName.length).replaceAll("\u0000.\*", ""));

                                int seqNo = Integer

                                        .parseInt((new String(seqNum, 0, seqNum.length).replaceAll("\u0000.\*", "")));

                                if (lastSeqNo == seqNo) {

                                    String acknowledgement = "Already Received seq no " + seqNo;

                                    //System.out.println("Sending Acknowledgement: " + acknowledgement);

                                    byte[] sendData = acknowledgement.getBytes();

                                    DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 1234);

                                    receiveSocket.send(packet);

                                    continue;

                                } else {

                                    lastSeqNo = seqNo;

                                    checkConnection = false;

                                }

                                fos.write(receiveData, 120, receiveData.length - 120);

                                fos.flush();

                                // System.out.println(fileNm + " Sequence Number: " + seqNo + " received");

                                String acknowledgement;

                                if (counter < fileSize)

                                    acknowledgement = fileNm + " Waiting for Sequence Number: " + (seqNo + 1);

                                else

                                    acknowledgement = "Received whole file - " + fileNm;

                                // System.out.println("Sending Acknowledgement: " + acknowledgement);

                                byte[] sendData = acknowledgement.getBytes();

                                DatagramPacket packet = new DatagramPacket(sendData, sendData.length, id, 1234);

                                receiveSocket.send(packet);

                                counter = counter + receiveData.length - 120;

                            } catch (IOException ex) {

                                checkConnection = true;

                            }

                        }

                    }

                    LocalTime endTime = LocalTime.now();

                    System.out.println("endReceiveTime: " + endTime);

                    loggerFile.info("End Time: " + endTime + " - File: " + fileDetails[0]

                            + " is successfully received from server to client of size: " + fileSize);

                    fos.close();

                } catch (IOException ex) {

                    System.out.println(ex);

                }

            }

        }

    }

}

public class Assignment4UDPClientFTP {

    public static String[] fileNameSource = { "/home/vmtwo/Desktop/ClientSource/ClientFile.png",

            "/home/vmtwo/Desktop/ClientSource/ClientFile.pdf", "/home/vmtwo/Desktop/ClientSource/ClientFile.mp4",

            "/home/vmtwo/Desktop/ClientSource/ServerFile.pdf", "/home/vmtwo/Desktop/ClientSource/ClientFile.txt" };

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

        Assignment4UDPClientFTP obj = new Assignment4UDPClientFTP();

        Logger logger = obj.LoggingFiles();

        String[] priority = { ".txt", ".png", ".pdf", ".mp4" };

        String[] setpriority = new String[fileNameSource.length];

        int count = 0;

        for (int i = 0; i < priority.length; i++) {

            for (int j = 0; j < fileNameSource.length; j++) {

                if (fileNameSource[j].substring(fileNameSource[j].length() - 4).equals(priority[i])) {

                    setpriority[count++] = fileNameSource[j];

                }

            }

        }

        DatagramSocket sendSocket = new DatagramSocket(6789);

        DatagramSocket receiveSocket = new DatagramSocket(9876);

        sendSocket.setSoTimeout(2000);

        ClientThread1 sct = new ClientThread1(sendSocket, receiveSocket, "send", setpriority, logger);

        sct.start();

        ClientThread1 rct = new ClientThread1(sendSocket, receiveSocket, "receive", setpriority, logger);

        rct.start();

    }

    public Logger LoggingFiles() {

        Logger logger = Logger.getLogger("ClientLog");

        FileHandler fh;

        try {

            // This block configure the logger with handler and formatter

            fh = new FileHandler("/home/vmtwo/Desktop/ClientDestination/ClientLogFile.log");

            logger.addHandler(fh);

            SimpleFormatter formatter = new SimpleFormatter();

            fh.setFormatter(formatter);

            // the following statement is used to log any messages

            logger.setUseParentHandlers(false);

            logger.info("Uncommon Files Information");

        } catch (SecurityException e) {

            e.printStackTrace();

        } catch (IOException e) {

            e.printStackTrace();

        }

        return logger;

    }

}

**OUTPUT**

Server:

$ javac Assignment4UDPServerFTP.java

$ java Assignment4UDPServerFTP

….

Sending a file to client: /home/vmone/Desktop/ServerSource/ServerFile.txt 11064

Timeout in receiving the client's status

Sending a file to client: /home/vmone/Desktop/ServerSource/ServerFile.txt 11064

Client: /home/vmtwo/Desktop/ClientSource/ClientFile.txt 11064

Sending to Client: Server is ready to receive: /home/vmtwo/Desktop/ClientSource/ClientFile.txt 11064

startReceiveTime: 12:50:57.499245

endReceiveTime: 12:50:57.697599

Client: EOF

Client: /home/vmtwo/Desktop/ClientSource/ClientFile.png 709290

Sending to Client: Server is ready to receive: /home/vmtwo/Desktop/ClientSource/ClientFile.png 709290

startReceiveTime: 12:50:57.735139

Timeout in receiving the client's status

Sending a file to client: /home/vmone/Desktop/ServerSource/ServerFile.txt 11064

Client: Client is ready to receive: /home/vmone/Desktop/ServerSource/ServerFile.txt 11064

startTime: 12:50:59.203195

Client: Thank You

endTime: 12:50:59.435831

Sending a file to client: /home/vmone/Desktop/ServerSource/ServerFile.png 709290

Client: Client is ready to receive: /home/vmone/Desktop/ServerSource/ServerFile.png 709290

startTime: 12:50:59.456458

endReceiveTime: 12:51:09.333845

Client: EOF

Client: /home/vmtwo/Desktop/ClientSource/ClientFile.pdf 32452164

Sending to Client: Server is ready to receive: /home/vmtwo/Desktop/ClientSource/ClientFile.pdf 32452164

startReceiveTime: 12:51:10.128784

Client: Thank You

endTime: 12:51:11.250268

And so on….

Client:

$ javac Assignment4UDPClientFTP.java

$ java Assignment4UDPClientFTP

…

Packet of file /home/vmtwo/Desktop/ClientSource/ClientFile.pdf and of sequence number 13879 is lost

Network is not reachable

Packet of file /home/vmtwo/Desktop/ClientSource/ClientFile.pdf and of sequence number 13879 is lost

Network is not reachable

Packet of file /home/vmtwo/Desktop/ClientSource/ClientFile.pdf and of sequence number 13879 is lost

Network is not reachable

Packet of file /home/vmtwo/Desktop/ClientSource/ClientFile.pdf and of sequence number 13879 is lost

Server: Thank You

endTime: 01:30:10.888647

Sending a file to server: /home/vmtwo/Desktop/ClientSource/ServerFile.pdf 17015975

Server: Sorry Server is already having this file: /home/vmone/Desktop/ServerSource/ServerFile.pdf

Sending a file to server: /home/vmtwo/Desktop/ClientSource/ClientFile.mp4 78159792

Server: Sorry Server is already having this file: /home/vmone/Desktop/ServerSource/ClientFile.mp4

Server: Thank You

endReceiveTime: 01:39:16.209546

Client: /home/vmone/Desktop/ServerSource/ClientFile.mp4 78159792

Sending to Server: Sorry Client is already having this file: /home/vmtwo/Desktop/ClientSource/ClientFile.mp4

Server: Bye

**RESULT**

Thus both the client and server exchange multimodal files using UDP socket programming.















