**CRC:1**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class TCPCRCClient1 {

private static final String CRC\_POLYNOMIAL = "100000111";

public static void main(String[] args) {

String serverAddress = "127.0.0.1";

int serverPort = 54321;

Scanner userInputScanner = new Scanner(System.in);

try (Socket clientSocket = new Socket(serverAddress, serverPort);

BufferedReader serverReader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter serverWriter = new PrintWriter(clientSocket.getOutputStream(), true)

) {

System.out.println("Server: " + serverReader.readLine());

System.out.print("Enter binary message: ");

String userMessage = userInputScanner.nextLine();

String messagePadded = userMessage + "00000000";

String computedCRC = computeCRC(messagePadded);

String fullMessage = userMessage + computedCRC;

System.out.println("Sending encoded message: " + fullMessage);

serverWriter.println(fullMessage);

System.out.println("Server: " + serverReader.readLine());

} catch (IOException ex) {

System.out.println("Connection error: " + ex.getMessage());

}

userInputScanner.close();

}

private static String computeCRC(String inputBits) {

char[] bitArray = inputBits.toCharArray();

int totalLength = bitArray.length;

int polynomialLength = CRC\_POLYNOMIAL.length();

for (int i = 0; i <= totalLength - polynomialLength; i++) {

if (bitArray[i] == '1') {

for (int j = 0; j < polynomialLength; j++) {

bitArray[i + j] = (bitArray[i + j] == CRC\_POLYNOMIAL.charAt(j)) ? '0' : '1';

}

}

}

int crcIndexStart = totalLength - polynomialLength + 1;

StringBuilder crcResult = new StringBuilder();

for (int i = crcIndexStart; i < totalLength; i++) {

crcResult.append(bitArray[i]);

}

return crcResult.toString();

}

}

**CRC:2**

import java.io.\*;

import java.net.\*;

public class TCPCRCServer1 {

private static final String CRC\_GENERATOR = "100000111";

public static void main(String[] args) {

int serverPort = 54321;

try (ServerSocket serverSocket = new ServerSocket(serverPort)) {

System.out.println("Server started on port " + serverPort);

Socket clientSocket = serverSocket.accept();

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

BufferedReader clientInput = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter clientOutput = new PrintWriter(clientSocket.getOutputStream(), true);

clientOutput.println("Connected to CRC Server!");

String receivedCodeword = clientInput.readLine();

System.out.println("Received codeword: " + receivedCodeword);

boolean isValid = verifyCRC(receivedCodeword);

if (isValid) {

clientOutput.println("No errors found in transmission.");

} else {

clientOutput.println("Error detected in transmission.");

}

clientSocket.close();

} catch (IOException e) {

System.out.println("Server error: " + e.getMessage());

}

}

private static boolean verifyCRC(String codeword) {

char[] bits = codeword.toCharArray();

int generatorLength = CRC\_GENERATOR.length();

int totalBits = bits.length;

for (int i = 0; i <= totalBits - generatorLength; i++) {

if (bits[i] == '1') {

for (int j = 0; j < generatorLength; j++) {

bits[i + j] = (bits[i + j] == CRC\_GENERATOR.charAt(j)) ? '0' : '1';

}

}

}

for (int i = totalBits - generatorLength + 1; i < totalBits; i++) {

if (bits[i] == '1') {

return false;

}

}

return true; // No errors

}

}

**Bit:1**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class TCPBitStuffingClient {

public static void main(String[] args) {

String serverAddress = "127.0.0.1";

int serverPort = 54321;

try (

Socket clientSocket = new Socket(serverAddress, serverPort);

PrintWriter toServer = new PrintWriter(clientSocket.getOutputStream(), true);

BufferedReader fromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

Scanner userInput = new Scanner(System.in)

) {

System.out.print("Enter bit sequence: ");

String inputBits = userInput.nextLine();

String stuffedData = applyBitStuffing(inputBits);

System.out.println("Stuffed Data Sent: " + stuffedData);

toServer.println(stuffedData); // Send stuffed data to server

String serverResponse = fromServer.readLine(); // Get server reply

System.out.println("Server Response (Destuffed): " + serverResponse);

} catch (IOException e) {

System.out.println("Connection error: " + e.getMessage());

}

}

private static String applyBitStuffing(String data) {

StringBuilder stuffed = new StringBuilder("01111110"); // Start flag

int oneCount = 0;

for (char bit : data.toCharArray()) {

stuffed.append(bit);

if (bit == '1') {

oneCount++;

if (oneCount == 5) {

stuffed.append('0'); // Stuff '0' after five '1's

oneCount = 0;

}

} else {

oneCount = 0;

}

}

stuffed.append("01111110"); // End flag

return stuffed.toString();

}

}

**Bit 2:**

import java.io.\*;

import java.net.\*;

public class TCPBitStuffingServer {

public static void main(String[] args) {

int serverPort = 54321;

try (ServerSocket serverSocket = new ServerSocket(serverPort)) {

System.out.println("Server is running on port " + serverPort);

while (true) {

try (

Socket clientSocket = serverSocket.accept();

BufferedReader fromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter toClient = new PrintWriter(clientSocket.getOutputStream(), true)

) {

String receivedData = fromClient.readLine();

System.out.println("Received Stuffed Data: " + receivedData);

String originalData = removeBitStuffing(receivedData);

toClient.println(originalData); // Send destuffed data back

System.out.println("Destuffed Data Sent: " + originalData);

}

}

} catch (IOException e) {

System.out.println("Server error: " + e.getMessage());

}

}

private static String removeBitStuffing(String stuffedData) {

StringBuilder destuffed = new StringBuilder();

int oneCount = 0;

for (int i = 8; i < stuffedData.length() - 8; i++) {

char bit = stuffedData.charAt(i);

destuffed.append(bit);

if (bit == '1') {

oneCount++;

if (oneCount == 5) {

i++; // Skip the stuffed '0'

oneCount = 0;

}

} else {

oneCount = 0;

}

}

return destuffed.toString();

}

}

**Byte:1**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class TCPByteStuffingSender {

private static final String FLAG\_BYTE = "01111110";

private static final String ESCAPE\_BYTE = "01111101";

public static void main(String[] args) {

String serverAddress = "127.0.0.1";

int serverPort = 6000;

try (

Socket clientSocket = new Socket(serverAddress, serverPort);

PrintWriter toServer = new PrintWriter(clientSocket.getOutputStream(), true);

Scanner userInput = new Scanner(System.in)

) {

System.out.print("Enter binary data (8-bit bytes): ");

String inputBits = userInput.nextLine();

String stuffedMessage = applyByteStuffing(inputBits);

System.out.println("Stuffed Data Sent: " + stuffedMessage);

toServer.println(stuffedMessage); // Send to server

} catch (IOException e) {

System.out.println("Connection error: " + e.getMessage());

}

}

public static String applyByteStuffing(String data) {

StringBuilder stuffed = new StringBuilder(FLAG\_BYTE); // Start with flag

for (int i = 0; i < data.length(); i += 8) {

String byteChunk = data.substring(i, Math.min(i + 8, data.length()));

if (byteChunk.equals(FLAG\_BYTE) || byteChunk.equals(ESCAPE\_BYTE)) {

stuffed.append(ESCAPE\_BYTE);

}

stuffed.append(byteChunk);

}

stuffed.append(FLAG\_BYTE); // End with flag

return stuffed.toString();

}

}

**Byte:2**

import java.io.\*;

import java.net.\*;

public class TCPByteStuffingReceiver {

private static final String FLAG\_BYTE = "01111110";

private static final String ESCAPE\_BYTE = "01111101";

public static void main(String[] args) {

int serverPort = 6000;

try (ServerSocket serverSocket = new ServerSocket(serverPort)) {

System.out.println("Server is running on port " + serverPort);

System.out.println("Waiting for client connection...");

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected!");

BufferedReader fromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

String receivedStuffedData = fromClient.readLine();

System.out.println("Received Stuffed Data: " + receivedStuffedData);

String originalData = removeByteStuffing(receivedStuffedData);

System.out.println("Unstuffed Data: " + originalData);

clientSocket.close();

} catch (IOException e) {

System.out.println("Server error: " + e.getMessage());

}

}

public static String removeByteStuffing(String stuffedData) {

// Remove starting and ending FLAG\_BYTE

String cleanData = stuffedData.substring(FLAG\_BYTE.length(), stuffedData.length() - FLAG\_BYTE.length());

StringBuilder unstuffedData = new StringBuilder();

for (int i = 0; i < cleanData.length(); i += 8) {

String currentByte = cleanData.substring(i, Math.min(i + 8, cleanData.length()));

if (currentByte.equals(ESCAPE\_BYTE)) {

i += 8; // Skip escape byte

currentByte = cleanData.substring(i, Math.min(i + 8, cleanData.length()));

}

unstuffedData.append(currentByte);

}

return unstuffedData.toString();

}

}

**TCP:1**

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class Client {

public static void main(String[] args) {

try {

Socket s = new Socket("localhost", 5555);

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

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

String str = "hello world";

dout.writeUTF(str);

dout.flush();

s.close();

System.out.println("Message sent to server: " + str);

} catch (IOException e) {

e.printStackTrace();

}

}

}

**TCP 2:**

import java.io.DataInputStream;

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

public class Server {

public static void main(String[] args) {

try {

ServerSocket ss = new ServerSocket(5555);

System.out.println("Server is running and waiting for a client...");

Socket s = ss.accept();

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

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

String message = din.readUTF();

System.out.println("Received from client: " + message);

s.close();

ss.close();

System.out.println("Connection closed.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**UDP:1**

import java.net.DatagramPacket;

import java.net.DatagramSocket;

public class UDPServer {

public static void main(String[] args) {

try {

DatagramSocket serverSocket = new DatagramSocket(5555);

System.out.println("UDP Server is running and waiting for a message...");

byte[] receiveBuffer = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);

serverSocket.receive(receivePacket);

String message = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("Received from client: " + message);

// Close the socket

serverSocket.close();

System.out.println("UDP Server socket closed.");

} catch (Exception e) {

e.printStackTrace();

}

}

}

**UDP:2**

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

public class UDPClient {

public static void main(String[] args) {

try {

// Create a UDP socket

DatagramSocket clientSocket = new DatagramSocket();

// Message to send

String message = "hello world via UDP";

byte[] sendBuffer = message.getBytes();

// Address and port of the server

InetAddress serverAddress = InetAddress.getByName("localhost");

int serverPort = 5555;

// Create packet and send it

DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, serverAddress, serverPort);

clientSocket.send(sendPacket);

System.out.println("Message sent to server: " + message);

// Close the socket

clientSocket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**LeakyBucket:**

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.Scanner;

public class LeakyBucket {

private static final int BucketSize = 500;

private static final int OutputRate = 200;

private static int bucket = 0;

private static String getCurrentTime() {

DateTimeFormatter f = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

return LocalDateTime.now().format(f);

}

private static void addPacket(int size) {

int avsp = BucketSize - bucket;

int rsp = size - avsp;

if (size > avsp) {

System.out.println(getCurrentTime() + " Bucket overflow! Can add only " + avsp + " bytes. Dropped " + rsp + " bytes.");

bucket = BucketSize;

} else {

bucket += size;

System.out.println(getCurrentTime() + " " + size + " bytes added to bucket");

}

processPackets();

}

private static void processPackets() {

if (bucket > 0) {

int sent = Math.min(bucket, OutputRate);

bucket -= sent;

System.out.println(getCurrentTime() + " - " + sent + " bytes outputted");

System.out.println(getCurrentTime() + " Remaining in bucket: " + bucket + " bytes");

} else {

System.out.println(getCurrentTime() + " Bucket empty. Nothing to output.");

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (true) {

System.out.println("\nEnter packet size (or 0 to exit):");

int packetSize = sc.nextInt();

if (packetSize == 0) {

break;

}

addPacket(packetSize);

}

while (bucket > 0) {

processPackets();

}

sc.close();

}

}