Computer Networks Lab Assessment 2

# Suryakumar P 21MIS1146

1. Client is sending a message to the server. The server encodes the message and returns to the client. (Encoding is done by replacing the character by the ASCII value of the remainder using the formula (ASCII (chr) mod (nth prime)) n-value is sent by the client) Write the program to implement the above.

**Problem Definition:**

To write a Java Program such that client is sending a message to the server. The server encodes the message and returns to the client. (Encoding is done by replacing the character by the ASCII value of the remainder using the formula (ASCII (chr) mod (nth prime)) n-value is sent by the client).

**Method:**

1. The server code listens for incoming connections on port 8080.
2. When a client connects to the server, the server accepts the connection and displays the client's IP address.
3. The server receives the n-value and the message from the client using the DataInputStream class.
4. The server calculates the nth prime number using the generateNthPrime() function.
5. The server encodes the message by iterating over each character, converting it to its ASCII value, and taking the remainder of the ASCII value divided by the nth prime.
6. The encoded message is sent back to the client using the DataOutputStream class.
7. The client code establishes a connection with the server on localhost (IP address 127.0.0.1) and port 8080.
8. The client sends the n-value and the message to the server using the DataOutputStream class.
9. The client receives the encoded message from the server using the DataInputStream class.
10. The client prints the encoded message received from the server.
11. The client closes the socket connection.
12. The server waits for the next client to connect and repeats the process.

**Code:**

*//Server.java*

import java.io.\*;

import java.net.\*;

import java.util.ArrayList;

import java.util.List;

public class Server {

*// Function to generate the nth prime number*

    private static int generateNthPrime(int n) {

        List<Integer> primes = new ArrayList<>();

        primes.add(2);

        int num = 3;

        while (primes.size() < n) {

            boolean isPrime = true;

            for (int prime : primes) {

                if (num % prime == 0) {

                    isPrime = false;

                    break;

                }

            }

            if (isPrime) {

                primes.add(num);

            }

            num += 2;

        }

        return primes.get(n - 1);

    }

*// Function to encode the message*

    private static String encodeMessage(String message, int n) {

        StringBuilder encodedMessage = new StringBuilder();

        int prime = generateNthPrime(n);

        for (char c : message.toCharArray()) {

            int ascii = (int) c;

            int encodedAscii = ascii % prime;

            encodedMessage.append(encodedAscii).append(" ");

        }

        return encodedMessage.toString().trim();

    }

    public static void main(String[] args) {

        try {

            ServerSocket serverSocket = new ServerSocket(8080);

            System.out.println("Server listening on port 8080...");

            while (true) {

                Socket clientSocket = serverSocket.accept();

                System.out.println("Client connected: " + clientSocket.getInetAddress());

*// Receive the n-value and the message from the client*

                DataInputStream inputStream = new DataInputStream(clientSocket.getInputStream());

                int n = inputStream.readInt();

                String message = inputStream.readUTF();

*// Encode the message*

                String encodedMessage = encodeMessage(message, n);

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

                System.out.println("Encoded Message: " + encodedMessage);

*// Send the encoded message back to the client*

                DataOutputStream outputStream = new DataOutputStream(clientSocket.getOutputStream());

                outputStream.writeUTF(encodedMessage);

                clientSocket.close();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

*//Client.java*

import java.io.\*;

import java.net.\*;

public class Client {

    public static void main(String[] args) {

        try {

            Socket socket = new Socket("localhost", 8080);

            DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream());

*// Send the n-value and the message to the server*

            int n = 3;

            String message = "Suryakumar P 21MIS1146";

            outputStream.writeInt(n);

            outputStream.writeUTF(message);

*// Receive the encoded message from the server*

            DataInputStream inputStream = new DataInputStream(socket.getInputStream());

            String encodedMessage = inputStream.readUTF();

            System.out.println("Encoded Message: " + encodedMessage);

            socket.close();

        } catch (IOException e) {

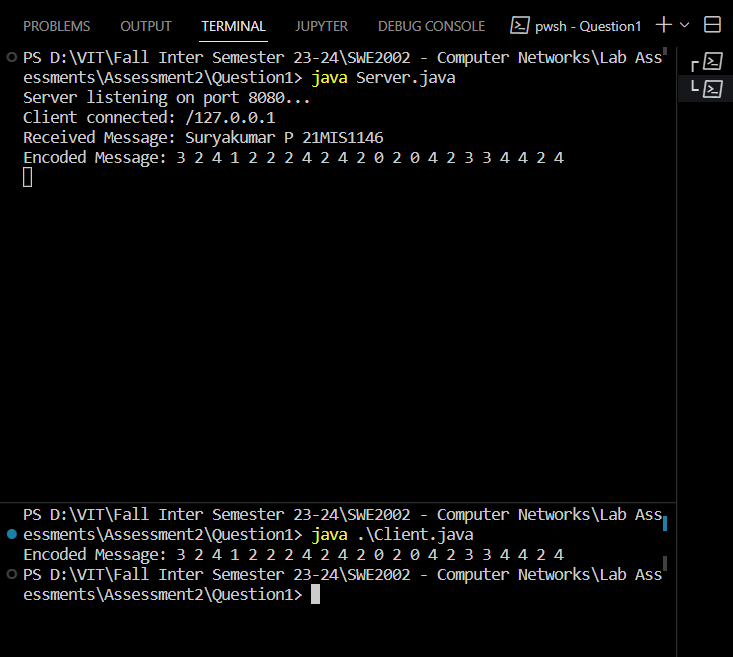
            e.printStackTrace();

        }

    }

}

**Output:**



2. Implement a TCP/IP socket-based ATM system. Server – to maintain the customer details (Name , Card no, Pin, Balance). Client – when a customer wants to withdraw an amount, validate his login with pin and balance.

**Problem Definition:**

To implement a TCP/IP socket-based ATM system. Server – to maintain the customer details (Name, Card no, Pin, Balance). Client – when a customer wants to withdraw an amount, validate his login with pin and balance through Java Program.

**Method:**

**Server-side:**

1. The server program listens on port 8080 for incoming client connections.
2. When a client connects to the server, a new thread is created to handle the client's request.
3. The client handler thread reads the card number and PIN from the client.
4. The server checks if the provided card number exists in the customer database and validates the PIN.
5. If the card number and PIN are valid, the server sends a "SUCCESS" response to the client.
6. The client handler thread reads the withdrawal amount from the client.
7. The server checks if the withdrawal amount is within the available balance for the customer.
8. If the balance is sufficient, the server updates the customer's balance, sends a "SUCCESS" response to the client, and also sends the updated balance, customer name, and account number to the client.
9. If the balance is insufficient, the server sends an "INSUFFICIENT\_BALANCE" response to the client.
10. If the card number or PIN is invalid, the server sends an "INVALID\_CREDENTIALS" response to the client.
11. The client handler thread closes the connection with the client.

**Client-side:**

1. The client program creates a socket connection to the server on localhost (IP address 127.0.0.1) and port 8080.
2. The client program obtains the output stream and input stream of the socket for communication with the server.
3. The client program sends the card number and PIN to the server.
4. The client program receives the login result from the server.
5. If the login is successful ("SUCCESS" response received), the client program sends the withdrawal amount to the server.
6. The client program receives the withdrawal result from the server.
7. If the withdrawal is successful ("SUCCESS" response received), the client program receives the updated balance, customer name, and account number from the server.
8. The client program displays the withdrawal success message along with the customer name, account number, and updated balance.
9. If the withdrawal fails due to insufficient balance ("INSUFFICIENT\_BALANCE" response received), the client program displays an insufficient balance message.
10. If the login fails due to invalid credentials ("INVALID\_CREDENTIALS" response received), the client program displays an invalid credentials message.
11. The client program closes the socket connection.

**Code:**

*//Server.java*

import java.io.\*;

import java.net.\*;

import java.util.HashMap;

import java.util.Map;

public class Server {

    private static Map<String, Customer> customers;

    static {

*// Initialize some customer details*

        customers = new HashMap<>();

        customers.put("1234567890123456", new Customer("Suryakumar", "1234", 5000000, "AC-1234"));

        customers.put("9876543210987654", new Customer("Tony Stark", "5678", 10000, "AC-5678"));

    }

    public static void main(String[] args) {

        try {

            ServerSocket serverSocket = new ServerSocket(8080);

            System.out.println("Server listening on port 8080...");

            while (true) {

                Socket clientSocket = serverSocket.accept();

                System.out.println("Client connected: " + clientSocket.getInetAddress());

*// Handle client request in a separate thread*

                Thread thread = new Thread(new ClientHandler(clientSocket));

                thread.start();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private static class ClientHandler implements Runnable {

        private Socket clientSocket;

        public ClientHandler(Socket clientSocket) {

            this.clientSocket = clientSocket;

        }

        @Override

        public void run() {

            try {

                DataInputStream inputStream = new DataInputStream(clientSocket.getInputStream());

                DataOutputStream outputStream = new DataOutputStream(clientSocket.getOutputStream());

*// Read card number and PIN from the client*

                String cardNumber = inputStream.readUTF();

                String pin = inputStream.readUTF();

*// Check if customer exists and validate the PIN*

                if (customers.containsKey(cardNumber) && customers.get(cardNumber).validatePin(pin)) {

                    Customer customer = customers.get(cardNumber);

                    outputStream.writeUTF("SUCCESS"); *// PIN validation successful*

*// Read the withdrawal amount from the client*

                    int withdrawalAmount = inputStream.readInt();

*// Check if the withdrawal amount is within the available balance*

                    if (withdrawalAmount <= customer.getBalance()) {

                        customer.withdraw(withdrawalAmount); *// Update the customer's balance*

                        outputStream.writeUTF("SUCCESS"); *// Withdrawal successful*

                        outputStream.writeInt(customer.getBalance()); *// Send updated balance to the client*

                        outputStream.writeUTF(customer.getName()); *// Send customer name to the client*

                        outputStream.writeUTF(customer.getAccountNumber()); *// Send customer account number to the client*

                    } else {

                        outputStream.writeUTF("INSUFFICIENT\_BALANCE"); *// Insufficient balance*

                    }

                } else {

                    outputStream.writeUTF("INVALID\_CREDENTIALS"); *// Invalid card number or PIN*

                }

                clientSocket.close();

            } catch (IOException e) {

                e.printStackTrace();

            }

        }

    }

    private static class Customer {

        private String name;

        private String pin;

        private int balance;

        private String accountNumber;

        public Customer(String name, String pin, int balance, String accountNumber) {

            this.name = name;

            this.pin = pin;

            this.balance = balance;

            this.accountNumber = accountNumber;

        }

        public boolean validatePin(String pin) {

            return this.pin.equals(pin);

        }

        public void withdraw(int amount) {

            this.balance -= amount;

        }

        public int getBalance() {

            return balance;

        }

        public String getName() {

            return name;

        }

        public String getAccountNumber() {

            return accountNumber;

        }

    }

}

*//Cliet.java*

import java.io.\*;

import java.net.\*;

public class Client {

    public static void main(String[] args) {

        try {

            Socket socket = new Socket("localhost", 8080);

            DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream());

            DataInputStream inputStream = new DataInputStream(socket.getInputStream());

*// Card number and PIN for login*

            String loginCardNumber = "1234567890123456";

            String pin = "1234";

*// Withdrawal amount*

            int withdrawalAmount = 2000;

*// Send card number and PIN to the server*

            outputStream.writeUTF(loginCardNumber);

            outputStream.writeUTF(pin);

*// Receive the login result from the server*

            String loginResult = inputStream.readUTF();

            if (loginResult.equals("SUCCESS")) {

*// Send the withdrawal amount to the server*

                outputStream.writeInt(withdrawalAmount);

*// Receive the withdrawal result from the server*

                String withdrawalResult = inputStream.readUTF();

                if (withdrawalResult.equals("SUCCESS")) {

*// Receive the updated balance from the server*

                    int updatedBalance = inputStream.readInt();

*// Receive the customer name from the server*

                    String customerName = inputStream.readUTF();

*// Receive the customer account number from the server*

                    String accountNumber = inputStream.readUTF();

                    System.out.println("Withdrawal successful!");

                    System.out.println("Customer: " + customerName);

                    System.out.println("Account Number: " + accountNumber);

                    System.out.println("Updated Balance: INR " + updatedBalance);

                } else if (withdrawalResult.equals("INSUFFICIENT\_BALANCE")) {

                    System.out.println("Insufficient balance for withdrawal!");

                } else {

                    System.out.println("Withdrawal failed!");

                }

            } else if (loginResult.equals("INVALID\_CREDENTIALS")) {

                System.out.println("Invalid card number or PIN!");

            } else {

                System.out.println("Login failed!");

            }

            socket.close();

        } catch (IOException e) {

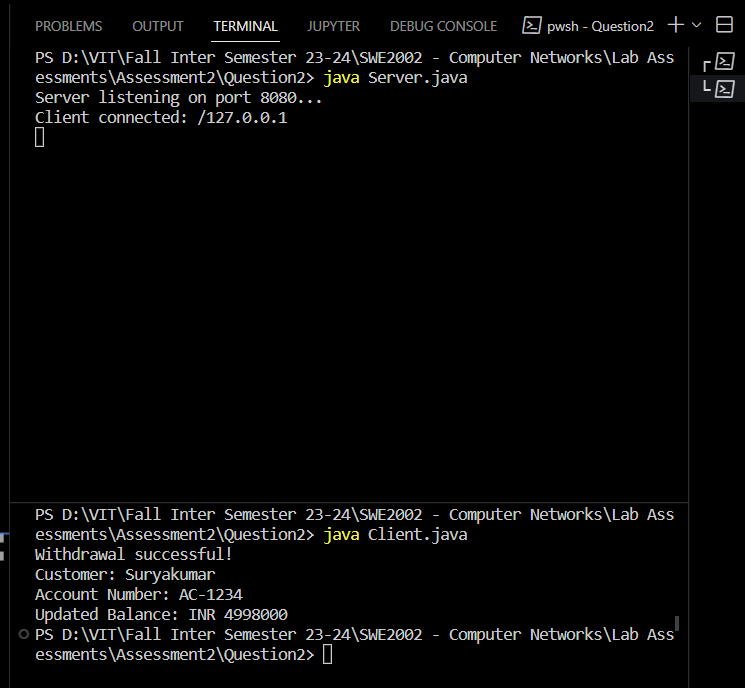
            e.printStackTrace();

        }

    }

}

**Output:**



3. In an IPV4 packet the value of header length is 1000 in binary. Write a code to find how many bytes of options are being carried by this packet.

**Problem Definition:**

To write a code to implement in an IPV4 packet the value of header length is 1000 in binary that finds how many bytes of options are being carried by that packet.

**Method:**

1. The code begins by defining a class named **IPv4OptionsLengthCalculator**.
2. Inside the class, the **main** method is defined, which serves as the entry point for the program.
3. The **headerLengthBinary** variable is declared and assigned a value of "1000", representing the header length field of the IPv4 packet in binary.
4. The **Integer.parseInt(headerLengthBinary, 2)** statement converts the binary string **headerLengthBinary** to its decimal representation. This step is necessary to perform calculations based on the decimal value of the header length.
5. The result of the conversion is stored in the **headerLengthDecimal** variable.
6. The code then calculates the number of bytes of options by multiplying the **headerLengthDecimal** value by 4 (each value in the header length field represents a 4-byte word) and subtracting 20 (the fixed size of the IPv4 header without options). The result is stored in the **optionsLength** variable.
7. Finally, the code prints the result using **System.out.println**, displaying the number of bytes of options carried by the IPv4 packet.

**Code:**

*//Main Program - Suryakumar 21MIS1146*

public class IPv4OptionsLengthCalculator {

    public static void main(String[] args) {

*// Header length value in binary*

        String headerLengthBinary = "1000";

*// Convert binary to decimal*

        int headerLengthDecimal = Integer.parseInt(headerLengthBinary, 2);

*// Calculate number of bytes of options*

        int optionsLength = (headerLengthDecimal \* 4) - 20;

        System.out.println("Number of bytes of options: " + optionsLength);

    }

}

**Output:**

