University of Dhaka

Department of Computer Science and Engineering

CSE 3111 – Computer Networking Laboratory Credits: 1.5 Batch: 27/3rd Year 1st Sem 2023

Instructors: Prof. Dr. Md. Abdur Razzaque (AR), Prof. Dr. Md. Mamun-Or-Rashid (MOR), Dr. Muhammad Ibrahim (MIb) and Mr. Md. Redwan Ahmed Rizvee (RAR)

Lab Experiment # 2

Name of the Experiment:

Introduction to Client-Server Communication using Socket Programming — Simulating an ATM Machine Operation

Creating TCP Connections using Socket Programming

a) Establish a TCP connection in between a server process, running on host A and a client process, running on host B and then perform some operation by the server process requested by the client and send responses from the server.
   i. Capital letter to Small letter conversion for a line of text
   ii. Send an integer and operation name (either 'prime' or 'palindrome') to the server and check whether it's a prime (or palindrome) or not

b) Using the above connection, design and implement a non-idempotent operation using exactly-once semantics that can handle the failure of request messages, failure of response messages and process execution failures.
   i. Design and describe an application-level protocol to be used between an automatic teller machine and a bank's centralized server. Your protocol should allow a user's card and password to be verified, the account balance (which is maintained at the centralized computer) to be queried, and an account withdrawal to be made (that is, money disbursed to the user). Your protocol entities should be able to handle the all-too-common cases in which there is not enough money in the account to cover the withdrawal. Specify your protocol by listing the messages exchanged and the action taken by the automatic teller machine or the bank's centralized computer on transmission and receipt of messages. Sketch the operation of your protocol for the case of a simple withdrawal with no errors.
   ii. HOME WORK – Enhance the above protocol so that it can handle errors related to both request and response messages to and from the server.

Resource Link: https://www.geeksforgeeks.org/socket-programming-in-java/

What is an idempotent operation?

In computing, an idempotent operation is one that has no additional effect if it is called more than once with the same input parameters. For example, removing an item from a set can be considered an idempotent operation on the set.

What is an exactly-once semantics?

As its name suggests, exactly-once semantics means that each message is delivered precisely once. The message can neither be lost nor delivered twice (or more times). Exactly-once is by far the most dependable message delivery guarantee.

// A Java program for a Client

```
import java.net.*;
import java.io.*;

public class Client
{
	// initialize socket and input output streams
	private Socket socket		= null;
	private DataInputStream input = null;
	private DataOutputStream out = null;

	// constructor to put ip address and port
	public Client(String address, int port)
	{
		// establish a connection
		try
		{
			socket = new Socket (address, port);
			System.out.println("Connected");

			// takes input from terminal
			input = new DataInputStream(System.in);

			// sends output to the socket
			out = new DataOutputStream(socket.getOutputStream());
		}
		catch(UnknownHostException u)
		{
			System.out.println(u);
		}
		catch(IOException i)
		{
			System.out.println(i);
		}

		// string to read message from input
		String line = "";

		// keep reading until "Over" is input
		while (!line.equals("Over"))
```

```
		{
			try
			{
				line = input.readLine();
				out.writeUTF(line);
			}
			catch(IOException i)
			{
				System.out.println(i);
			}
		}

		// close the connection
		try
		{
			input.close();
			out.close();
			socket.close();
		}
		catch(IOException i)
		{
			System.out.println(i);
		}
	}

	public static void main(String args[])
	{
		Client client = new Client("IP Address of Server machine", 5000);
	}
}
```

// A Java program for a Server

```
import java.net.*;
import java.io.*;

public class Server
{
	//initialize socket and input stream
	private Socket		 socket = null;
	private ServerSocket server = null;
	private DataInputStream in	 = null;

	// constructor with port
	public Server(int port)
	{
```

```
		// starts server and waits for a connection
		try
		{
			server = new ServerSocket(port);
			System.out.println("Server started");

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

			socket = server.accept();
			System.out.println("Client accepted");

			// takes input from the client socket
			in = new DataInputStream(
				new BufferedInputStream(socket.getInputStream()));

			String line = "";

			// reads message from client until "Over" is sent
			while (!line.equals("Over"))
			{
				try
				{
					line = in.readUTF();
					System.out.println(line);

				}
				catch(IOException i)
				{
					System.out.println(i);
				}
			}
			System.out.println("Closing connection");

			// close connection
			socket.close();
			in.close();
		}
		catch(IOException i)
		{
			System.out.println(i);
		}
	}

	public static void main(String args[])
	{
		Server server = new Server(5000);
	}
}
```