Lab 1. Introduction to Network Programming

Familiarity with the Lab environment and Client-Server model, Unix basic commands.

Objective

The primary objective of this lab session is to develop familiarity with the lab environment understand the Client-Server model, and practice basic Unix commands.

Theory

Network programming has become an essential skill for developers, as most modern applications involve some level of network interaction. Applications like email clients, web browsers, IDEs, word processors, antivirus programs, music players, multiplayer games, and even point-of-sale systems, all incorporate networking features to enhance functionality.

Examples of Networked Applications:

- **Text Editors**: Save and open files directly from FTP servers (e.g., BBEdit).
- **IDEs**: Communicate with source code repositories (e.g., Eclipse, IntelliJ IDEA).
- **Word Processors**: Open files from URLs (e.g., Microsoft Word).
- **Antivirus Programs**: Update virus definitions from the internet (e.g., Norton AntiVirus).
- **Music Players**: Communicate with online databases to fetch track details (e.g., Winamp, iTunes).
- **Games**: Enable multiplayer functionality in real time (e.g., Halo).
- **Point-of-Sale Systems**: Communicate with servers for transaction processing (e.g., IBM SurePOS ACE).
- **Scheduling Applications**: Synchronize calendars over a network (e.g., Microsoft Outlook).

In network programming, understanding the lab environment is crucial. This environment simulates real-world scenarios where multiple devices (computers, servers, routers) communicate over a network.

Lab Environment Components

1. **Operating System**
   Unix/Linux-based systems (e.g., Ubuntu, CentOS) are preferred for network programming due to their robust networking tools and stability. They provide native support for most networking utilities, making it easier to implement and test network applications.

2. **Networking Utilities**
    - **Terminal**: The primary interface for interacting with the operating system. It allows executing commands, writing scripts, and running network programs.
    - **Text Editors**: Tools like `vim`, `nano`, or `gedit` for writing and editing code. IDEs like Eclipse or IntelliJ IDEA may also be used for more complex projects.
    - **Compilers and Interpreters**: Tools like `gcc` for C/C++, `javac` for Java, and `python3` for Python are used to compile or interpret code.

3. **Network Configuration and Testing Tools**:
    - **`ifconfig` / `ip`**: Used to configure and display network interface parameters, such as IP addresses and network masks.
    - **`ping`**: Tests the reachability of a host on a network and checks basic connectivity.
    - **`netstat`**: Displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.
    - **`traceroute`**: Tracks the path that a packet takes from the source to the destination, helping to diagnose routing issues.

4. **Packet Analyzers**:
   Tools like Wireshark allow capturing and analyzing network traffic, which is useful for debugging network communication.

Client-Server Model in Network Programming

The Client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and delivers the data packets requested back to the client. Clients do not share any of their resources. Examples of the Client-Server Model are Email, World Wide Web, etc.

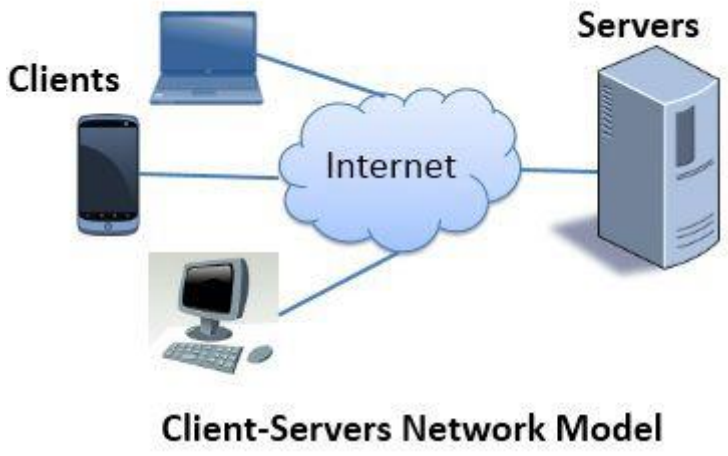


Figure: Client-Server Model in Network Programming

Unix Basic Commands in Network Programming

In network programming on Unix-like operating systems, a variety of commands are used to perform tasks such as network configuration, troubleshooting, data transfer, and monitoring. Below are some of the commonly used Unix commands in network programming:. Some of the Unix Basic Commands in Network Programming are as follows:

- ifconfig/ip
  It is Used to view or configure network interface settings. (deprecated in favor of ip command on most Linux distributions).For example: ifconfig displays all network interfaces.

- ping
  Test connectivity between your machine and a remote host using ICMP (Internet Control Message Protocol).For Example: ping youtube.com checks whether the Youtube server is reachable from your machine.

- netstat
  netstat displays network connections, routing tables, interface statistics, etc..
  For Example: netstat -tuln lists all listening ports (TCP/UDP).

- telnet
  telnet establishes a simple connection to a remote host for testing purposes. Mostly used for debugging network services such as HTTP, SMTP, etc. For Example:.
  telnet localhost 25 tests SMTP service running locally on port 25.
- arp
  arp manipulates or displays the system's ARP (Address Resolution Protocol) cache. ARP maps IP addresses to MAC addresses.
- **hostname**
  hostname displays or sets the system's hostname.
  For Example: hostname displays current hostname.

Conclusion

This lab session provided valuable hands-on experience with network programming, basic Unix commands, and essential network tools and understanding the Client-Server model.

Understanding and using the following Networking Utility commands: ifconfig, netstat, ping, arp, telnet, ftp, finger, whois, etc.

Objective

To understand and use the Networking Utility commands: ifconfig, netstat, ping, arp, telnet, ftp, finger, whois, etc.

Theory

ifconfig (Interface Configuration)

ifconfig (interface configuration) is a command-line utility in Unix and Linux-based systems used for configuring network interfaces. It allows users to view and change the configuration of network interfaces, such as IP addresses, netmasks, broadcast addresses, and more.

netstat (Network Statistics)

netstat is a command-line utility used to display various network-related information on Unix-like operating systems, including active network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. It's a useful tool for network troubleshooting and monitoring.

```
C:\Users\2aayu>netstat -r
===========================================================================
Interface List
  4...70 1a b8 9d ef 14 ......Microsoft Wi-Fi Direct Virtual Adapter
 18...72 1a b8 9d ef 13 ......Microsoft Wi-Fi Direct Virtual Adapter #2
 15...70 1a b8 9d ef 13 ......Intel(R) Wi-Fi 6 AX201 160MHz
  1...........................Software Loopback Interface 1
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.1.254     192.168.1.69     35
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
      192.168.1.0    255.255.255.0         On-link      192.168.1.69    291
     192.168.1.69  255.255.255.255         On-link      192.168.1.69    291
    192.168.1.255  255.255.255.255         On-link      192.168.1.69    291
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link      192.168.1.69    291
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link      192.168.1.69    291
===========================================================================
Persistent Routes:
  None
```

ping (Packet Internet Groper)

ping is used to test the connectivity between two networked devices. It sends ICMP Echo Request packets to the target host and waits for Echo Reply packets. It also measures round-trip time and packet loss.

```
C:\Users\2aayu>ping youtube.com

Pinging youtube.com [2404:6800:4002:82c::200e] with 32 bytes of data:
Reply from 2404:6800:4002:82c::200e: time=19ms
Reply from 2404:6800:4002:82c::200e: time=19ms
Reply from 2404:6800:4002:82c::200e: time=21ms
Reply from 2404:6800:4002:82c::200e: time=21ms

Ping statistics for 2404:6800:4002:82c::200e:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 19ms, Maximum = 21ms, Average = 20ms

C:\Users\2aayu>
```

arp (Address Resolution Protocol)

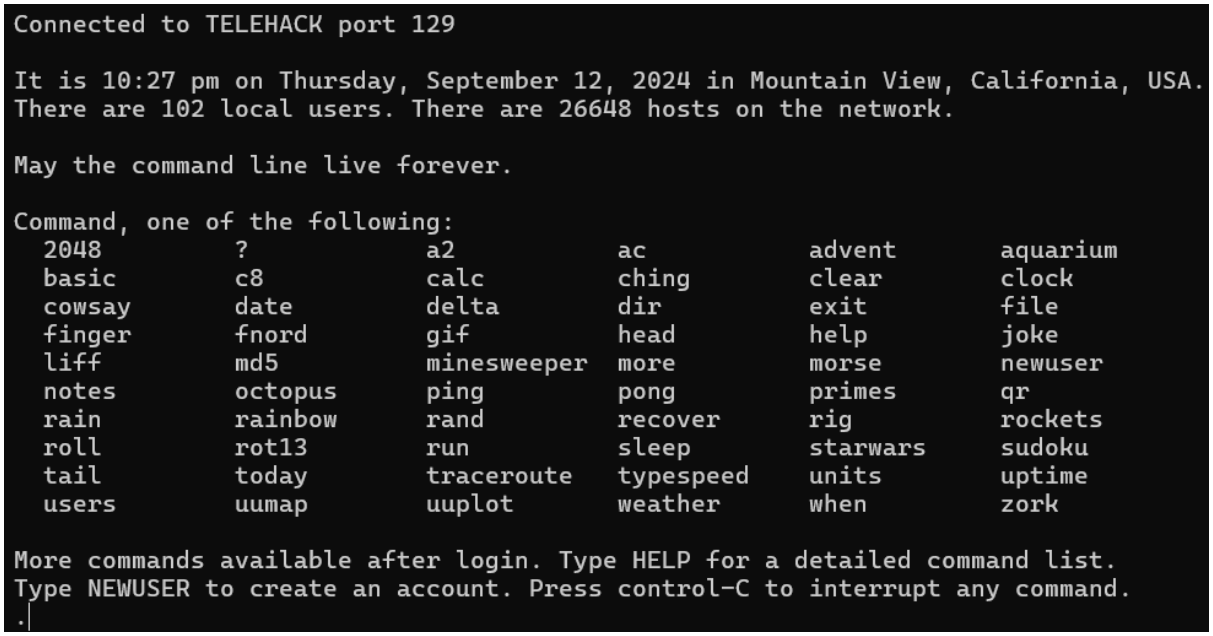
arp is used to view and manipulate the ARP table on a networked device. The ARP table maps IP addresses to MAC addresses, which is essential for communication on a local network.

```
C:\Users\2aayu>arp -a

Interface: 192.168.1.69 --- 0xf
  Internet Address      Physical Address      Type
  192.168.1.71          64-5a-04-2f-16-7b     dynamic
  192.168.1.254         54-37-bb-93-e7-94     dynamic
  192.168.1.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.192.152.143       01-00-5e-40-98-8f     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\Users\2aayu>
```

telnet (Telecommunication Network)

telnet is used to establish a remote connection to a server or another device over a network. It allows users to interact with the remote system's command-line interface.

```
C:\Users\2aayu>telnet telehack.com
```



ftp (File Transfer Protocol)
ftp is a command-line utility for transferring files between a local system and a remote server. It can be used to upload, download, and manage files on a remote server.

finger (User Information Lookup Program)
finger is used to retrieve information about system users, such as their login name, home directory, and the shell they are using. It can also show when they last logged in.

whois (Domain Information Lookup)
whois is used to query databases that store information about registered domain names, including the registrant, administrative contact, and expiration dates.

Conclusion
Therefore, we've studied and tested Networking Utility commands.

Lab 2. InetAddress Class

Create and use InetAddress objects and display the information of InetAddress.

Objective

To understand and demonstrate the use of InetAddress objects for retrieving and displaying information about IP addresses and hostnames.

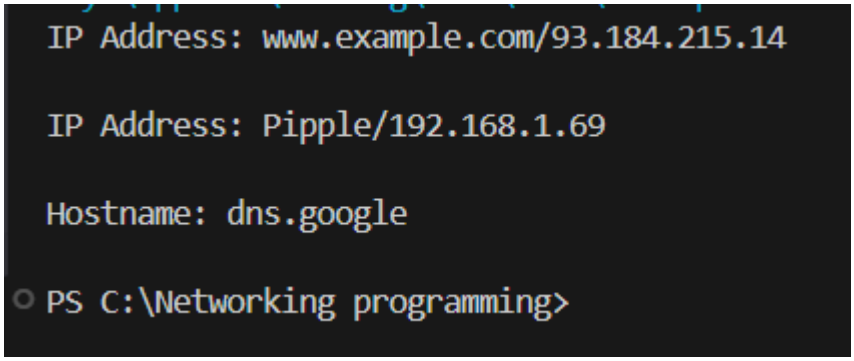
Theory

The InetAddress class in Java represents an IP address and provides methods to interact with the internet address information. You can use this class to get the IP address, hostname, or to resolve hostnames to IP addresses and vice versa. An IP address is a unique numerical identifier assigned to each device connected to a network, allowing devices to communicate with one another. Hostnames, on the other hand, are human-readable labels associated with IP addresses, making it easier to reference and remember them.

Source Code

```
import java.net.InetAddress;
import java.net.UnknownHostException;

public class NetworkInfoFinder {
   public static void main(String[] args) {

      try {
         InetAddress exampleAddress = InetAddress.getByName("www.example.com");
         System.out.println("IP Address: " + exampleAddress);
         System.out.println();

         InetAddress localAddress = InetAddress.getLocalHost();
         System.out.println("IP Address: " + localAddress);
         System.out.println();

         InetAddress googleDnsAddress = InetAddress.getByAddress(new byte[]{8, 8, 8, 8});
         System.out.println("Hostname: " + googleDnsAddress.getHostName());
         System.out.println();

      } catch (UnknownHostException e) {
         System.out.println("Error: Unable to resolve the hostname or IP address.");
      }
   }
}
```

Output



Conclusion

Therefore, in this lab,we've created and used InetAddress objects and displayed the information of InetAddress.

Create Network Interface and display the properties of a Network Interface.

Objective
To create Network Interface and display the properties of a Network Interface.

Theory
A **network interface** is the point of interaction between a computer system and a network. It can be hardware or software and is responsible for managing the transmission of data between the network and the device (such as a computer, server, or router). The interface ensures that the system can communicate with other devices over the network, enabling tasks such as sending and receiving data packets.
To interact with and manage network interfaces, several networking utility commands can be used:

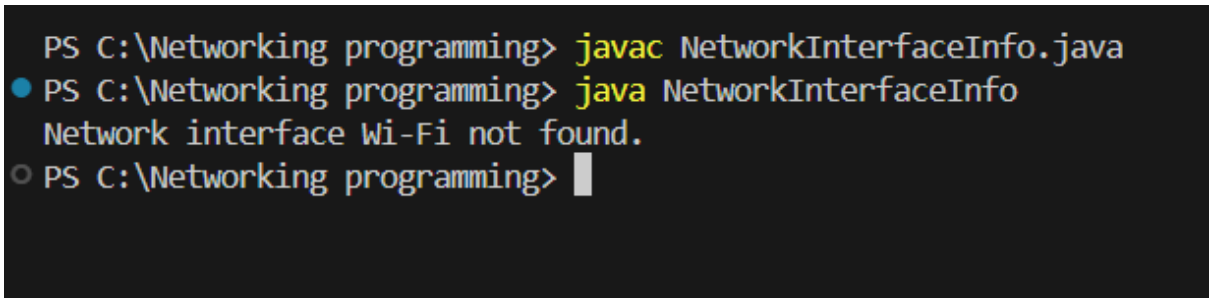
- **ifconfig**: This command is used to configure, manage, and query network interfaces on Unix-based systems.
- **ip**: This modern replacement for ifconfig is used to display and manage routing, devices, policy routing, and tunnels.

Source Code

```
import java.net.NetworkInterface;
import java.net.SocketException;

public class NetworkInterfaceInfo {

	public static void main(String[] args) {

		String targetInterfaceName = "en0";

		try {
			// Find and display properties of the specific network interface
			NetworkInterface networkInterface = NetworkInterface.getByName(targetInterfaceName);
			if (networkInterface == null) {
			System.out.println("Network interface " + targetInterfaceName + " not found.");
			return;
		}

		// Display properties
		System.out.println("Interface Name: " + networkInterface.getName());
		System.out.println("Display Name: " + networkInterface.getDisplayName());
		System.out.println("Is Up: " + networkInterface.isUp());
		System.out.println("Is Loopback: " + networkInterface.isLoopback());
```

```
                System.out.println("Is Virtual: " + networkInterface.isVirtual());
                System.out.println("Is Point-to-Point: " + networkInterface.isPointToPoint());
                System.out.println("Supports Multicast: " +
                networkInterface.supportsMulticast());
                // Display IP addresses using Streams
                networkInterface.getInetAddresses().asIterator().forEachRemaining(address ->
                System.out.println("  IP Address: " + address.getHostAddress())
                );
        } catch (SocketException e) {
                System.out.println("Error retrieving network interfaces.");
        }
}
```

Output



Conclusion

Therefore, we created a Network Interface and displayed the properties of a Network Interface.

Lab 3. URLs and URIs

Write a Java Program that uses all eight methods of the URL class to split URLs entered on the command line into their component parts.

Theory

A URI is a broader concept that encompasses both URLs and URNs (Uniform Resource Names). It is a string of characters used to identify a resource either by location, name, or both. URLs are a subset of URIs used to locate resources on the internet.

A URL is a reference to a web resource that specifies its location on a network and how to retrieve it. It is the most common form of URI (Uniform Resource Identifier). A URL typically consists of several components that describe the resource's location and access protocol.

Objective

To write a Java Program that uses all eight methods of the URL class to split URLs entered on the command line into their component parts.

The class includes methods that allow you to retrieve various components of a URL:

- **getProtocol()**: Returns the protocol (e.g., http, https).
- **getHost()**: Returns the hostname (e.g., www.example.com).
- **getPort()**: Returns the port number, or -1 if not specified.
- **getPath()**: Returns the path portion of the URL.
- **getQuery()**: Returns the query string, or null if not present.
- **getRef()**: Returns the reference (fragment), or null if not present.
- **getFile()**: Returns the file part of the URL, which includes the path and query.
- **getAuthority()**: Returns the authority of the URL, combining host and port.

Source Code

```
import java.net.MalformedURLException;
import java.net.URL;

public class Ex {

	public static void main(String[] args) {

		 if (args.length != 1) {
			System.out.println("Usage: java URLComponents <URL>");
			 return;
			}

		String urlString = args[0];
		try {
			// Create URL object from the command line argument
```

```
                URL url = new URL(urlString);

                // Display the various components of the URL
                System.out.println("URL: " + url.toString());
                System.out.println("Protocol: " + url.getProtocol());
                System.out.println("Host: " + url.getHost());
                System.out.println("Port: " + url.getPort());
                System.out.println("Path: " + url.getPath());
                System.out.println("Query: " + url.getQuery());
                System.out.println("File: " + url.getFile());
                System.out.println("Authority: " + url.getAuthority());
                System.out.println("Ref: " + url.getRef());

            } catch (MalformedURLException e) {
                System.out.println("Invalid URL format: " + e.getMessage());
            }
        }
}
```

Output

Conclusion

Therefore, this lab demonstrated how to use the URL class in Java to parse and display various components of a URL. By employing methods such as getProtocol(), getHost(), getPort(), and others, the program successfully extracts and shows different parts of the URL.

Write a program that reads a URL from the command line, then prints the raw data located at the URL.

Objective

The objective of this lab is to develop a program that reads a URL from the command line, retrieves the raw data located at that URL, and prints it to the console.

Source Code

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;

public class URLReader {

	public static void main(String[] args) {

		if (args.length != 1) {
			System.out.println("Usage: java URLReader <URL>");
			 return;
		}

		String urlString = args[0];

		 try {
			// Create URL object
			URL url = new URL(urlString);

			// Open connection to the URL
			HttpURLConnection connection = (HttpURLConnection) url.openConnection();
			connection.setRequestMethod("GET");

			 // Check response code
			int responseCode = connection.getResponseCode();
			if (responseCode == HttpURLConnection.HTTP_OK) {
				// Read the response
				BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
				String inputLine;
				StringBuilder response = new StringBuilder();
```

```
                    while ((inputLine = in.readLine()) != null) {
                        response.append(inputLine).append("\n");
                        }
                    in.close();

                     // Print the raw data
                    System.out.println("Raw data from the URL:");
                    System.out.println(response.toString());
                    }else {
                    System.out.println("Failed to retrieve data. HTTP response code: " + responseCode);
                }

                } catch (IOException e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
}
```

Output

```
Raw data from the URL:
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
```

Conclusion

This lab demonstrated how to read and print raw data from a URL.

Lab 4. URLConnection

Theory

URLConnection is a class in the java.net package that provides a flexible and powerful way to handle HTTP and other types of network connections. It is used to connect to and interact with resources identified by a URL. The URLConnection class provides methods for reading from and writing to a resource, as well as handling various types of requests and responses.

Write a Java program to create a URLConnection using the openConnection() method of the URL object and then use it to examine the document's properties and content.

Objective

To create a URLConnection using the openConnection() method of the URL object and then use it to examine the document's properties and content.

Source Code

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.util.List;
import java.util.Map;

public class URLConnectionExample {

	public static void main(String[] args) {

		if (args.length != 1) {
			System.out.println("Usage: java URLConnectionExample <URL>");
			return;
		}

		String urlString = args[0];

		try {
			// Create a URL object
			URL url = new URL(urlString);

			// Open a connection to the URL
			URLConnection urlConnection = url.openConnection();
```

```
                    // Display document properties
                    System.out.println("URL: " + url);
                    System.out.println("Content Type: " +
                    urlConnection.getContentType());
                    System.out.println("Content Length: " +
                    urlConnection.getContentLength());
                    System.out.println("Date: " + urlConnection.getDate());
                    System.out.println("Last Modified: " +
                    urlConnection.getLastModified());
                    System.out.println("Expiration: " + urlConnection.getExpiration());

                    // Display the header fields
                    System.out.println("\n--- Header Fields ---");
                    Map<String, List<String>> headerFields =
                    urlConnection.getHeaderFields();
                    for (Map.Entry<String, List<String>> entry : headerFields.entrySet()) {
                        System.out.println(entry.getKey() + ": " + entry.getValue());
                    }

                    // Read and display the content
                    System.out.println("\n--- Content ---");
                    BufferedReader in = new BufferedReader(new
                    InputStreamReader(urlConnection.getInputStream()));
                    String inputLine;
                    while ((inputLine = in.readLine()) != null) {
                        System.out.println(inputLine);
                    }
                    in.close();

                } catch (Exception e) {
                    System.out.println("Error: " + e.getMessage());
                }
        }
}
```

Output

```
PS C:\Networking programming> java URLConnectionExample https://www.w3schools.com/xml/note.xml
URL: https://www.w3schools.com/xml/note.xml
Content Type: text/xml
Content Length: 164
Date: 1726381564000
Last Modified: 1725619510000
Expiration: 0

--- Header Fields ---
null: [HTTP/1.1 200 OK]
X-Cache: [HIT]
Server: [ECS (nag/998C)]
Last-Modified: [Fri, 06 Sep 2024 10:45:10 GMT]
X-Content-Security-Policy: [frame-ancestors 'self' https://mycourses.w3schools.com https://pathfinder.w3schools.com;]
Date: [Sun, 15 Sep 2024 06:26:04 GMT]
Cache-Control: [public,max-age=31536000,public]
Etag: ["0bf5ad8490db1:0+ident"]
Content-Security-Policy: [frame-ancestors 'self' https://mycourses.w3schools.com https://pathfinder.w3schools.com;]
Vary: [Accept-Encoding]
Content-Length: [164]
Age: [757033]
Content-Type: [text/xml]
X-Powered-By: [ASP.NET]

--- Content ---
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
PS C:\Networking programming>
```

Conclusion

Therefore. we created a URLConnection using the openConnection() method of the URL object and then use it to examine the document's properties and content.

Write a Java program to create a URLConnection and display the response code and message.

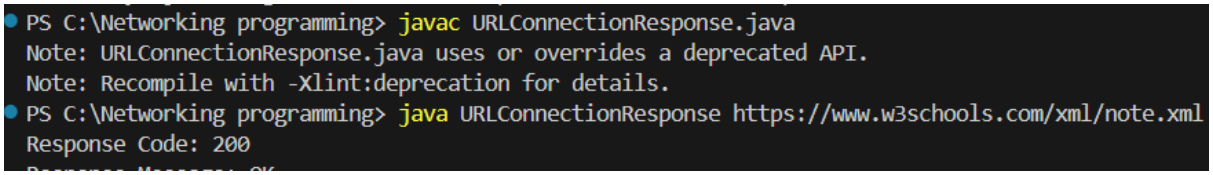
Objective

To create a URLConnection and display the response code and message.

Source Code

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class DayTimeUDPClient {
	private static final int PORT = 13;
	private static final String HOSTNAME = "localhost";

	public static void main(String[] args) {
		try (DatagramSocket socket = new DatagramSocket()) {
			socket.setSoTimeout(10000);
			InetAddress host = InetAddress.getByName(HOSTNAME);
			DatagramPacket request = new DatagramPacket(new byte[1], 1, host,
			PORT);
			DatagramPacket response = new DatagramPacket(new byte[1024],
			1024);
			socket.send(request);
			socket.receive(response);
			String result = new String(response.getData(), 0, response.getLength(),
			"US-ASCII");
			System.out.println(result);

		} catch (IOException ex) {
			ex.printStackTrace();
		}
	}
}
```

Output



Conclusion

Therefore this lab helps to create a URLConnection and display the response code and message.

Lab 5 - Socket Programming

Theory

Socket programming is a fundamental aspect of network programming that enables communication between computers over a network. Sockets provide a standard interface for network communication and are used in various applications, including web servers, client-server applications, and distributed systems. A socket is an endpoint for sending or receiving data across a computer network. It provides a way for programs to communicate over a network by abstracting the details of the underlying network protocols.

Create a simple client and server sockets that implement the Daytime service in Java.

Objective

To create a simple client and server sockets that implement the **Daytime service** in Java.

Source code

DayTimeUDPClient.java

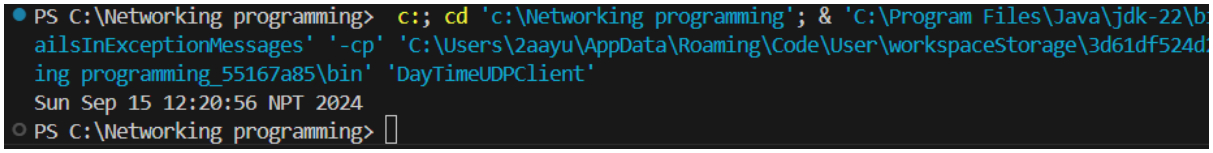
```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class DayTimeUDPClient {
	private static final int PORT = 13;
	private static final String HOSTNAME = "localhost";

	public static void main(String[] args) {
		try (DatagramSocket socket = new DatagramSocket()) {
			socket.setSoTimeout(10000);
			InetAddress host = InetAddress.getByName(HOSTNAME);
			DatagramPacket request = new DatagramPacket(new byte[1], 1, host,
			PORT);
			DatagramPacket response = new DatagramPacket(new byte[1024],
			1024);
			socket.send(request);
			socket.receive(response);
			String result = new String(response.getData(), 0, response.getLength(),
			"US-ASCII");
			System.out.println(result);
		} catch (IOException ex) {
			ex.printStackTrace();
```

```
            }
        }
}
```

DaytimeUDPServer.java

```
import java.net.DatagramPacket;
import java.io.IOException;
import java.net.*;
import java.util.Date;
import java.util.logging.*;

public class DaytimeUDPServer {
        private final static int PORT = 13;
        private final static Logger audit = Logger.getLogger("requests");
        private final static Logger errors = Logger.getLogger("errors");

        public static void main(String[] args){
                try (DatagramSocket socket = new DatagramSocket(PORT)) {
                        while (true) {
                                try {
                                        DatagramPacket request = new DatagramPacket(new
                                        byte[1024], 1024);
                                        socket.receive(request);

                                        String daytime = new Date().toString();
                                        byte[] data = daytime.getBytes("US-ASCII");
                                        DatagramPacket response = new DatagramPacket(data,
                                        data.length, request.getAddress(), request.getPort());
                                        socket.send(response);
                                        audit.info(daytime + " " + request.getAddress());

                                }catch (IOException | RuntimeException ex) {
                                        errors.log(Level.SEVERE, ex.getMessage(), ex);
                                }
                        }
                } catch (IOException ex) {
                        errors.log(Level.SEVERE, ex.getMessage(), ex);
                }
        }
}
```

Output



Conclusion

Therefore, we created a simple client and server sockets that implement the Daytime service in Java.

Create a Java-based simple Low Port Scanner program.

Objective

To Create a Java-based simple Low Port Scanner program

Source Code

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

public class LowPortScanner {
	public static void main(String[] args) {
		String host = "localhost";
		if (args.length > 0) {
			host = args[0];  // Use the host provided as a command-line argument
		}

		for (int i = 1; i < 1024; i++) {
			try {
				// Attempt to create a socket connection to the host and port
				Socket s = new Socket(host, i);
				System.out.println("Port " + i + " on " + host + " is open.");
				s.close(); // Close the socket after checking
			} catch (UnknownHostException e) {
				// Handle the case where the host cannot be resolved
				System.err.println("Unknown host: " + e.getMessage());
				break;
			} catch (IOException e) {
				// Handle the case where the port is not open or other IO errors
				// Do nothing here; we assume the port is closed or there was an
				error
			}
		}
	}
}
```

Output

```
PS C:\Networking programming>  & 'C:\Program Files\Java\jdk-22\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMes
cp' 'C:\Users\2aayu\AppData\Roaming\Code\User\workspaceStorage\3d61df524d290e58ac0c30d861e6e167\redhat.java\jdt_ws\Networking progr
167a85\bin' 'ex'
Port 135 on localhost is open.
Port 445 on localhost is open.
PS C:\Networking programming>
```

Conclusion

Therefore, we created a Java-based simple **Low Port Scanner** program

Lab 6. Advanced Socket Programming

Write a simple program to implement whois as a simple directory service protocol using a socket.

Objective

To implement whois as a simple directory service protocol using a socket.

Source Code

WhoisServer.java

```
import java.io.*;
import java.net.*;
import java.util.HashMap;
import java.util.Map;

public class WhoisServer {
	private static final int PORT = 4321;
	private static Map<String, String> directory = new HashMap<>();

	public static void main(String[] args) {
		directory.put("example.com", "93.184.216.34");
		directory.put("google.com", "142.250.190.14");
		directory.put("facebook.com", "31.13.71.36");

		try (ServerSocket serverSocket = new ServerSocket(PORT)) {
			System.out.println("WHOIS Server is running...");

			while (true) {
				try (Socket clientSocket = serverSocket.accept();
					BufferedReader in = new BufferedReader(new
					InputStreamReader(clientSocket.getInputStream()));
					PrintWriter out = new
					PrintWriter(clientSocket.getOutputStream(), true)) {

					String domainName = in.readLine();
					System.out.println("Received query for: " +
					domainName);

					String ipAddress =
					directory.getOrDefault(domainName, "Domain not
					found");
					System.out.println(ipAddress);
					}catch (IOException e) {
```

```
                                System.out.println("Error handling client: " +
                                e.getMessage());
                        }
                    }
                } catch (IOException e) {
                    System.out.println("Could not start server: " + e.getMessage());
                }
        }
}
```

WhoisClient.java

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

public class WhoisClient {
        private static final String SERVER_ADDRESS = "localhost";
        private static final int SERVER_PORT = 4321;

        public static void main(String[] args) {
                if (args.length != 1) {
                        System.out.println("Usage: java WhoisClient <domain_name>");
                        return;
                }

                 String domainName = args[0];

                try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
                        BufferedReader in = new BufferedReader(new
                        InputStreamReader(socket.getInputStream()));
                        PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {

                        System.out.println(domainName);
                        String response = in.readLine();
                        System.out.println("IP address for " + domainName + ": " +response);

                } catch (IOException e) {
                        System.out.println("Error: " + e.getMessage());
                }
        }
}
```

Output

```
PS C:\Networking programming>  & 'C:\Program Files\Java\jdk-22\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-
cp' 'C:\Users\2aayu\AppData\Roaming\Code\User\workspaceStorage\3d61df524d290e58ac0c30d861e6e167\redhat.java\jdt_ws\Networking programming_55
167a85\bin' 'WhoisServer'
WHOIS Server is running...
```

Conclusion

Therefore, we to implemented **whois** as a simple directory service protocol using a socket.

Write a Java program to create a SingleFileHTTPServer program using ServerSocket that is supplied with the filename, local port, and content encoding from the command line.

Objective

To create a SingleFileHTTPServer program using ServerSocket that is supplied with the filename, local port, and content encoding from the command line.

Source Code

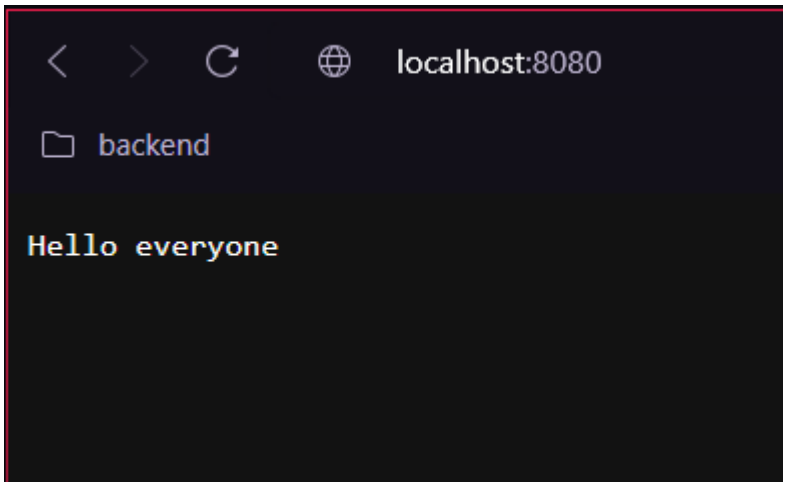
```
import java.io.*;
import java.net.*;

public class SingleFileHTTPServer {
	public static void main(String[] args) {
		if (args.length != 3) {
			 System.out.println("Usage: java SingleFileHTTPServer <filename> 
			<port> <encoding>");
			return;
		}

		String filename = args[0];
		int port = Integer.parseInt(args[1]);
		String encoding = args[2];

		try (ServerSocket serverSocket = new ServerSocket(port)) {
			System.out.println("Server is listening on port " + port);

			while (true) {
				try (Socket socket = serverSocket.accept()) {
					System.out.println("Client connected");

					// Read the HTTP request
					 BufferedReader in = new BufferedReader(new 
					InputStreamReader(socket.getInputStream()));
					String requestLine = in.readLine();
					if (requestLine != null && !requestLine.isEmpty()) {
						System.out.println("Request: " + requestLine);

						// Send the HTTP response
						OutputStream out = socket.getOutputStream();
						File file = new File(filename);
						if (file.exists() && !file.isDirectory()) {
							// Send HTTP headers
```

```
                                        out.write(("HTTP/1.1 200 OK\r\n").getBytes());
                                        out.write(("Content-Type: text/plain; charset=" + encoding + "\r\n").getBytes());
                                        out.write(("Content-Length: " + file.length() + "\r\n").getBytes());
                                        out.write(("\r\n").getBytes());

                                        // Send file content
                                        try (FileInputStream fileInputStream = new FileInputStream(file)) {
                                            byte[] buffer = new byte[4096];
                                            int bytesRead;
                                            while ((bytesRead = fileInputStream.read(buffer)) != -1){
                                                out.write(buffer, 0, bytesRead);
                                            }
                                        }
                                    } else {
                                        // File not found
                                        out.write(("HTTP/1.1 404 Not Found\r\n").getBytes());
                                        out.write(("Content-Type: text/plain; charset=" + encoding + "\r\n").getBytes());
                                        out.write(("\r\n").getBytes());
                                        out.write(("File not found.\r\n").getBytes());
                                    }
                                }
                            } catch (IOException e) {
                                System.err.println("Error handling client: " + e.getMessage());
                            }
                        }
                    } catch (IOException e) {
                        System.err.println("Error starting server: " + e.getMessage());
                    }
                }
            }
```

Sample.txt
Hello everyone

Output

```
PS C:\Networking programming> java SingleHTTP sample.txt 8080 UTF-8
Server is listening on port 8080
Client connected
Request: GET / HTTP/1.1
Client connected
Request: GET /favicon.ico HTTP/1.1
```



Conclusion

Therefore, a SingleFileHTTPServer program using ServerSocket that is supplied with the filename, local port, and content encoding from the command line is created.

Lab 7. Building a UDP Client/Server

Create and run a Java-based echo client and echo server application and display its working.

Objective

To create and run a Java-based echo client and echo server application and display its working.

Theory

UDP is a connectionless communication protocol in the Internet Protocol suite. It is used for transmitting data over a network in a way that is efficient and simple, but it does not guarantee delivery, order, or integrity of the data packets.

DatagramSocket:

- **send(DatagramPacket packet)**: Sends a datagram packet to the specified address and port.
- **receive(DatagramPacket packet)**: Receives a datagram packet from the socket.
- **close()**: Closes the socket.

DatagramPacket:

- **getAddress()**: Gets the address of the sender.
- **getPort()**: Gets the port of the sender.
- **getData()**: Gets the data from the packet.
- **getLength()**: Gets the length of the data in the packet.

Source Code

DaytimeUDPClient.java

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.io.IOException;

public class DayTimeUDPClient {
	public static void main(String[] args) {
		final String SERVER_ADDRESS = "localhost";  // Server address
		final int SERVER_PORT = 9876;  // Port number of the server

		try (DatagramSocket socket = new DatagramSocket()) {
			String message = "Hello, UDP Server!";
			byte[] sendData = message.getBytes();
			InetAddress serverAddress =
			InetAddress.getByName(SERVER_ADDRESS);
```

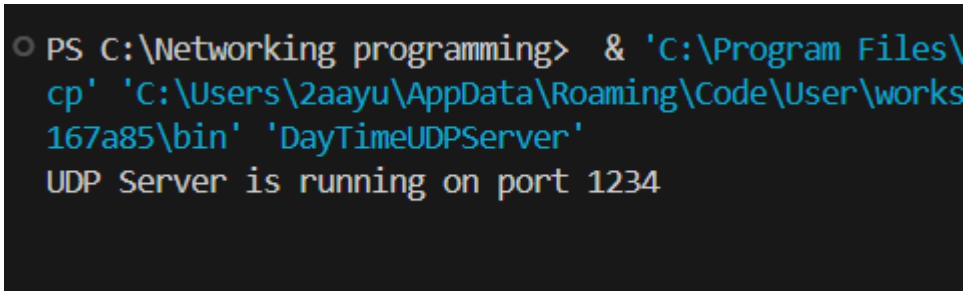
```
                // Send packet to the server
                DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, SERVER_PORT);
                socket.send(sendPacket);

                // Receive response from the server
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
                socket.receive(receivePacket);

                String responseMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("Received response: " + responseMessage);
            } catch (IOException e) {
                System.err.println("I/O error: " + e.getMessage());
            }
        }
}
```

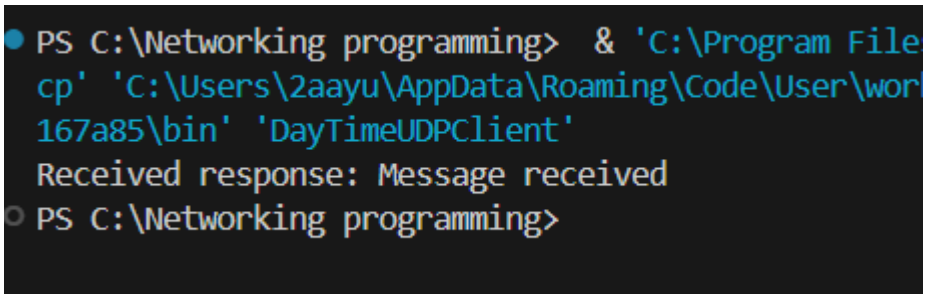
DaytimeUDPServer.java

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.io.IOException;

public class DaytimeUDPServer {
    public static void main(String[] args) {
        final int PORT = 9876;  // Port number for the server

        try (DatagramSocket socket = new DatagramSocket(PORT)) {
            System.out.println("UDP Server is running on port " + PORT);

            byte[] receiveData = new byte[1024];

            while (true) {
                DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
                socket.receive(receivePacket);  // Receive the packet from the client

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

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

                // Prepare response
                String responseMessage = "Message received";
                DatagramPacket responsePacket = new DatagramPacket(
                responseMessage.getBytes(),
                responseMessage.length(),
                receivePacket.getAddress(),
                receivePacket.getPort()
                );
                socket.send(responsePacket);  // Send response back to client
            }
        } catch (SocketException e) {
            System.err.println("Socket error: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("I/O error: " + e.getMessage());
        }
    }
}
```

Output





Conclusion

Therefore, we created and run a Java-based echo client and echo server application and display its working.

Lab 8. Building RMI Client/Server

Create and run a Java-based RMI Server and Client application.

Objective
To create and run a Java-based RMI Server and Client application.

Theory
Remote Method Invocation (RMI) is a Java technology that allows for the execution of methods on an object located in a different Java Virtual Machine (JVM). This is particularly useful in distributed computing where different parts of a program might run on different machines. It allows for objects to communicate over a network as if they were local, enabling the development of distributed applications.

Key concepts in RMI are:

1. **Remote Objects**:
    - Objects that reside in a different JVM or on a different machine and can be invoked remotely.
2. **RMI Registry**:
    - A name service that allows remote objects to be registered and looked up by clients.
3. **Stub and Skeleton**:
    - **Stub**: A client-side proxy that represents the remote object. It forwards the method calls from the client to the actual remote object.
    - **Skeleton**: (Deprecated in newer Java versions) A server-side object that receives method calls from the stub and invokes them on the actual remote object.
4. **Remote Interface**:
    - An interface that declares the methods that can be invoked remotely. This interface extends java.rmi.Remote.
5. **RMI Server**:
    - The server application that creates and exports remote objects and registers them with the RMI registry.
6. **RMI Client**:
    - The client application that looks up remote objects in the RMI registry and invokes methods on them.

Source Code
Hello.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
public interface Hello extends Remote {
        String sayHello() throws RemoteException;
}
```

ImplExample.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ImplExample extends UnicastRemoteObject implements Hello {
   protected ImplExample() throws RemoteException {
      super();
   }

   @Override
   public String sayHello() throws RemoteException {
      return "Hello, World!";
   }
}
```

Client.java

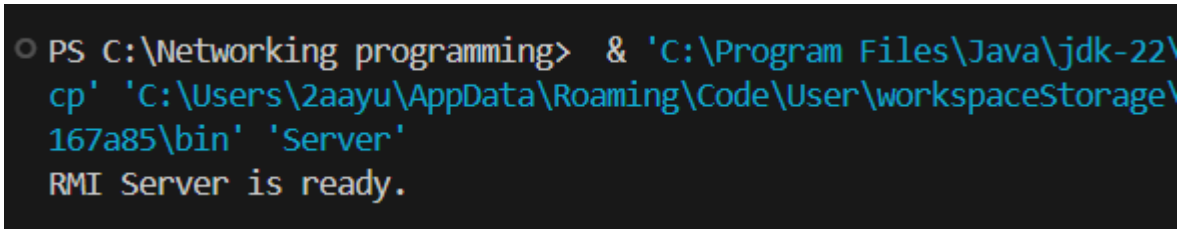
```
import java.rmi.Naming;

public class Client {
   public static void main(String[] args) {
      try {
         // Lookup the remote object
         Hello stub = (Hello) Naming.lookup("rmi://localhost/Hello");

         // Call the remote method
         String response = stub.sayHello();
         System.out.println(response);
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

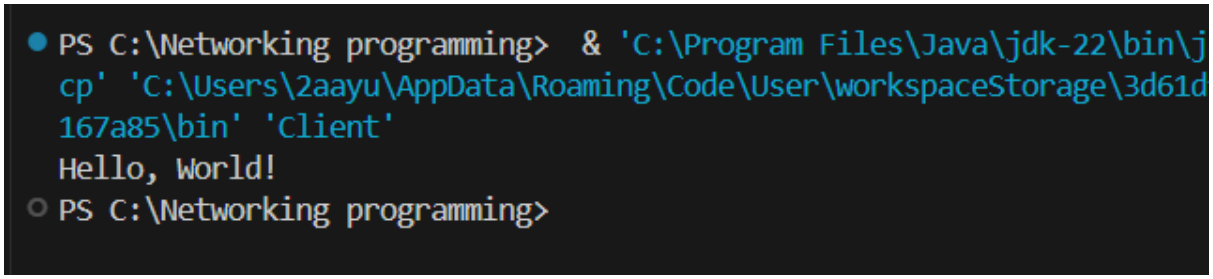
Server.java

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
```

```
public class Server {
   public static void main(String[] args) {
      try {
         // Create an instance of the implementation
         ImplExample obj = new ImplExample();

         // Create the RMI registry (if not already running)
         LocateRegistry.createRegistry(1099);

         // Bind the remote object to the RMI registry
         Naming.rebind("Hello", obj);

         System.out.println("RMI Server is ready.");
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Output





Conclusion

Therefore, Java-based RMI Server and Client application is created.