Write the use of the following network configuration commands in respective environment - Unix /Windows

i) tcpdump ii) netstat iii) ifconfig / ipconfig iv) nslookup v) traceroute

i) `tcpdump`

Unix/Linux:

- Description: `tcpdump` is a packet analyzer that allows you to display and analyze network traffic.

- Usage:

```bash
tcpdump [options] [expression]
```

- Example:

```bash
sudo tcpdump -i eth0
```

Windows:

- Windows doesn't have a built-in equivalent to `tcpdump`. However, you can use tools like Wireshark for similar packet capturing and analysis.

ii) `netstat`

Unix/Linux:

- Description: `netstat` displays network connections, routing tables, interface statistics, masquerade connections, etc.

- Usage:

```bash
netstat [options]
```

- Example:

```bash
netstat -an
```

Windows:

- Description: `netstat` in Windows is similar but uses slightly different options.

- Usage:

  ```cmd
  netstat [options]
  ```

- Example:

  ```cmd
  netstat -a
  ```

iii) `ifconfig` / `ipconfig`

Unix/Linux:

- Description: `ifconfig` displays or configures network interface parameters on Unix/Linux systems.

- Usage:

  ```bash
  ifconfig [interface] [options]
  ```

- Example:

  ```bash
  ifconfig eth0
  ```

Windows:

- Description: `ipconfig` displays IP configuration information for all network interfaces.

- Usage:

  ```cmd
  ipconfig [options]
  ```

- Example:

```cmd
ipconfig /all
```

iv) `nslookup`

Unix/Linux:

- Description: `nslookup` is a command-line tool for querying DNS servers to obtain domain name or IP address mapping or for any other DNS records.

- Usage:

```bash
nslookup [options] [name | -] [server]
```

- Example:

```bash
nslookup example.com
```

Windows:

- Description: `nslookup` works similarly on Windows.

- Usage:

```cmd
nslookup [options] [name | -] [server]
```

- Example:

```cmd
nslookup example.com
```

v) `traceroute`

Unix/Linux:

- Description: `traceroute` traces the route that packets take to get from your computer to a destination IP address or domain.

- Usage:

  ```bash
  traceroute [options] [host]
  ```

- Example:

  ```bash
  traceroute example.com
  ```

Windows:

- Description: `tracert` is the Windows equivalent of `traceroute`.

- Usage:

  ```cmd
  tracert [options] [host]
  ```

- Example:

  ```cmd
  tracert example.com
  ```

Write the Echo client and server programs using UDP. The Echo clients should verify whether the textstring they received from the server is the same text string that they sent or not.

UDPEchoServer.java:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class UDPEchoServer {
    public static void main(String[] args) {
        final int serverPort = 9999;

        try (DatagramSocket serverSocket = new DatagramSocket(serverPort)) {
            System.out.println("UDP Echo server is listening on port " +
serverPort);

            while (true) {
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);

                // Receive packet from client
                serverSocket.receive(receivePacket);

                String clientMessage = new String(receivePacket.getData(), 0,
receivePacket.getLength());
                System.out.println("Received message from " +
receivePacket.getAddress() + ": " + clientMessage);

                // Echo the message back to the client
                serverSocket.send(new DatagramPacket(receivePacket.getData(),
receivePacket.getLength(), receivePacket.getAddress(), receivePacket.getPort()));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

UDPEchoClient.java:

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

public class UDPEchoClient {
```

```
    public static void main(String[] args) {
        final String serverHost = "localhost";
        final int serverPort = 9999;

        try (DatagramSocket clientSocket = new DatagramSocket()) {
            InetAddress serverAddress = InetAddress.getByName(serverHost);

            // Message to be sent
            String message = "Hello, UDP Server!";
            byte[] sendData = message.getBytes();

            // Send packet to the server
            DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, serverAddress, serverPort);
            clientSocket.send(sendPacket);

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

            String serverResponse = new String(receivePacket.getData(), 0,
receivePacket.getLength());
            System.out.println("Received message from server: " +
serverResponse);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Write a simple HTTP web client program using TCP sockets to download a web page. Get the URL andpass it for buffering the content and write it as a html file and make it to get downloaded.

SimpleHttpClient.java:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.net.URL;

public class SimpleHttpClient {

    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("Usage: java SimpleHttpClient <URL>");
            System.exit(1);
        }

        String urlStr = args[0];
        String host = "";
        String path = "/";

        try {
            // Parse the URL to extract host and path
            URL url = new URL(urlStr);
            host = url.getHost();
            path = url.getPath().isEmpty() ? "/" : url.getPath();
        } catch (Exception e) {
            System.err.println("Invalid URL format");
            System.exit(1);
        }

        // Create socket
        try (Socket socket = new Socket(host, 80)) {
            // Send HTTP GET request
            BufferedWriter out = new BufferedWriter(new
FileWriter("downloaded_page.html"));
            out.write("GET " + path + " HTTP/1.1\r\n");
            out.write("Host: " + host + "\r\n");
            out.write("Connection: close\r\n");
            out.write("\r\n");
            out.flush();
```

```
            // Receive and buffer the content
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            StringBuilder response = new StringBuilder();
            String line;

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

            // Save the content to an HTML file
            saveToFile("downloaded_page.html", response.toString());

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void saveToFile(String filename, String content) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filename))) {
            writer.write(content);
            System.out.println("File '" + filename + "' saved successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Write a program to implement the DNS using UDP sockets. The DNS should have different domains with corresponding IP addresses. Resolve the user given domain with the IP and display it to the user. Also indicate the error message to the user when domain is not resolved

DNSServer.java:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.HashMap;
import java.util.Map;

public class DNSServer {
    private static final Map<String, String> dnsRecords = new HashMap<>();

    static {
        // Predefined DNS records
        dnsRecords.put("example.com", "192.168.1.1");
        dnsRecords.put("google.com", "8.8.8.8");
        dnsRecords.put("yahoo.com", "98.138.219.231");
        // Add more domains and corresponding IP addresses as needed
    }

    public static void main(String[] args) {
        try {
            // Create UDP socket
            DatagramSocket socket = new DatagramSocket(9876);

            System.out.println("DNS Server is running...");

            while (true) {
                // Receive client request
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
                socket.receive(receivePacket);

                // Process DNS request
                String domain = new String(receivePacket.getData(), 0,
receivePacket.getLength());
                System.out.println("Received DNS request for domain: " + domain);

                // Lookup IP address
```

```
                String ipAddress = dnsRecords.getOrDefault(domain, "Domain not
found");

                // Send response to the client
                InetAddress clientAddress = receivePacket.getAddress();
                int clientPort = receivePacket.getPort();
                byte[] sendData = ipAddress.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, clientAddress, clientPort);
                socket.send(sendPacket);

                System.out.println("Sent DNS response to client: " + ipAddress +
"\n");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

DNSClient.java:

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class DNSClient {
    public static void main(String[] args) {
        try {
            // Create UDP socket
            DatagramSocket socket = new DatagramSocket();

            // Get user input for the domain to resolve
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter domain to resolve: ");
            String domain = scanner.nextLine();

            // Send domain to DNS server
            byte[] sendData = domain.getBytes();
            InetAddress serverAddress = InetAddress.getByName("localhost");
            int serverPort = 9876;
            DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, serverAddress, serverPort);
            socket.send(sendPacket);
```

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

            // Display the resolved IP address
            String ipAddress = new String(receivePacket.getData(), 0,
receivePacket.getLength());
            System.out.println("Resolved IP address: " + ipAddress);

            // Close the socket
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Write a program a client-server application for CHAT using TCP

ChatServer.java:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class ChatServer {

    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Server is listening on port 12345...");

            // Wait for a client to connect
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientSocket);

            // Create input and output streams for communication
            InputStream clientInput = clientSocket.getInputStream();
            OutputStream clientOutput = clientSocket.getOutputStream();

            // Create a thread for receiving messages from the client
            Thread receiveThread = new Thread(() -> {
                try {
                    while (true) {
                        byte[] buffer = new byte[1024];
                        int bytesRead = clientInput.read(buffer);
                        if (bytesRead == -1) {
                            break; // Connection closed by the client
                        }

                        String receivedMessage = new String(buffer, 0,
bytesRead);
                        System.out.println("Client: " + receivedMessage);
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
            receiveThread.start();

            // Create a thread for sending messages to the client
```

```
            Thread sendThread = new Thread(() -> {
                try {
                    while (true) {
                        // Read a message from the console
                        String message = System.console().readLine();

                        // Send the message to the client
                        clientOutput.write(message.getBytes());
                        clientOutput.flush();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
            sendThread.start();

            // Wait for both threads to finish
            receiveThread.join();
            sendThread.join();

            // Close the server socket
            serverSocket.close();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

ChatClient.java:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class ChatClient {

    public static void main(String[] args) {
        try {
            // Connect to the server
            Socket socket = new Socket("localhost", 12345);
            System.out.println("Connected to server: " + socket);

            // Create input and output streams for communication
            InputStream serverInput = socket.getInputStream();
```

```
            OutputStream serverOutput = socket.getOutputStream();

            // Create a thread for receiving messages from the server
            Thread receiveThread = new Thread(() -> {
                try {
                    while (true) {
                        byte[] buffer = new byte[1024];
                        int bytesRead = serverInput.read(buffer);
                        if (bytesRead == -1) {
                            break; // Connection closed by the server
                        }

                        String receivedMessage = new String(buffer, 0,
bytesRead);
                        System.out.println("Server: " + receivedMessage);
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
            receiveThread.start();

            // Create a thread for sending messages to the server
            Thread sendThread = new Thread(() -> {
                try {
                    while (true) {
                        // Read a message from the console
                        String message = System.console().readLine();

                        // Send the message to the server
                        serverOutput.write(message.getBytes());
                        serverOutput.flush();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
            sendThread.start();

            // Wait for both threads to finish
            receiveThread.join();
            sendThread.join();

            // Close the socket
            socket.close();
```

```
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Write a code simulating ARP protocols for client and server.

ARPProtocolServer.java:

```
import java.util.HashMap;
import java.util.Map;

public class ARPProtocolServer {

    private Map<String, String> ipToMacMapping;

    public ARPProtocolServer() {
        this.ipToMacMapping = new HashMap<>();
        // Simulate a network with IP-MAC mappings
        ipToMacMapping.put("192.168.1.1", "00:11:22:33:44:55");
        ipToMacMapping.put("192.168.1.2", "11:22:33:44:55:66");
        ipToMacMapping.put("192.168.1.3", "22:33:44:55:66:77");
    }

    public String resolveMacAddress(String ipAddress) {
        return ipToMacMapping.getOrDefault(ipAddress, "MAC address not found");
    }

    public static void main(String[] args) {
        ARPProtocolServer server = new ARPProtocolServer();

        // Simulate ARP request from the client
        String clientIpAddress = "192.168.1.2";
        String clientMacAddress = server.resolveMacAddress(clientIpAddress);

        System.out.println("ARP Reply from Server:");
        System.out.println("IP Address: " + clientIpAddress);
        System.out.println("MAC Address: " + clientMacAddress);
    }
}
```

ARPProtocolClient.java:

```
public class ARPProtocolClient {

    public static void main(String[] args) {
        // Simulate ARP request to the server
        String serverIpAddress = "192.168.1.2";

        System.out.println("ARP Request from Client:");
        System.out.println("IP Address: " + serverIpAddress);
```

```
        // Assuming that the client sends an ARP request to the server
        // and receives a response with the MAC address
    }
}
```

Write a program to implement ARP protocols.

ARPProtocolSimulation.java:

```
import java.util.HashMap;
import java.util.Map;

public class ARPProtocolSimulation {

    private Map<String, String> ipToMacMapping;

    public ARPProtocolSimulation() {
        this.ipToMacMapping = new HashMap<>();
    }

    public void addMapping(String ipAddress, String macAddress) {
        ipToMacMapping.put(ipAddress, macAddress);
    }

    public String resolveMacAddress(String ipAddress) {
        return ipToMacMapping.get(ipAddress);
    }

    public static void main(String[] args) {
        ARPProtocolSimulation arpSimulator = new ARPProtocolSimulation();

        // Add some IP-MAC mappings to the simulation
        arpSimulator.addMapping("192.168.1.1", "00:11:22:33:44:55");
        arpSimulator.addMapping("192.168.1.2", "11:22:33:44:55:66");
        arpSimulator.addMapping("192.168.1.3", "22:33:44:55:66:77");

        // Simulate ARP request and response
        String ipAddressToResolve = "192.168.1.2";
        String resolvedMacAddress =
arpSimulator.resolveMacAddress(ipAddressToResolve);

        System.out.println("ARP Request:");
        System.out.println("IP Address to Resolve: " + ipAddressToResolve);

        if (resolvedMacAddress != null) {
            System.out.println("ARP Response:");
            System.out.println("Resolved MAC Address: " + resolvedMacAddress);
        } else {
            System.out.println("MAC Address not found for the given IP.");
        }
    }
```

```
}
```