Lab 3: Configure following options on server socket and tests them: SO_KEEPALIVE, SO_LINGER, SO_SNDBUF, SO_RCVBUF, TCP_NODELAY.

Explanation:

SO_KEEPALIVE:

SO_KEEPALIVE is a socket option that allows the operating system to automatically send keep-alive packets on an idle TCP connection to check if the other end is still responsive.

In the code, we set SO_KEEPALIVE to 1 (enabled) using setsockopt.

This option helps detect and close inactive connections more reliably.

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

public class ServerSocketExample {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(8080);

            // Enable SO_KEEPALIVE
            serverSocket.setKeepAlive(true);

            System.out.println("Server is running. Waiting for a client to connect...");

            Socket clientSocket = serverSocket.accept();

            // Test SO_KEEPALIVE
            System.out.println("SO_KEEPALIVE enabled: " + clientSocket.getKeepAlive());
```

```
            // Close the sockets
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

SO_LINGER:

SO_LINGER is a socket option that controls what happens when you try to close a socket that still has data to be sent or received.

In the code, we set SO_LINGER to (1, 30), which means that when we close the socket, it will linger for 30 seconds to send any remaining data.

After the timeout, the socket will be forcefully closed.

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

public class ServerSocketExample {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(8080);

            // Enable SO_LINGER with a timeout of 30 seconds
            serverSocket.setSoLinger(true, 30);
```

```
            System.out.println("Server is running. Waiting for a client to connect...");

            Socket clientSocket = serverSocket.accept();

            // Test SO_LINGER
            System.out.println("SO_LINGER enabled: " + clientSocket.getSoLinger());

            // Close the sockets
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

SO_SNDBUF and SO_RCVBUF:

SO_SNDBUF and SO_RCVBUF control the send and receive buffer sizes, respectively.

In the code, we set both to a buffer size of 8192 bytes using setsockopt.

These options allow you to tune the socket's buffer sizes, which can impact the performance of data transmission.

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

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

```
        try {
            ServerSocket serverSocket = new ServerSocket(8080);

            // Set SO_SNDBUF and SO_RCVBUF to custom values (e.g., 8192 bytes)
            serverSocket.setSendBufferSize(8192);
            serverSocket.setReceiveBufferSize(8192);

            System.out.println("Server is running. Waiting for a client to connect...");

            Socket clientSocket = serverSocket.accept();

            // Test SO_SNDBUF and SO_RCVBUF
            System.out.println("SO_SNDBUF: " + clientSocket.getSendBufferSize());
            System.out.println("SO_RCVBUF: " + clientSocket.getReceiveBufferSize());

            // Close the sockets
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

TCP_NODELAY:

TCP_NODELAY is a socket option that controls the Nagle's algorithm, which can introduce delays in small data packet transmission.

In the code, we set TCP_NODELAY to 1 (enabled) using setsockopt, disabling the algorithm.

This option can be useful when you want to reduce latency in applications that send small packets frequently.

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;

public class ServerSocketExample {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(8080);

            // Disable TCP_NODELAY
            serverSocket.setTcpNoDelay(false);

            System.out.println("Server is running. Waiting for a client to connect...");

            Socket clientSocket = serverSocket.accept();

            // Test TCP_NODELAY
            System.out.println("TCP_NODELAY enabled: " + !clientSocket.getTcpNoDelay());

            // Close the sockets
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

In the main() function, we create a server socket, bind it to an address and port, and start listening for incoming connections. When a client connects, it demonstrates the use of these options. For example, SO_KEEPALIVE will help ensure the connection is still alive, SO_LINGER controls how the socket behaves when closing, and SO_SNDBUF and SO_RCVBUF set the buffer sizes for data transmission. Additionally, TCP_NODELAY affects the transmission of small packets.

Testing these options would involve connecting to the server and observing their effects in real-world scenarios. You may need a client program to fully test these options as it requires interaction with the server. You can monitor network traffic and behavior to ensure that these socket options are functioning as expected in your application. Configuring and testing the socket options (SO_KEEPALIVE, SO_LINGER, SO_SNDBUF, SO_RCVBUF, TCP_NODELAY) on a server socket involves several steps.

Algorithm:

Initialize the Server Socket:

Create a server socket using the appropriate socket library in your programming language (e.g., Python's socket library).

Configure SO_KEEPALIVE:

Use the setsockopt method to enable SO_KEEPALIVE on the server socket. Set it to 1 (enabled).

Configure SO_LINGER:

Use the setsockopt method to configure SO_LINGER on the server socket. Set it with a tuple containing (1, timeout), where timeout is the desired linger timeout in seconds.

Configure SO_SNDBUF and SO_RCVBUF:

Use setsockopt to configure SO_SNDBUF and SO_RCVBUF on the server socket. Set the desired buffer sizes using appropriate values (e.g., 8192 bytes).

Configure TCP_NODELAY:

Use setsockopt to enable TCP_NODELAY on the server socket. Set it to 1 (enabled).

Bind and Listen:

Bind the server socket to a specific IP address and port.

Start listening for incoming connections using the listen method.

Accept Client Connections:

In a loop, accept incoming client connections using the accept method.

For each accepted connection, you can print a message to indicate that a connection has been established.

Test SO_KEEPALIVE:

Leave a client connection idle for a while (longer than the SO_KEEPALIVE interval).

Observe if the server socket sends keep-alive packets to the client to maintain the connection.

Test SO_LINGER:

Establish a client connection.

Simulate a scenario where the server closes the socket while data is still in the send buffer.

Observe if the SO_LINGER option delays the socket closure and ensures data transmission.

Test SO_SNDBUF and SO_RCVBUF:

Measure the data transmission performance by sending/receiving data between the server and client.

Adjust the buffer sizes and observe the impact on data transmission speed and efficiency.

Test TCP_NODELAY:

Measure the latency and performance of sending small packets over the connection with TCP_NODELAY enabled.

Compare this to the performance with TCP_NODELAY disabled to see the difference in packet transmission behavior.

Cleanup and Close:

Properly close the client sockets after testing.

Close the server socket when the server is finished testing or interrupted (e.g., by a keyboard interrupt).

Analyzing Results:

Examine the results of your tests and monitor network traffic, if necessary, to ensure that the socket options are behaving as expected.

Java Code:

```
import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.net.SocketException;


public class ServerSocketOptionsTest {

    public static void main(String[] args) {

        int port = 8080;


        try {

            // Create a server socket

            ServerSocket serverSocket = new ServerSocket(port);


            // Set SO_KEEPALIVE option (1 enables, 0 disables)

            serverSocket.setKeepAlive(true);


            // Set SO_LINGER option (Linger on close with a timeout of 30 seconds)

            serverSocket.setSoLinger(true, 30);


            // Set SO_SNDBUF option (send buffer size)

            serverSocket.setSendBufferSize(8192);
```

```
        // Set SO_RCVBUF option (receive buffer size)
        serverSocket.setReceiveBufferSize(8192);

        // Set TCP_NODELAY option (disable Nagle's algorithm, true enables, false disables)
        serverSocket.setTcpNoDelay(true);

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

        while (true) {
            // Accept incoming connections
            Socket clientSocket = serverSocket.accept();
            System.out.println("Accepted connection from " + clientSocket.getInetAddress());

            // You can perform any necessary server-side operations here

            // Close the client socket
            clientSocket.close();
        }
    } catch (SocketException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```