Modul 4. Stream dan datagram

Tugas Pendahuluan

1. Jelaskan secara singkat perbedaan antara stream dan datagram dalam pemrograman jaringan.
2. Jelaskan keuntungan dan kekurangan masing-masing model.
3. Tentukan situasi di mana penggunaan model stream lebih sesuai dan jelaskan alasannya.
4. Identifikasi skenario di mana datagram lebih efisien dan berikan contoh penggunaannya.

Materi :

Dalam pemrograman jaringan dengan Java, dua protokol utama yang digunakan untuk mentransfer data adalah TCP (Transmission Control Protocol) dan UDP (User Datagram Protocol). Konsep stream dan datagram berkaitan dengan cara data ditransfer menggunakan protokol ini.

Stream:

Stream adalah aliran data yang kontinu antara dua titik dalam jaringan, yang disampaikan dalam urutan berurutan. Biasayanya Terkait dengan protokol TCP.

Karakteristik Utama:

1. Stream berorientasi koneksi, artinya ada koneksi yang dapat diandalkan antara pengirim dan penerima.
2. Data ditransfer dalam bentuk aliran yang kontinu, tanpa batasan paket tertentu.
3. Protokol TCP menyediakan mekanisme untuk memastikan keandalan dan pengiriman data yang lengkap.

Contoh Penggunaan dalam Java:

Pada tingkat Java, `Socket` dan `ServerSocket` digunakan untuk implementasi stream dengan TCP.

Server (Stream):

```
ServerSocket serverSocket = new ServerSocket(8080);
Socket clientSocket = serverSocket.accept();
InputStream inputStream = clientSocket.getInputStream();
// Baca data dari inputStream
```

Klien (Stream):

```
Socket socket = new Socket("localhost", 8080);
OutputStream outputStream = socket.getOutputStream();
// Kirim data melalui outputStream
```

Datagram:

Datagram adalah paket data independen yang dikirimkan antara dua titik dalam jaringan tanpa membangun koneksi terlebih dahulu. Terkait dengan protokol UDP.

Karakteristik Utama:

1. Datagram tidak memerlukan koneksi terlebih dahulu; setiap paket dianggap independen.
2. Data dikirim dalam bentuk paket terpisah (datagram), dan tidak ada jaminan bahwa paket-paket tersebut akan sampai atau sampai dalam urutan yang benar.
3. Karena tidak memerlukan pengaturan koneksi, datagram sering lebih cepat tetapi kurang dapat diandalkan.

Contoh Penggunaan dalam Java:

Pada tingkat Java, `DatagramSocket` dan `DatagramPacket` digunakan untuk implementasi datagram dengan UDP.

Server (Datagram):

```
DatagramSocket serverSocket = new DatagramSocket(9876);
byte[] receiveData = new byte[1024];
DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
serverSocket.receive(receivePacket);
// Baca data dari receivePacket.getData()
```

Klien (Datagram):

```
DatagramSocket clientSocket = new DatagramSocket();
InetAddress serverAddress = InetAddress.getByName("localhost");
byte[] sendData = "Hello, Server!".getBytes();
DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, 9876);
clientSocket.send(sendPacket);
```

Perbandingan:

Stream (TCP):

- Berorientasi koneksi.
- Data dikirim sebagai aliran kontinu.
- Keandalan dan pengiriman data yang lengkap dijamin oleh protokol TCP.
- Digunakan ketika dibutuhkan koneksi yang andal dan dapat diandalkan.

Datagram (UDP):

- Tidak berorientasi koneksi.
- Data dikirim dalam paket terpisah.
- Tidak ada jaminan pengiriman atau pengiriman dalam urutan yang benar.
- Digunakan ketika kecepatan dan efisiensi lebih diutamakan daripada keandalan.

Pemilihan antara stream dan datagram tergantung pada kebutuhan spesifik aplikasi, seperti apakah keandalan atau kecepatan yang lebih diutamakan.

Situasi penggunaan

Penggunaan model stream dan datagram dalam pemrograman jaringan tergantung pada kebutuhan spesifik aplikasi dan karakteristik yang diinginkan. Berikut adalah situasi-situasi di mana penggunaan stream atau datagram lebih sesuai:

Menggunakan Stream:

1. Aplikasi yang Memerlukan Keandalan:

Contoh: Sistem perbankan online, transfer data file besar. Alasan: Model stream (TCP) menjamin keandalan pengiriman data. Dalam situasi di mana setiap byte data harus diterima dengan benar dan tidak boleh hilang, stream lebih sesuai.

2. Aplikasi yang Mengutamakan Integritas Data:

Contoh: Panggilan video konferensi, streaming media berkualitas tinggi. Alasan: Stream memastikan data diterima dalam urutan yang benar dan tanpa kehilangan paket. Ini penting untuk aplikasi yang membutuhkan integritas data.

3. Pertukaran Data Kontinu:

Contoh: Komunikasi real-time seperti permainan daring, telepon VoIP. Alasan: Model stream cocok untuk pertukaran data kontinu karena data dikirim sebagai aliran yang berkelanjutan, memungkinkan aplikasi untuk mengolah data secara dinamis.

Menggunakan Datagram:

1. Aplikasi yang Memerlukan Kecepatan:

Contoh: Pemantauan sensor suhu, aplikasi Internet of Things (IoT) Alasan: Datagram (UDP) tidak memerlukan pembangunan koneksi, sehingga lebih cepat. Situasi di mana kecepatan pengiriman data lebih diutamakan daripada keandalan lebih cocok untuk datagram.

2. Pertukaran Data Terpisah:

Contoh: Pengiriman pesan singkat, sistem notifikasi. Alasan: Datagram memungkinkan pengiriman paket data independen, berguna untuk aplikasi yang memerlukan pertukaran pesan atau informasi yang tidak terkait satu sama lain.

3. Aplikasi yang Toleran Terhadap Kehilangan Data:

Contoh: Layanan pencarian cepat, game online ringan. Alasan: Datagram tidak menjamin pengiriman data atau urutan paket, sehingga lebih sesuai untuk aplikasi yang dapat toleran terhadap kehilangan data tanpa perlu pertukaran data yang rumit.

Kombinasi Penggunaan:

Terkadang, aplikasi dapat memanfaatkan kedua model tergantung pada konteksnya. Misalnya, aplikasi panggilan video mungkin menggunakan model stream untuk audio dan model datagram untuk data video jika toleran terhadap sedikit kehilangan data.

Penting untuk mempertimbangkan karakteristik khusus dari masing-masing model (stream dan datagram) dan bagaimana mereka memenuhi kebutuhan fungsional dan kinerja dari suatu aplikasi.

Contoh 1:

Berikut adalah contoh aplikasi jaringan sederhana menggunakan Java untuk mengirimkan file ke server menggunakan model stream (TCP). Aplikasi ini terdiri dari dua bagian: server dan klien.

Server (FileServer.java):

```
1. import java.io.*;
2. import java.net.ServerSocket;
3. import java.net.Socket;
4.
5. public class FileServer {
6.     public static void main(String[] args) {
7.         final int PORT = 12345;
8.
9.         try {
10.            ServerSocket serverSocket = new ServerSocket(PORT);
11.            System.out.println("Server is listening on port " + PORT);
12.
13.            while (true) {
14.                Socket clientSocket = serverSocket.accept();
15.                System.out.println("Client connected: " + clientSocket.getInetAddress());
16.
17.                // Menerima file dari klien
18.                receiveFile(clientSocket);
19.            }
20.        } catch (IOException e) {
21.            e.printStackTrace();
22.        }
23.    }
24.
25.    private static void receiveFile(Socket clientSocket) {
26.        try {
27.            // Inisialisasi input stream untuk membaca data dari klien
28.            InputStream inputStream = clientSocket.getInputStream();
29.            ObjectInputStream objectInputStream = new ObjectInputStream(inputStream);
30.
31.            // Menerima informasi file (nama dan ukuran)
32.            String fileName = objectInputStream.readUTF();
33.            long fileSize = objectInputStream.readLong();
34.
35.            System.out.println("Receiving file: " + fileName + " (" + fileSize + " bytes)");
36.
37.            // Inisialisasi output stream untuk menulis data ke file
38.            FileOutputStream fileOutputStream = new FileOutputStream("server_" + fileName);
39.                                    BufferedOutputStream   bufferedOutputStream   =   new
BufferedOutputStream(fileOutputStream);
40.
41.            // Menerima dan menulis data file
42.            byte[] buffer = new byte[1024];
43.            int bytesRead;
44.            while ((bytesRead = inputStream.read(buffer)) != -1) {
45.                bufferedOutputStream.write(buffer, 0, bytesRead);
46.            }
47.
48.            // Menutup output stream dan socket
49.            bufferedOutputStream.close();
50.            clientSocket.close();
51.
52.            System.out.println("File received successfully.");
```

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

Klien (FileClient.java):

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

public class FileClient {
    public static void main(String[] args) {
        final String SERVER_IP = "localhost";
        final int SERVER_PORT = 12345;

        try {
            // Membaca file yang akan dikirim ke server
            String filePath = "path/to/your/file.txt";
            File file = new File(filePath);

            Socket clientSocket = new Socket(SERVER_IP, SERVER_PORT);
            System.out.println("Connected to server.");

            // Mengirim file ke server
            sendFile(file, clientSocket);

            // Menutup socket setelah selesai
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void sendFile(File file, Socket clientSocket) {
        try {
            // Inisialisasi output stream untuk mengirim data ke server
            OutputStream outputStream = clientSocket.getOutputStream();
            ObjectOutputStream objectOutputStream = new ObjectOutputStream(outputStream);

            // Mengirim informasi file (nama dan ukuran)
            objectOutputStream.writeUTF(file.getName());
            objectOutputStream.writeLong(file.length());
            objectOutputStream.flush();

            // Inisialisasi input stream untuk membaca data dari file
            FileInputStream fileInputStream = new FileInputStream(file);
          BufferedInputStream bufferedInputStream = new BufferedInputStream(fileInputStream);

            // Membaca dan mengirim data file
            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = bufferedInputStream.read(buffer)) != -1) {
                outputStream.write(buffer, 0, bytesRead);
            }

            // Menutup output stream dan file
```

```
            bufferedInputStream.close();
            clientSocket.shutdownOutput();

            System.out.println("File sent successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Cara Menjalankan:

1. Jalankan `FileServer` pada server.
2. Sesuaikan path file yang akan dikirim pada `FileClient` dan jalankan pada klien.
3. Aplikasi ini akan mengirimkan file dari klien ke server menggunakan model stream (TCP). Pastikan untuk menyesuaikan path file dan konfigurasi jaringan sesuai dengan kebutuhan Anda.

Contoh 2 :

Mengirim Pesan ke Server dan Disebarkan ke Semua Klien

ChatServer.java

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

public class ChatServer {
    private static final int PORT = 9876;

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

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

                // Menerima pesan dari klien
                serverSocket.receive(receivePacket);

                // Menampilkan pesan yang diterima
                String message = new String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("Received message from " + receivePacket.getAddress() + ": " + message);

                // Menyebarkan pesan ke semua klien yang terhubung (simulasi)
                broadcastMessage(serverSocket, receivePacket);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

```

```
32.         private static void broadcastMessage(DatagramSocket serverSocket, DatagramPacket receivePacket) throws IOException {
33.         // Simulasi: Menyebarkan pesan ke semua klien yang terhubung
34.         // Untuk implementasi yang lebih canggih, simpan daftar klien yang terhubung
35.         // dan kirim pesan ke setiap klien.
36.         // Simulasi di sini hanya menunjukkan konsep dasar.
37.         byte[] sendData = receivePacket.getData();
38.         for (int port = 10000; port <= 10002; port++) {
39.                             serverSocket.send(new DatagramPacket(sendData, sendData.length, receivePacket.getAddress(), port));
40.         }
41.     }
42. }
43. ```
44.
```

ChatClient.java

```
 1. // ChatClient.java
 2. import java.io.BufferedReader;
 3. import java.io.IOException;
 4. import java.io.InputStreamReader;
 5. import java.net.DatagramPacket;
 6. import java.net.DatagramSocket;
 7. import java.net.InetAddress;
 8.
 9. public class ChatClient {
10.     private static final String SERVER_IP = "localhost";
11.     private static final int SERVER_PORT = 9876;
12.
13.     public static void main(String[] args) {
14.         try {
15.             DatagramSocket clientSocket = new DatagramSocket();
16.
17.             // Memasukkan nama klien
18.             System.out.print("Enter your name: ");
19.             BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
20.             String name = reader.readLine();
21.
22.             // Mengirim pesan ke server
23.             sendMessage(clientSocket, name);
24.
25.             // Menutup socket setelah mengirim pesan
26.             clientSocket.close();
27.         } catch (IOException e) {
28.             e.printStackTrace();
29.         }
30.     }
31.
32.         private static void sendMessage(DatagramSocket clientSocket, String name) throws IOException {
33.         BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
34.
35.         while (true) {
36.             // Memasukkan pesan
37.             System.out.print("Enter your message (or type 'exit' to quit): ");
38.             String message = reader.readLine();
39.
40.             if ("exit".equalsIgnoreCase(message)) {
```

```
41.                 break; // Keluar dari loop jika pengguna mengetik 'exit'
42.             }
43.
44.             // Mengirim pesan ke server
45.             byte[] sendData = (name + ": " + message).getBytes();
46.                 DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
InetAddress.getByName(SERVER_IP), SERVER_PORT);
47.             clientSocket.send(sendPacket);
48.         }
49.     }
50. }
51. ```
52.
```

Catatan:

- Contoh di atas hanya simulasi sederhana. Dalam implementasi nyata, Anda perlu menyimpan daftar klien yang terhubung dan mengirim pesan ke setiap klien yang terkait.
- Pastikan untuk memeriksa kondisi koneksi yang terputus dan menangani eksepsi dengan baik.
- Jalankan beberapa instansi `ChatClient` untuk melihat bagaimana pesan disebarkan ke semua klien yang terhubung.

Latihan :

Latihan: Memperbaiki Program Pemrograman Jaringan

Tujuan Latihan:

Latihan ini bertujuan untuk memberikan pemahaman tentang kesalahan umum yang dapat terjadi dalam program pemrograman jaringan menggunakan model datagram dan stream. Mahasiswa diharapkan dapat mengidentifikasi kesalahan penulisan atau logika dalam program dan memperbaikinya.

Instruksi:

1. Identifikasi Kesalahan:
   - Berikan beberapa contoh program pemrograman jaringan menggunakan model datagram dan stream yang mengandung kesalahan penulisan atau logika.
   - Sebutkan kesalahan-kesalahan yang mungkin terjadi, seperti kesalahan sintaks, kesalahan logika, atau kurangnya penanganan eksepsi.
2. Perbaiki Program:
   - Perbaiki program-program tersebut dengan memperbaiki kesalahan yang telah diidentifikasi.
   - Pastikan program dapat berjalan tanpa kesalahan dan sesuai dengan kebutuhan fungsional yang diinginkan.

3. Dokumentasikan Perubahan:

- Dokumentasikan perubahan yang Anda lakukan pada setiap program.
- Jelaskan alasan di balik setiap perubahan yang Anda buat.

4. Uji Coba:

- Jalankan program yang telah diperbaiki untuk memastikan bahwa mereka berfungsi dengan baik dan sesuai dengan ekspektasi.

- Uji coba dapat melibatkan simulasi pengiriman data, penggunaan klien dan server, atau skenario khusus lainnya yang sesuai dengan program yang diperbaiki.

Kesalahan pada Datagram (UDP) - Chat Server

```
1. // Program with errors
2. import java.io.IOException;
3. import java.net.DatagramPacket;
4. import java.net.DatagramSocket;
5.
6. public class ChatServer {
7.     private static final int PORT = 9876;
8.
9.     public static void main(String[] args) {
10.         try {
11.             DatagramSocket serverSocket = new DatagramSocket(PORT);
12.             System.out.println("Server is listening on port " + PORT);
13.
14.             while (true) {
15.                 byte[] receiveData = new byte[1024];
16.                             DatagramPacket  receivePacket  =  new  DatagramPacket(receiveData,
receiveData.length);
17.
18.                 // Menerima pesan dari klien
19.                 serverSocket.receive(receivePacket);
20.
21.                 // Menampilkan pesan yang diterima
22.                                 String  message  =  new  String(receivePacket.getData(),  0,
receivePacket.getLength());
23.                 System.out.println("Received message: " + message);
24.             }
25.         } catch (IOException e) {
26.             e.printStackTrace();
27.         }
28.     }
29. }
30. ```
31.
```

Kesalahan pada Datagram (UDP) - Chat Client

```
// Program with errors
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class ChatClient {
    private static final String SERVER_IP = "localhost";
    private static final int SERVER_PORT = 9876;

    public static void main(String[] args) {
        try {
            DatagramSocket clientSocket = new DatagramSocket();

            // Memasukkan nama klien
            System.out.print("Enter your name: ");
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
```

```
            String name = reader.readLine();

            // Mengirim pesan ke server
            sendMessage(clientSocket, name);

            // Menutup socket setelah mengirim pesan
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void sendMessage(DatagramSocket clientSocket, String name) throws IOException
{
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        while (true) {
            // Memasukkan pesan
            System.out.print("Enter your message (or type 'exit' to quit): ");
            String message = reader.readLine();

            if ("exit".equalsIgnoreCase(message)) {
                break; // Keluar dari loop jika pengguna mengetik 'exit'
            }

            // Mengirim pesan ke server
            byte[] sendData = (name + ": " + message).getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
InetAddress.getByName(SERVER_IP), SERVER_PORT);
            clientSocket.send(sendPacket);
        }
    }
}
```

Catatan:

1. Kesalahan yang diidentifikasi bisa berupa kesalahan sintaks, kurangnya penanganan eksepsi, atau kesalahan logika dalam program.
2. Perbaikan dilakukan untuk memastikan program dapat berjalan dengan baik dan memenuhi kebutuhan fungsional yang diinginkan.
3. Lakukan langkah-langkah serupa untuk program dengan model stream (TCP) atau contoh lainnya.

Tugas Rumah :

Tujuan Tugas:

Tugas ini bertujuan untuk memberikan pemahaman dan pengalaman praktis dalam mengimplementasikan pemrograman jaringan menggunakan model stream (TCP) dan datagram (UDP).

Instruksi:

Bagian 1: Model Stream (TCP)

1. Server TCP (FileServerTCP.java):
   - Implementasikan server TCP yang dapat menerima file dari klien dan menyimpannya di server.
   - Server harus dapat menangani multiple klien secara bersamaan.
   - Berikan informasi kepada klien setiap kali file berhasil diterima.
3. Klien TCP (FileClientTCP.java):
   - Implementasikan klien TCP yang dapat mengirim file ke server.
   - Klien harus dapat menangani pengiriman file dari path yang ditentukan oleh pengguna.
4. Uji Coba:
   - Jalankan beberapa klien sekaligus untuk menguji kemampuan server menangani multiple klien.
   - Kirimkan file dari beberapa klien ke server secara bersamaan dan periksa apakah file tersebut berhasil diterima.

Bagian 2: Model Datagram (UDP)

1. Server UDP (ChatServerUDP.java):
   - Implementasikan server UDP yang dapat menerima pesan dari klien dan menyebarkannya ke semua klien yang terhubung.
   - Server harus dapat menangani multiple klien secara bersamaan.
   - Pesan yang diterima oleh server harus mencakup informasi pengirim dan kontennya.
2. Klien UDP (ChatClientUDP.java):
   - Implementasikan klien UDP yang dapat mengirim pesan ke server.
   - Klien harus dapat menangani pengiriman pesan ke server.
3. Uji Coba:
   - Jalankan beberapa instansi klien sekaligus untuk menguji kemampuan server menangani multiple klien.
   - Kirimkan pesan dari salah satu klien dan periksa apakah pesan tersebut disebarkan ke semua klien yang terhubung.

Pengumpulan Tugas:

1. Kumpulkan source code dari implementasi server dan klien (TCP dan UDP). Dan pastikan sudah di kirimkan juga ke Github atau Gitlab
2. Sertakan dokumentasi singkat yang menjelaskan bagaimana program berfungsi dan hasil uji coba yang dilakukan.