LAPORAN PRAKTIKUM

953633205

PRAKTIKUM SISTEM TERDISTRIBUSI

MODUL: 3

"PEMROGRAMAN SOCKET DENGAN KONSEP"

"MULTI-THREADING"

Eska Smara Nofiansi	M0516018
Fabian Gilanggi	M0517014
Henny Nurcahyaning Tyas	M0517022

KELOMPOK : 6

HARI : JUMAT

TANGGAL : 17 APRIL 2020

WAKTU : 12.45 – 15.00 WIB

ASISTEN : HAIDAR HENDRI SETYAWAN M0516023

TAUFIQ ODHI DWI PUTRA M0516043

PROGRAM STUDI INFORMATIKA

UNIVERSITAS SEBELAS MARET

2020

Modul 2

PEMROGRAMAN SOCKET DENGAN KONSEP MULTI-THREADING

Eska Smara Nofiansi M0516018, Fabian Gilanggi M0517014, Henny Nurcahyaning Tyas M0517022/ Kelompok 6/ Jumat, 17 April 2020

fgilanggi@student.uns.ac.id, hnytyas72@student.uns.ac.id, rarasmara@student.uns.ac.id

Asisten : Haidar Hendri Setyawan
Taufiq Odhi Dwi Putra

Abstraksi – Pengertian socket adalah interface pada jaringan yang menjadi titik komunikasi antarmesin pada Internet Protocol. Jika tidak terjadi komunikasi maka tidak ada pertukaran data dan informasi jaringan. Oleh karena itu dilakukan pemrograman untuk menggunakan mengatur lalu lintas data pada socket tersebut. Pada percobaan kali ini menggunakan konsep Multi-threading, dimana koneksi ke server akan diciptakan suatu thread baru.

Kata Kunci - Socket, Multi-Threading, Komunikasi Client-Server.

I. PENDAHULUAN

Socket adalah sebuah abstaksi perangkat lunak yang digunakan suatu "terminal" dari suatu hubugan antara dua mesin atau proses yang saling berinterkoneksi. Socket dapat melakukan beberapa operasi yang meliputi koneksi ke main remote, mengirim data (Write), menerima data (Read), menutup koneksi (Close), bind to port, listen pada data yang masuk dan menerima koneksi dari mesin remote pada port tertentu.

Multi-Threading adalah sebuah mekanisme dimana dalam suatu proses ada beberapa thread yang mengerjakan tugasnya masing-masing dalam waktu yang bersamaan. Dengan Multi-threading proses dapat dilakukan lebih cepat, dimana thread ini bekerja secara pararel.

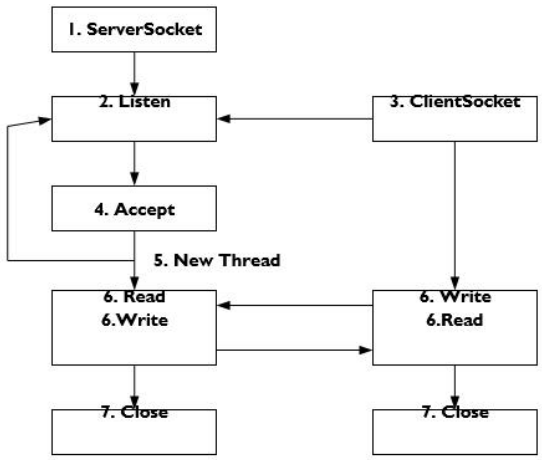
Dalam laporan ini kami membuat socket programming client server TCP dan UDP dengan konsep multi-threading untuk proses penjumlahan. Socket server yang didalamnya terdapat sustu method untuk melakukan penjumlahan suatu data array bertipe integer, dengan mengambil indeks maksimal 5x. Pada socket client mengirimkan data yang akan dijumlahkan, dan socket server melakukan proses penjumlahan dan mengirimkan hasilnya ke socket lain.

II. DASAR TEORI

2.1 Multi-Threading

Penggunaan Multi-Threading untuk pemrogaman socket sangat penting, karena suatu server harus melayani banyak client. Aplikasi socket untuk server hanya dapat melayani client tunggal, setelah itu server berhenti. Dengan menggunakan konsep multi-threading maka setiap client koneksi

ke server akan diciptakan thread baru. Sehingga server selalu running dan siap menerima request dari client (seperti program web server apache).



Gambar 1. Diagram block

2.2 Java Thread

Ketika pembuatan object baru class *java.lang.Thread* memberikan penanganan control dan sinkronisasi eksekusinya. Sebuah object yang diperlukan sebuah thread harus menerapkan interface *java.lang.runnable*. Dalam interface ini menyediakan suatu method kunci run().

```
public interface Runnable{
Abstract public void run();
}
```

Sebuah thread dimulai siklus hidupnya dengan memanggil method run(). Untuk membuat sebuah class yang akan bertindak sebagai sebuah thread, didefinisikan dengan bentuk sebagai berikut :

```
Class namaClass implements

Runnable {
Void run(){
//implementasi kode.....
}
```

2.3 *Pengontrolan Thread*

Method *start()* digunakan untuk memulai sebuah thread dieksekusi. Method *stop()* digunakan untuk menghentikan eksekusi suatu thread. Method *suspend()* digunakan untuk menghentikan sementara waktu (pause) untuk kemudian dapat dilanjutkan kembali dengan menggunakan method *resume()*. Jika beban yang harus dikerjakan oleh thread berat/mahal, maka dapat digunakan method *suspend()* dan *resume()*. Method static *sleep(n_msec)* digunakan untuk menghentikan thread yang active selama n millisecond.

III. ALAT DAN LANGKAH PERCOBAAN

Alat yang digunakan adalah sebagai berikut:

1. Laptop
2. IDE Java Eclipse

Langkah dalam melakukan pemrograman socket dengan konsep Multi-Threading antara lain

a) Pembuatan Object baru
b) Membuat instance object dari class (implementasi interface runnable)
c) Buat object class java.lang. Thread dengan parameter konstructor
d) Panggil Method Start

e) Menciptakan sebuah output data stream

f) Pengontrolan Thread, start(), stop(), suspend(), resume(), dll.

g) Close socket

IV. HASIL DAN ANALISIS PERCOBAAN

Program terdiri dari 2 program yaitu UTP dan UDP

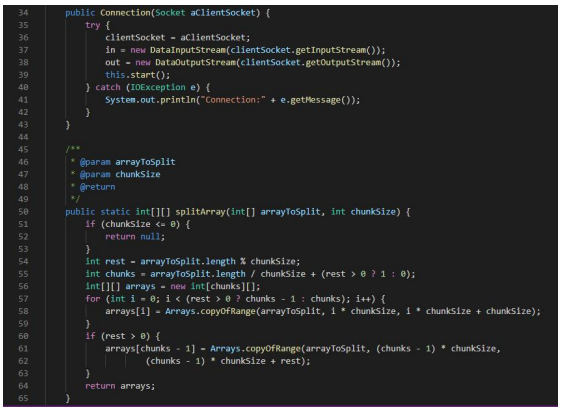
1. UTP

TCP adalah bagian inti penting dari Internet Protocol (IP) yang kemudian sering disebut sebagai TCP/IP. TCP berada di transport layer yang memiliki urutan rapi dan menyediakan komunikasi yang dapat diandalkan.

1.1 TCP Server

Pada Program yang dibuat TCP Server merupakan program yang dibuat untuk server pada socket. Server UTP dirun terlebih dahulu sebelum menjalankan UTP Client.

```
C: > Users > Fabian > Downloads > 03 > TcpServer.java > Connection > run()
package psister;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Arrays;

class BatchSummation implements Runnable {
    Thread thread;
    int[] summ;
    int result = 0;

    BatchSummation(int[] arr) {
        this.summ = arr;

        this.thread = new Thread(this);

        this.thread.start();
    }

    public void run() {
        this.result = Arrays.stream(this.summ).sum();
    }
}

class Connection extends Thread {
    DataInputStream in;
    DataOutputStream out;
    Socket clientSocket;
```



```
public Connection(Socket aClientSocket) {
    try {
        clientSocket = aClientSocket;
        in = new DataInputStream(clientSocket.getInputStream());
        out = new DataOutputStream(clientSocket.getOutputStream());
        this.start();
    } catch (IOException e) {
        System.out.println("Connection:" + e.getMessage());
    }
}

/**
 * @param arrayToSplit
 * @param chunkSize
 * @return
 */
public static int[][] splitArray(int[] arrayToSplit, int chunkSize) {
    if (chunkSize <= 0) {
        return null;
    }
    int rest = arrayToSplit.length % chunkSize;
    int chunks = arrayToSplit.length / chunkSize + (rest > 0 ? 1 : 0);
    int[][] arrays = new int[chunks][];
    for (int i = 0; i < (rest > 0 ? chunks - 1 : chunks); i++) {
        arrays[i] = Arrays.copyOfRange(arrayToSplit, i * chunkSize, i * chunkSize + chunkSize);
    }
    if (rest > 0) {
        arrays[chunks - 1] = Arrays.copyOfRange(arrayToSplit, (chunks - 1) * chunkSize,
                (chunks - 1) * chunkSize + rest);
    }
    return arrays;
}
```

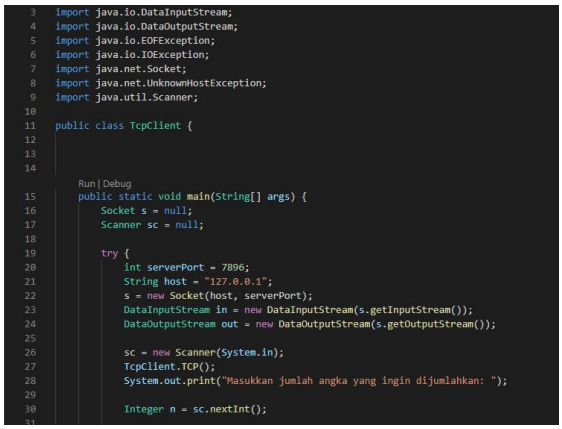
```
public void run() {
    try {
        String data = in.readUTF();
        String[] splitted = data.split(",");
        int[] number = new int[splitted.length];
        Arrays.fill(number, 0);
        for (int i = 0; i < splitted.length; i++) {
            try {
                number[i] = Integer.valueOf(splitted[i].trim());
            } catch (NumberFormatException e) {
                number[i] = 0;
            }
        }

        int[][] chuncks = Connection.splitArray(number, 5);
        BatchSummation[] threads = new BatchSummation[chuncks.length];
        for (int i = 0; i < chuncks.length; i++) {
            threads[i] = new BatchSummation(chuncks[i]);
            System.out.println("Spawning new thread with id:"
            + threads[i].thread.getId() + " for number: " + Arrays.toString(chuncks[i]));
        }

        for (BatchSummation th: threads) {
            th.thread.join();
        }

        int result = 0;
        for (BatchSummation th : threads) {
            result += th.result;
```

```
            out.writeUTF(Integer.toString(result));
        } catch (EOFException e) {
            System.out.println("EOF:" + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO:" + e.getMessage());
        } catch (InterruptedException e) {
            System.out.println("InterruptedException:" + e.getMessage());
        } finally {
            try {
                clientSocket.close();
            } catch (IOException e) {
            }
        }
    }
}

public class TcpServer {
    Run | Debug
    public static void main(String[] args) {
        try {
            int serverPort = 7896;
            ServerSocket listenSocket = new ServerSocket(serverPort);
            System.out.println("Server sedang running.....");
            while (true) {
                Socket clientSocket = listenSocket.accept();
                Connection c = new Connection(clientSocket);
            }
        } catch (IOException e) {
            System.out.println("Listen :" + e.getMessage());
        }
```

Gambar 2.Source code UTP Server

1.2 TCP Client

UTP Client merupakan class yang digunakan untuk melakukan request ke server, kemudian dari request tersebut server akan merespon request client.



```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class TcpClient {



    Run | Debug
    public static void main(String[] args) {
        Socket s = null;
        Scanner sc = null;

        try {
            int serverPort = 7896;
            String host = "127.0.0.1";
            s = new Socket(host, serverPort);
            DataInputStream in = new DataInputStream(s.getInputStream());
            DataOutputStream out = new DataOutputStream(s.getOutputStream());

            sc = new Scanner(System.in);
            TcpClient.TCP();
            System.out.print("Masukkan jumlah angka yang ingin dijumlahkan: ");

            Integer n = sc.nextInt();
```

```
            String[] number = new String[n];

            sc.nextLine();

            for (int i = 0; i < n; i++) {
                System.out.print("Masukkan angka ke-" + (i + 1) + ": ");

                String next = sc.nextLine();

                try {
                    Integer.parseInt(next);

                    number[i] = next;
                } catch (Exception e) {
                    number[i] = "0";
                }
            }

            String input = String.join(",", number);

            out.writeUTF(input);
            String data = in.readUTF();
            System.out.println("Received: " + data);
        } catch (UnknownHostException e) {
            System.out.println("Sock:" + e.getMessage());
        } catch (EOFException e) {
            System.out.println("EOF:" + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO:" + e.getMessage());
        } finally {
            if (sc != null)
                sc.close();

            if (s != null)
                try {
                    s.close();
                } catch (IOException e) {
                    System.out.println("close:" + e.getMessage());
                }

        }
    }
    public static void TCP() {
        System.out.println("Menjalankan TCP.....");
    }
}
```

Gambar 3. Source code UTP Client

1.3 Output

Berikut adalah output ketika Server dan Client dijalankan dan request ke server.

```
62858@DESKTOP-26BCPKC MINGW64 /f/Sister/Praktikum/Praktikum_03/Tuga
$  C:\\Users\\62858\\.vscode\\extensions\\vscjava.vscode-java-debug
oaming\\Code\\User\\workspaceStorage\\7aa4d8357f40192ef0cb6a3b5aa27
============ Mulai Program (TCP) ============
Masukkan jumlah angka yang ingin dijumlahkan: 3
Masukkan angka ke-1: 13
Masukkan angka ke-2: 33
Masukkan angka ke-3: 1
Received: 47

62858@DESKTOP-26BCPKC MINGW64 /f/Sister/Praktikum/Praktikum_03/Tuga
$
```

Gambar 4. Hasil Output

2. UDP

UDP adalah kependekan dari User Datagram Protocol metupakan bagian dari internet protocol. UDP adalah jenis protokol internet yang memungkinkan sebuah perangkat lunak pada komputer bisa mengirimkan pesan ke komputer lain melalui jaringan tanpa perlu ada komunikasi awal.

2.1 UDP Server

Pada Program yang dibuat UDP Server merupakan program yang dibuat untuk server pada socket. Server UDP dirun terlebih dahulu sebelum menjalankan UDP Client.

C: > Users > Fabian > Downloads > 03 > UdpServer.java > UdpServer

```java
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.Socket;
import java.util.Arrays;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.ServerSocket;

class BatchSummationUDP implements Runnable {
    Thread thread;
    int[] summ;
    int result = 0;

    BatchSummationUDP(int[] arr) {
        this.summ = arr;
        this.thread = new Thread(this);
        this.thread.start();
    }
    public void run() {
        this.result = Arrays.stream(this.summ).sum();
    }
}

class ConnectionUDP extends Thread {
    DatagramSocket aSocket;
    DatagramPacket request;

    public ConnectionUDP(DatagramSocket aSocket, DatagramPacket request) {
        this.aSocket = aSocket;
        this.request = request;

        this.start();
    }

    /**
     * @param arrayToSplit
     * @param chunkSize
     * @return
     */
    public static int[][] splitArray(int[] arrayToSplit, int chunkSize) {
        if (chunkSize <= 0) {
            return null;
        }
        int rest = arrayToSplit.length % chunkSize;
        int chunks = arrayToSplit.length / chunkSize + (rest > 0 ? 1 : 0);
        int[][] arrays = new int[chunks][];
        for (int i = 0; i < (rest > 0 ? chunks - 1 : chunks); i++) {
            arrays[i] = Arrays.copyOfRange(arrayToSplit, i * chunkSize, i * chunkSize + chunkSize);
        }
        if (rest > 0) {
            arrays[chunks - 1] = Arrays.copyOfRange(arrayToSplit, (chunks - 1) * chunkSize,
                    (chunks - 1) * chunkSize + rest);
        }
        return arrays;
    }
    public void run() {
        try {
            String data = new String(request.getData());
            String[] splitted = data.split(",");
            int[] number = new int[splitted.length];
            Arrays.fill(number, 0);
            for (int i = 0; i < splitted.length; i++) {
                try {
                    number[i] = Integer.valueOf(splitted[i].trim());
                } catch (NumberFormatException e) {
                    number[i] = 0;
                }
            }

            int[][] chuncks = ConnectionUDP.splitArray(number, 5);
            BatchSummationUDP[] threads = new BatchSummationUDP[chuncks.length];
            for (int i = 0; i < chuncks.length; i++) {
                threads[i] = new BatchSummationUDP(chuncks[i]);
                System.out.println("Spawning new thread with id:" + threads[i].thread.getId() + " for number: "
                        + Arrays.toString(chuncks[i]));
            }
            for (BatchSummationUDP th : threads) {
                th.thread.join();
            }

            int result = 0;
            for (BatchSummationUDP th : threads) {
                result += th.result;
            }
            byte[] response = Integer.toString(result).getBytes();

            DatagramPacket reply = new DatagramPacket(
                    response, response.length, request.getAddress(),
                    request.getPort());

            aSocket.send(reply);
        } catch (EOFException e) {
            System.out.println("EOF:" + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO:" + e.getMessage());
        } catch (InterruptedException e) {
            System.out.println("InterruptedException:" + e.getMessage());
        } finally {
            aSocket.close();
        }
    }
}
public class UdpServer {
    Run | Debug
    public static void main(String[] args) {
        DatagramSocket aSocket = null;

        try {
            int serverPort = 7896;

            aSocket = new DatagramSocket(serverPort);

            byte[] buffer = new byte[256];

            System.out.println("Server sedang running.....");
            while (true) {
                DatagramPacket request = new DatagramPacket(buffer, buffer.length);

                aSocket.receive(request);

                ConnectionUDP c = new ConnectionUDP(aSocket, request);
            }
        } catch (IOException e) {
            System.out.println("Listen: " + e.getMessage());
        }
    }
}
```

Gambar 6.Source code UDPServer

2.2 UDP Client

UDP Client merupakan class yang digunakan untuk melakukan request ke server, kemudian dari request tersebut server akan merespon request client.

```java
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class UdpClient {

    Run | Debug
    public static void main(String[] args) {
        DatagramSocket s = null;
        Scanner sc = null;

        try {
            int serverPort = 7896;
            String host = "127.0.0.1";

            s = new DatagramSocket();

            InetAddress aHost = InetAddress.getByName(host);

            sc = new Scanner(System.in);
            UdpClient.UDP();
            System.out.print("Masukkan jumlah angka yang ingin dijumlahkan: ");

            Integer n = sc.nextInt();
```

```
        Integer n = sc.nextInt();

        String[] number = new String[n];

        sc.nextLine();

        for (int i = 0; i < n; i++) {
            System.out.print("Masukkan angka ke-" + (i + 1) + ": ");

            String next = sc.nextLine();

            try {
                Integer.parseInt(next);

                number[i] = next;
            } catch (Exception e) {
                number[i] = "0";
            }
        }

        String input = String.join(",", number);
        byte[] m = input.getBytes();

        DatagramPacket request = new DatagramPacket(m, input.length(), aHost, serverPort);

        s.send(request);

        byte[] buffer = new byte[m.length];

        DatagramPacket reply = new DatagramPacket(buffer, buffer.length);

        s.receive(reply);
            s.receive(reply);

            System.out.println("Received: " + new String(reply.getData()));
        } catch (UnknownHostException e) {
            System.out.println("Sock:" + e.getMessage());
        } catch (EOFException e) {
            System.out.println("EOF:" + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO:" + e.getMessage());
        } finally {
            if (sc != null)
                sc.close();

            if (s != null)
                s.close();

        }

    }

    public static void UDP() {
        System.out.println("Menjalankan UDP.....");
    }
}
```

Gambar 6.Source code UDP Client

2.3 Output

```
62858@DESKTOP-26BCPKC MINGW64 /f/Sister/Praktikum/Praktikum_03/Tugas/
$  C:\\Users\\62858\\.vscode\\extensions\\vscjava.vscode-java-debug-0
oaming\\Code\\User\\workspaceStorage\\7aa4d8357f40192ef0cb6a3b5aa27fe
============ Mulai Program (UDP) ============
Masukkan jumlah angka yang ingin dijumlahkan: 4
Masukkan angka ke-1: 5
Masukkan angka ke-2: 12
Masukkan angka ke-3: 55
Masukkan angka ke-4: 33
Received: 105
```

V. KESIMPULAN

Dalam pemograman socket dengan konsep Multi-Threading server dapat melayani banyak client. Setiap client koneksi ke server akan diciptakan suatu thread baru. Sehingga server selalu running dan siap menerima request dari client (seperti program web server apache).

VI. DAFTARPUSTAKA

Winarno. 2020. *Modul Praktikum Sistem Terdistribusi"*.

Oracle. 2017. "*All About Sockets"*. https://docs.oracle.com/javase/tutorial/networking/sockets/

[**Eska Smara Nofiansi]** Mahasiswa Informatika Angkatan 2016

[Fabian Gilanggi] Mahasiswa Informatika Angkatan 2017

[Henny Nurcahyaning Tyas] Mahasiswa Informatika Angkatan 2017