Software Practice 1 - Socket

- Terms of socket programming
- Socket
- Implementation (TCP, UDP)
- Socket with multithread
- Serialization

Lab practice

Prof. Joonwon Lee
T.A. Jaehyun Song
Jongseok Kim
T.A. Sujin Oh
Junseong Lee

(42)
(43)
Terms of Network

- **Packet**
  - A formatted unit of data carried by a packet-switched network

- **Acknowledgement (ack)**
  - A signal passed between communicating processes or computers to signify acknowledgement, or receipt of response, as a part of a communications protocol
Terms of Network

- **Internet Protocol (IP)**
  - IP has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet header.
  - Example of network address
    - “111.222.333.444” or “skku.edu” or “localhost”

- **Port**
  - An endpoint of communication in an operating system.
  - While the term is used for receiver connectors on hardware devices, in software it is a logical construct that identifies a specific process or a type of network service.
Terms of Network

- **Transmission Control Protocol (TCP)**
  - TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating by an IP network

- **User Datagram Protocol (UDP)**
  - UDP uses a simple connectionless transmission model with a minimum of protocol mechanism
  - There is no guarantee of reliability, delivery, ordering, or duplicate protection
Terms of Network

- **Server**
  - A computer program or a device that provides functionality for other programs or devices, called clients

- **Client**
  - A piece of computer hardware or software that accesses a service made available by a server
## TCP versus UDP

<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td>Connection oriented protocol</td>
<td>Connection-less protocol</td>
</tr>
<tr>
<td>I/O stream</td>
<td>Connection in byte stream</td>
<td>Connection in message stream</td>
</tr>
<tr>
<td>Comm. type</td>
<td>Not support multicasting or broadcasting</td>
<td>Support both</td>
</tr>
<tr>
<td>Transport reliability</td>
<td>Provide error control and flow control</td>
<td>Not provide any of them</td>
</tr>
<tr>
<td>Packet is called as</td>
<td>Segment</td>
<td>Datagram</td>
</tr>
</tbody>
</table>
TCP versus UDP

TCP (connection oriented)

Error!
Data is corrupted, please resend.

UDP (connectionless)

Not all data is present.
Do not resend.
Socket

- An internal endpoint for sending or receiving data at a single node in a computer network
- Concretely, it is a representation of this endpoint in networking software, such as an entry in a table, and is a form of system resource.
Socket programming

- TCP version

- 5 steps for communication
  1. Create server
  2. Create client
  3. Establish connection
  4. Data transport
  5. Close connection

- Examples are written without try-catch
Process of TCP

Client

Server

Syn (sequence = n)

Ack (ack = n+1)

Data (sequence = m)

Ack (sequence = m+1)

TIME
Java class for TCP

- **ServerSocket**
  - [http://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html?is-external=true](http://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html?is-external=true)

- **Socket**
  - [https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html](https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html)
TCP server side

ServerSocket ss = new ServerSocket(5000);

while (true) {
    Socket soc = ss.accept();
    OutputStream out = soc.getOutputStream();
    DataOutputStream dos = new DataOutputStream(out);

    dos.writeUTF("message from server");

    dos.close();
    soc.close();
}
TCP client side

String serverIP = "localhost";
Socket soc = new Socket(serverIP, 5000);

InputStream in = soc.getInputStream();
DataInputStream dis = new DataInputStream (in);

System.out.println (dis.readUTF ());

dis.close();
soc.close();
Socket programming

- UDP version

- 4 steps for communication
  1. Create server
  2. Create client
  3. Request data
  4. Receive data

- Examples are written without try-catch
Java class for UDP

- **DatagramSocket**
  - [http://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html?is-external=true](http://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html?is-external=true)

- **DatagramPacket**
  - [http://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html](http://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html)

- **MulticastSocket**
  - [http://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html](http://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html)
Types of transport

- **Unicast**
  - One-to-one association between a sender and destination
  - Each destination address uniquely identifies a single receiver endpoint
DatagramSocket sender = new DatagramSocket (5000);
byte[] receiveData = new byte[1024];
byte[] sendData = new byte[1024];
String message = "get you";

while (true) {
    DatagramPacket rp = new DatagramPacket (receiveData, receiveData.length);
    sender.receive (rp);
    String data = new String (rp.getData ());

    InetAddress ip = rp.getAddress ();
    int port = rp.getPort ();
    sendData = message.getBytes();
    DatagramPacket sp = new DatagramPacket (sendData, sendData.length, ip, port);
    sender.send (sp);
}
DatagramSocket receiver = new DatagramSocket ();
InetAddress ip = InetAddress.getByName("localhost");

BufferedReader in = new BufferedReader (new InputStreamReader (System.in));
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
sendData = in.readLine ().getBytes();

DatagramPacket sp = new DatagramPacket (sendData, sendData.length, ip, 5000);
receiver.send (sp);

DatagramPacket rp = new DatagramPacket (receiveData, receiveData.length);
String rData = new String (rp.getData ());

receiver.close ();
Types of transport

- **Broadcast**
  - One-to-all association
  - A single datagram from one sender is routed to all of the possibly multiple endpoints associated with the broadcast address
Types of transport

- **Multicast**
  - One-to-many-of-many association
  - Differs from broadcast in that the destination address designates a subset not necessarily all, of the accessible nodes
  - [https://examples.javacodegeeks.com/core-java/net/multicastsocket-net/java-net-multicastsocket-example/](https://examples.javacodegeeks.com/core-java/net/multicastsocket-net/java-net-multicastsocket-example/)
Socket with multithread

- **Socket server of background thread**
  - Thread of socket server cannot do anything except for waiting connection
  - Therefore, it has to be implemented with multithread to concurrently do its all jobs

- **Deal with comm. channels for many clients**
  - Even when the multiple clients request to connect with server simultaneously, it is also necessary to deal with multithread
Background socket thread

- 3 components are necessary
  - Background server class (ServerThread.java)
  - Foreground main class (MainThread.java)
  - Client class (same with previous TCP client)
public class ServerThread extends Thread {
    private String name = null;
    private static SimpleDateFormat sdfDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:SSS");
    private static String getLog (String msg) {
        return "[" + sdfDate.format(new Date ()) + "] Server thread: " + msg;
    }

    public ServerThread () {
        this.name = "ServerThread";
    }

    public void run () {
        ServerSocket ss = null;
        try {
            ss = new ServerSocket(5000);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        while (true) {
            Socket soc = null;
            OutputStream out = null;
            try {
                soc = ss.accept();
                System.out.println(ServerThread.getLog("new connection arrived"));
                out = soc.getOutputStream();
                DataOutputStream dos = new DataOutputStream (out);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
ServerThread.java

dos.writeUTF("message from server");

dos.close();
soc.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} finally {
    try {
        soc.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
public classMainThread{
    private static SimpleDateFormat sdfDate = new SimpleDateFormat("yyy-MM-dd HH:mm:SSS");
    private static String getLog(String msg) {
        return "[" + sdfDate.format(new Date ()) + "] Main thread: " + msg;
    }

    public static void main(String[] args) {
        Thread t = new ServerThread();
        t.start();
        System.out.println(getLog("server thread started"));
        boolean flag = true;
        while (flag) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println(getLog("server still alive"));
        }
    }
}
Multiple channels

- 3 components are necessary
  - Channel management class (Server.java)
  - Communication server class (CommThread.java)
  - Client class (Clients.java)
public class Server {
    private static ArrayList<Thread> arr = new ArrayList<Thread>();
    private static SimpleDateFormat sdfDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:SSS");

    public static String getLog(String msg) {
        return "[" + sdfDate.format(new Date()) + "] Server thread: " + msg;
    }

    public static void main(String[] args) {
        ServerSocket ss = null;
        int id = 0;
        try {
            id = 0;
            ss = new ServerSocket(5000);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println("server is ready");
Server.java

```java
while (true) {
    try {
        Socket soc = ss.accept();
        System.out.println(Server.getLog("new connection arrived"));
        Thread t = new CommThread(soc, id++);
        t.start();
        arr.add(t);
        Iterator<Thread> iter = arr.iterator();
        while (iter.hasNext()) {
            t = iter.next();
            if (!t.isAlive()) {
                iter.remove();
            }
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```
public class CommThread extends Thread {
    private Socket soc;
    private int id;
    public CommThread(Socket soc, int id) {
        this.soc = soc;
        this.id = id;
    }

    public void run () {
        try {
            OutputStream os = soc.getOutputStream ();
            DataOutputStream dos = new DataOutputStream (os);

            dos.writeUTF("message from server (" + id + ")");
            System.out.println(Server.getLog("message is sent (" + id + ")");

            dos.close ();
            this.soc.close ();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
class Test extends Thread {
    public void run () {
        try {
            Socket soc = new Socket("localhost", 5000);
            DataInputStream dis = new DataInputStream(soc.getInputStream());

            System.out.println(dis.readUTF());
            dis.close();
            soc.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

public class Clients {
    public static void main (String[] args) {
        for (int i = 0; i < 100; i++) {
            new Test().start();
        }
    }
}
Serialization

- The process of translating data structures or object state into a format that can be stored or transmitted and reconstructed later

- Serialization in Java
  - public interface serializable (in java.io.Serializable)
  - Classes that do not implement this interface will not have any of their state serialized or deserialized
public class Employee implements java.io.Serializable {
    public String name;
    public String address;
    public transient int SSN;
    public int number;

    public void mailCheck() {
        System.out.println("Mailing a check to " + name + " " + address);
    }
}
Example of serialization

```java
public class SerializeDemo {

    public static void main(String[] args) {
        Employee e = new Employee();
        e.name = "Reyan Ali";
        e.address = "Phokka Kuan, Ambehta Peer";
        e.SSN = 11122333;
        e.number = 101;

        try {
            FileOutputStream fileOut = new FileOutputStream("/tmp/employee.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(e);
            out.close();
            fileOut.close();
            System.out.printf("Serialized data is saved in /tmp/employee.ser");
        } catch (IOException i) {
            i.printStackTrace();
        }
    }

```
public class DeserializeDemo {

    public static void main(String[] args) {
        Employee e = null;
        try {
            FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            e = (Employee) in.readObject();
            in.close();
            fileIn.close();
        } catch (IOException i) {
            i.printStackTrace();
            return;
        } catch (ClassNotFoundException c) {
            System.out.println("Employee class not found");
            c.printStackTrace();
            return;
        }

        System.out.println("Deserialized Employee...");
        System.out.println("Name: " + e.name);
        System.out.println("Address: " + e.address);
        System.out.println("SSN: " + e.SSN);
        System.out.println("Number: " + e.number);
    }
}
[Lab – Practice]

- Multiple channel + Serialization
- Make 4 classes (Server, CommThread, Client, Student)
- Establish connection between two programs in a single node with port number 5000
[Lab – Practice]

- **Server**
  - If a client connects, it starts a new thread of CommThread and adds that to arraylist.
  - It prints each status (Start, Stop)
  - If it have processed 10 clients, it will stop.

- **CommThread**
  - Receive and print object information.

- **Client**
  - Send and print object information.

- **Student**
  - Object to be exchanged, and it has a num variable
Example of Output

Output:

Server Start
<input Name>: send0
<input Name>: send1
...
<input Name>: send9
Server Stop
Receive: student0
Receive: student1
...

Upload to i-Campus

- Compress your all java files to zip file
- File name: studentID_lab9.zip

Due date

- Today 23:59:59
  - Class 42 (05/28 Monday)
  - Penalty: -10% of each lab score per one day