Systems Software

COMP20081

Lecture 14 – Java Sockets

Dr Michalis Mavrovouniotis

School of Science and Technology

ERD 200

Office Hours: Thursday 12:00-14:00

Recall and Lecture Overview

- Recall
  - TCP/IP Reference Model
  - TCP/IP and UDP/IP protocols
- Overview
  - Concept of Java Socket
  - Stream Sockets
  - Datagram Sockets

Client-Server Model

- It is a structure used in distributed applications
- A server is a program that provides a number of services
- A client will request for services and wait for a response
  - Server receives a request from a client and process it
  - Server sends the service back to the client
- Servers are usually designed to handle multiple clients
- Usually servers and clients communicate over the internet
- Applications using the client-server model: email, WWW.

Concept of Socket

- The basic function in distributed programming is to transfer data between computers
- Each computer in a TCP/IP network has an **IP address**
- A socket acts as an endpoint instance of a communication connection between two computer processes and is identified by an **IP address** and a **port number**.
- A connection over a TCP/IP network is represented by a pair of sockets
- From the programmer’s viewpoint, a socket represents the mechanism to transfer data between computers

Streams and Datagrams

Uni-directional

Overheads

Unordered

Best-effort

Connectionless

Unreliable

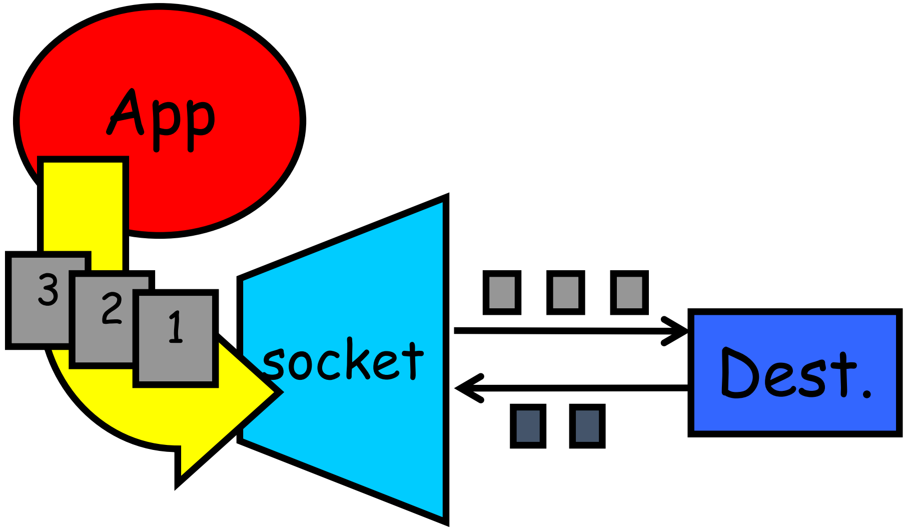
Ordered

Reliable

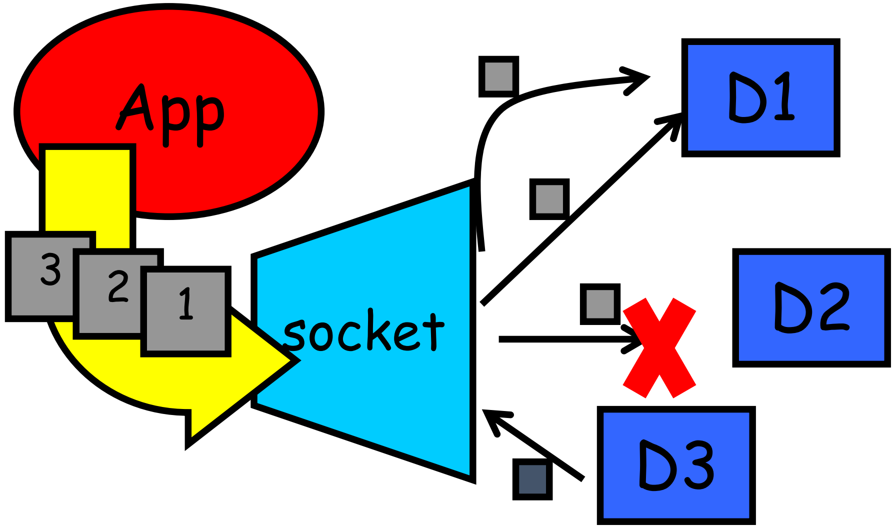
Bi-directional

Connection-oriented

Stream Sockets



Datagram Sockets



Java Sockets

- Java Sockets serve as communication channels between computers
- Supported by the **java.net** package API
- TCP Socket classes: ServerSocket and Socket
- UDP Socket classes: DatagramSocket and DatagramPacket
- A socket can be easily instantiated by calling the constructor of the classes.

TCP Sockets

- To set up a TCP server socket, we need a ServerSocket
- To establish a connection to this server, the server must react by accepting the connection set-up by calling accept() and providing a new socket for this connection.
- Often, a server is set to endlessly wait for connections accept() blocks until a new connection request arrives

```
int port = 9090;
ServerSocket server = new ServerSocket(port);
while(true) {
        System.out.println(“Waiting for client…”);
        Socket client = server.accept();
        System.out.println(“Client ” + client.getInetAddress() + “ connected.”);
}
```

How a Client Requests a Connection?

- After a ServerSocket is set up on the server, a client can address it to establish a connection.

```
int port = 9090;
Socket server = new Socket(“localhost”, port);
System.out.println(“Connection to “ + server.getInetAddress());
```

- Instead of using the host name, we can also use the IP address

```
int port = 9090;
Socket server = new Socket(“127.0.0.1”, port);
System.out.println(“Connection to “ + server.getInetAddress());
```

Socket and ServerSocket Methods

- **Socket** class
  - void close();
  - InetAddress getInetAddress();
- **ServerSocket** class
  - void close();
  - accept();
  - InetAddress getInetAddress();

Streams

- To exchange data over sockets, the **Socket** class uses **streams**
  - e.g., getInputStream() and getOutputStream()
- The **java.io** package has classes for input and output respectively.
- For transmitting simple types as int, long, String, the DataOutputStream provides a write method for each of these types: writeInt(), writeLong(), writeChar() and others.
- For transmitting strings, usually the UTF-8 format is used, which transfers the characters in Unicode: writeUTF().
- Similarly, DataInputStream exists with methods readInt(), readLong(), readChar() and readUTF()
- Converts different types into byte sequence for simple streams, and vice versa

Design Server and Client: TCP

- **Server**
  - SocketServer setup
  - Listen and accept connection
  - Setup I/O streams
  - Send to/receive from client
- **Client**
  - Socket setup (include server address)
  - Setup I/O streams
  - Send to/receive from server

TimeServer

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

public class TimeServer {
    public static void main(String[] args) throws IOException {
        ServerSocket server = new ServerSocket(9090);
        while(true){
                System.out.println("Waiting...");
                //establish connection
                Socket client = server.accept();
                System.out.println("Connected " + client.getInetAddress());
                //create IO streams
                DataInputStream inFromClient = new DataInputStream(client.getInputStream());
                DataOutputStream outToClient = new DataOutputStream(client.getOutputStream());
                System.out.println(inFromClient.readUTF()); //get any data from client
                Date date = new Date();
                outToClient.writeUTF(date.toString()); //send date to client
                client.close();
        }
    }
}
```

TimeClient

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

public class TimeClient {
  public static void main(String[] args)  throws IOException {
          Socket server = new Socket("localhost", 9090);
          System.out.println("Connected to " + server.getInetAddress());
          //create io streams
          DataInputStream inFromServer = new DataInputStream(server.getInputStream());
          DataOutputStream outToServer = new DataOutputStream(server.getOutputStream());
          //send to server
          outToServer.writeUTF("Time");
          //read from server
          String data = inFromServer.readUTF();
          System.out.println("Server said: " + data);
          server.close();
  }
}
```

UDP Sockets

- To set up a UDP socket, we need a DatagramSocket
- To establish a connection to this server, the server will act as a receiver of accepting DatagramPackets, e.g., receive().

```
int port = 9090;
DatagramSocket socket = new DatagramSocket(port);
while(true) {
        System.out.println(“Waiting for packets…”);
        byte[] data = new byte[1024];  //need to create an empty array and fill it
        DatagramPacket receivedPacket = new DatagramPacket(data, data.length);
        socket.receive(receivedPacket);
        String message = new String(receivedPacket.getData());
        System.out.println(“Received from client ” + receivedPacket.getAddress() + “ : ” + message);
}
```

How a Client Sends Packet?

- **After a** DatagramSocket **is set up as server(receiver), a client (sender) can send** DatagramPacket **without establishing a connection**
- **IP Address and Port are attached in the** DatagramPacket

```
DatagramSocket socket = new DatagramSocket();
byte[] data = “This is the message”.getBytes(); //convert a string to bytes
DatagramPacket packet = new DatagramPacket(data, data.length);
InetAddress dest = InetAddress.getByName(“localhost”);
int port = 9090;
packet.setAddress(dest); //add IP Address
packet.setPort(port);      //add Port
socket.send(packet);      //send the UDP datagram
socket.close();
```

DatagramSocket and DatagramPacket Methods

- **DatagramPacket** class
  - byte[] getData();
  - void setAddress(InetAddress a);
  - void setPort(int port);
  - InetAddress getAddress();
  - int getPort();
- **DatagramSocket** class
  - void connect(InetAddress a, int port);
  - void close();
  - void receive(DatagramPacket p);
  - void send(DatagramPacket p);

Convert Bytes to Data

- No input/output stream support
- Send DatagramPacket

```
String message = “Some message”;
byte[] data = message.getBytes();
DatagramPacket sendPacket = new DatagramPacket(data, data.length);
/*send DatagramPacket via DatagramSocket*/
```

- Receive DatagramPacket

```
byte[] data = new byte[1024];
DatagramPacket receivedPacket = new DatagramPacket(data, data.length);
/*receive DatagramPacket via DatagramSocket*/
data = receivedPacket.getData();
String message = new String(data);
```

Design Server and Client: UDP

- **Server**
  - DatagramSocket setup
  - Create empty byte array
  - Fill array with DatagramPacket received from client
  - Also, server can also send back to client
    - Get information of the client from the packet
- **Client**
  - DatagramSocket setup
  - Create byte data and DatagramPacket
  - Add server address and port to DatagramPacket
  - Send to server
  - Also, client can also receive from the server

Summary

- Discussed the concept of socket
- Stream and Datagram sockets
- Illustrated Stream Sockets and Datagram Sockets