Socket programming

- **Socket API**
  - Explicitly created, used, released by apps
  - Client/server paradigm
  - Two types of transport service via socket API:
    - unreliable datagram
    - reliable, byte stream-oriented

- **Socket**
  - An application-created, OS-controlled interface (a "door") into which application process can both send and receive messages to/from another application process over the network

Socket-programming using TCP

- **Client must contact server**
  - Server process must be running
  - Server must have created socket (door) that welcomes client's contact

- **Client contacts server**
  - Creating a TCP socket
  - Specifying IP address, port number of server process
  - Establish connection to server TCP (in Java this is done automatically when you create a socket, in C you must explicitly do it)

- **When contacted by client, server creates new socket for server process to communicate with client**
  - Allows server to talk with multiple clients
  - Source port numbers used to distinguish clients

Socket programming with TCP

Example client-server app:
1. client reads line from standard input (in Từ a server via socket (_pkg/server stream))
2. server reads line from socket
3. server converts line to uppercase, sends back to client
4. client reads, prints modified line from socket (in /from server stream)

Stream jargon

- **A stream**
  - A sequence of characters that flow into or out of a process.
  - **An input stream** is attached to some input source for the process, e.g., keyboard or socket.
  - **An output stream** is attached to an output source, e.g., monitor or socket.

Client/server interaction: TCP

**Server** (running on host 1.4.3)
- create socket, port 1000
- create connection socket
- accept incoming connection from host 1.4.3
- read reply from connection socket
- send reply to connection socket
- close connection socket

**Client**
- create socket
- connect to host 1.4.3, port 1000
- send request using client socket
- read reply from client socket
- close client socket
**Example: Java client (TCP)**

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

class TCPClient {
    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');

        BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);

        clientSocket.close();
    }
}
```

**Example: Java server (TCP)**

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

class TCPServer {
    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;

        ServerSocket welcomeSocket = new ServerSocket(6789);
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());

            clientSentence = inFromClient.readLine();
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

**Socket-programming using UDP**

**Socket programming with UDP**

- No "connection" between client and server
- No handshaking
- Sender explicitly attaches IP address and port of destination to each packet
- Server must extract IP address, port of sender from received packet
Client/server interaction: UDP

Server (running on hostid)

- Create socket, clientSocket = DatagramSocket()
- Create address (hostid, port)
- Send datagram request

Client

- Create socket, clientSocket = DatagramSocket()
- Create address (hostid, port)
- Write reply to serverSocket

Example: Java client (UDP)

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

public class UDPClient {
    public static void main(String[] args) throws Exception {
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, 6789);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new String(receivePacket.getData());
        System.out.println("FROM SERVER:" + modifiedSentence);
        clientSocket.close();
    }
}
```

Example: Java server (UDP)

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

public class UDPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(6789);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while (true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String(receivePacket.getData());
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```