Network programming (IT423+IT432)

Spring 2017

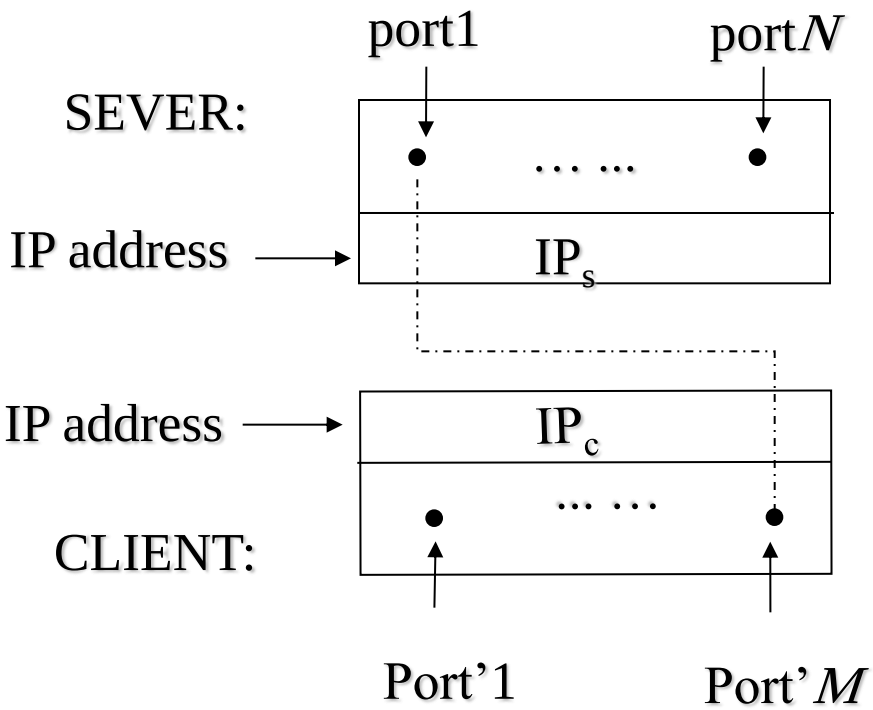
Dr. Islam Taj-Eddin

IT Dept., FCI, Assiut Univ.

Sockets for Clients

Sever - client communication

- A socket can be considered as a connection point.



Socket$_s$(Ip$_c$, Port's*M*, port1)

Socket$_c$(Ip$_s$, Port1, port'*M*)

Ports

- In general, computers only have one Internet address, but computers usually need to communicate with more than one host at a time.
- Ports map incoming data to a particular process or application running on a computer,
- With every socket, there needs to be an associated port number
- Example - regular post. The IP address is like the street address while the port number would represent the house number.

Ports

- Ports can range in value from 1 to 65535, with ports 1 to 1023 being reserved for root access in Unix.
- On Windows NT, 95 and Macs, any user can listen on any port.
- Only one program can listen on a given TCP port at the same time.
- However, many remote hosts can connect to the same remote port.

?

- You must at least specify the remote host and port to connect to.
- The host may be specified as either a string like "utopia.poly.edu" or as an `InetAddress` object.
- The port should be an int between 0 and 65535.

Internet Addresses

- Every computer on the net has an associated four-byte IP address that is unique. Usually represented in dotted quad format like 148.131.31.1. where each byte is an unsigned value in the range of 0 to 255.
- Since numbers are hard to remember, these numbers are mapped into names like www.uwinnipeg.ca or www.umanitoba.ca.
- It's the numerical address that is fundamental, not the name. You can have multiple names that represent the same IP address.

Internet Addresses

- java. net. InetAddress class are used to manage such addresses.
- Some are shown here: -
  - InetAddress getByName(String host)
  - InetAddress[] getAllByName(String host)
  - InetAddress getLocalHost()
  - String getHostName()
  - byte[] getHostAddress()
  - boolean isMulticastAddress()

TCP

- TCP (Transmission Control Protocol)
  - a connection-based protocol that provides a reliable flow of data between two computers.
  - It allows retransmission of lost data.
  - It provides multiple paths through different routers in case one goes down.
  - Bytes are delivered in the order they are sent.
- Applications using TCP to communicate
  - HTTP, FTP, Telnet, etc.

TCP

- Java networking classes use TCP
- java.net package provides the functionality for networking
  - java.net
    - URL()
    - URLConnection()
    - Socket()
    - SocketServer()

Sockets

- Host's native networking software handles the splitting of data into packets on the transmitting side of a connection, and the reassembly of packets on the receiving side.

- Java programmers are presented with a higher level abstraction called a **socket**.

Sockets

- A socket represents a reliable connection for data transmission between two hosts.

- Seven fundamental operations
    1. Connect to a remote machine
    2. Send data
    3. Receive data
    4. Close the connection
    5. Bind to a port
    6. Listen for incoming data
    7. Accept connections from remote machines on the bound port

Establishing a Connection

- Accomplished through the class constructors:
  - java.net.Socket()
    - client side implementation
  - java.net.ServerSocket()
    - server side implementation
- Each socket is associated with exactly one host.
- To connect to a different host, a new socket object must be created.

Sending/Receiving Data

- Sending and Receiving data is accomplished with input and output streams.
  - public InputStream getInputStream()
  - public OutputStream getOutputStream()
  - both of these classes throw an IOException

Closing a Connection

- There's a method to close a socket.
  - public synchronized void close()
  - this method throws an IOException as well.

- For a well behaved program, we need to close the I/O streams as well.

- Close any streams connected to a socket before closing the socket.
  - OutStream.close()
  - InStream.close()
  - Socket.close()

Example: Chat Client(l)

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

public class chat_client extends Thread {

    public static void main(String a[]) throws IOException {
        Socket sock = null;
        String host = “localhost";
        int port  9100;
```

Declaring a Socket object

Example: Chat Client(2)

```
if (a.length > 1) {
      port = Integer.parseInt(a[i]);
} //if
if (a.length > 0) {
      host = a[0];
}//if
System.out.println("using port “+ port + “connecting to “  + host);
try {
      sock = new Socket(host,port);
}//try
```

Establish a connection
to the specified bost and port

Example: Chat Client(3)

```
    catch (java.net.ConnectException e){
        System.out.println(e);
        System.exit(0);
    }//catch
    SendData sd = new SendData(sock);
    RcveData rd = new RcveData(sock);

    sd.start();
    rd.start();
    }//main
}//chat_client
```

Instantiate the supporting classes

Example: EchoTest(l)

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

public class EchoTest {
public static void main(String[] args) {
   String host="truffle";
Socket eSocket = null;
DataoutputStream os = null;
DataInputstream is = null;
DataInputStream stdIn = new DataInpuuStream(System.in);
```

Example: EchoTest(2)

```
try {
        eSocket = new Socket(host,7);
        os = new DataOutputStream(eSocket.getOutputStream());
        is = new
                DataTnputStream(eSocket.getInputStream()); }
//try
catch (UnknownHostException e) {
   System.err.println(“Unknown host: "+host);
}//catch
catch (IOException e) {
   System.err.println("No I/O connection to: “ + host);
}///catch
```

Example: EchoTest(3)

```
if (eSocket !=null && os!=null && is!=null) {
    try {
        String uTnput;
        while ((uInput=stdIn.readLine()) != null) {
            os.writeBytes(userlnput);
            os.writeByte('\n');
            System.out.println("echo:
                                +is.readLineo);
            if (userInput.equals("quit”))
              break;
        }//while
        os.close();
        is.close();
```

Example: EchoTest(4)

```
            eSocket.close();
        }//try
        catch (IOException e)
        System.err.println("I/O failed on the connection to: “ + host);
        }//catch
    }//if
  }//try
}//class
```

Socket Options

- TCP_NODELAY
- SO_LINGER
- SO_TIMEOUT
- SO_SNDBUF (Java 1.2 and later)
- SO_RCVBUF (Java 1.2 and later)
- SO_KEEPALIVE (Java 1.3 and later)

Socket Options

- Several methods set various socket options. Most of the time the defaults are fine.

```
public void setTcpNoDelay(boolean on) throws SocketException
public boolean getTcpNoDelay() throws SocketException
public void setSoLinger(boolean on, int val) throws SocketException
public int getSoLinger() throws SocketException
public synchronized void setSoTimeout(int timeout) throws SocketException
public synchronized int getSoTimeout() throws SocketException
```