Lecture 6
Application Layer –
Socket Programming in Java

Reading for Java Client/Server see “Relevant Links”

Some Material in these slides from J.F Kurose and K.W. Ross
All material copyright 1996-2007
Chapter 2: Application Layer

• Principles of network applications
• Web and HTTP
• FTP (Skip)
• Electronic Mail
  • SMTP, POP3, IMAP
• DNS
• Socket programming with TCP
  • Processes, Addresses and Ports
• Socket programming with UDP
Network Process Communication
Communication Between Hosts

• So, how do you “talk” to another host?
• What do you need to know?
  • Minimum
    • Name or Address of host
    • Agree on Protocol
  • Rely on Other Layers of Network
    • Lower layers to get message delivered
Addressing in Processes

• To receive messages, **process** running on a **host (machine)** must have **identifier**
  • Host device has unique 32-bit number **IP address**
    • Like: **146.187.134.22**

• **Question**
  • Is IP address of host on which process runs enough for identifying the process?
Addressing in Processes

• **NO**, it's not enough to have just host IP address
  • As we saw before, **146.187.134.22**

• **Each** process must have its own identifier
  • **Many** processes can run on same host!!
  • What is the identifier called?
Addressing in Processes

• Process “identifiers” = Port Numbers

• Port Numbers
  • Standard way to uniquely identify processes
  • 16 bit numbers, from 1 – 65,535
  • Certain port numbers are reserved by the operating system
    • Port numbers below 1023 are reserved
    • Above 1023, anyone can use

Reserved Numbers

$ /etc/services  
C:\WINDOWS\system32\drivers\etc\services  

Linux
Windows
Addressing in Processes

Server Side
Well Known Port Numbers
• IP and port number of the server
• Well-known and advertised so client knows where to find service

Client Side
Ephemeral Port Numbers
• Port number on client side
• Generally allocated automatically by the kernel
Addressing in Processes

• Three Groups of Port Numbers
  • 1. Well Known Port Numbers 1 - 1023
    • IANA assigns port numbers to protocols that have been standardized using RFC process
  • 2. Registered Port Numbers 1024 - 49151
    • Non-RFC Server Application can reserve one of these port numbers, through IANA
      • If approved, the IANA will register that port number and assign it to the application
Addressing in Processes

• Three Groups of Port Numbers
  • 3. Ephemeral Port Numbers  49152-65535
  • These ports are neither reserved nor maintained by IANA

• Can be used for any purpose without registration, so they are appropriate for a private protocol, or temporary connection
Wireshark Capture Showing Ephemeral Ports
Addressing in Processes

• When you create a socket in Java, you associate it to a port number
• Called “binding to a port”
• So, unique identifiers for addressing a process on a given machine consist of:
  - IP address and port number
  - Example:
    To send HTTP message to gaia.cs.umass.edu web server:
    IP address: 128.119.245.12
    Port number: 80
Client Server Programming in Java
Goal of Client/Server Programming

• What’s the goal for applications in a networked environment?
  • Get information from my machine and move it to another machine or vice-versa
  • Similar to reading and writing files
    Except ... files exist on a remote machine
  • Network programming in a nutshell !!!
What is a socket?

- A **socket**, software abstraction used to represent the "terminals" of a connection between two machines – **like electric socket**

- **For a Given Connection**
  - Socket on each machine and
  - A hypothetical "cable" running between two machines
  - Each "cable" end plugs into socket
  - Physical hardware and cabling between machines unknown and not needed for communication
  - Its an abstraction ... !!!!!
Socket Programming

Goal Learn to build client/server applications that communicate using sockets

Socket API – What is it?
• TCP/IP Stack funded by ARPA
  • Resides in Operating System
• Needed an API into stack
• Supported by most languages, C, C#, Java, Python
• Was introduced BSD 4.1 UNIX, 1981

• Two types of transport service supported by the socket API:
  1. Unreliable - UDP
  2. Reliable - TCP
Clients and Servers

- Servers
  - What do they do?
    - Serve files
    - Serve video streams
    - Serve emails
    - Serve print jobs
    - Server data
    - Do not serve Pizza !!!
Clients and Servers

- What do Clients do? What is their relationship to servers?
  - Request things from servers
  - Browser programs
  - Email readers and writers
  - Request data from databases
Socket Communication

- A server (program) runs on a specific computer and has a socket that is bound to a specific port.
- Server waits and listens to socket for a client to make a connection request.
Socket Communication

• If everything goes well, server accepts connection
• Upon acceptance, server gets new socket bound to a different port.
• Needs a new socket so that it can continue to listen to the original socket for connection requests while serving the connected client
• New socket is for responding back to the client
Socket Programming Using TCP

**Socket:** Door between application process and end-end-transport protocol (UDP or TCP)

**TCP Service:** reliable transfer of *bytes* from one process to another
Socket Programming Using TCP

Client must first create a connection to server before sending data!!

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

Client contacts server by
- Creating client-side TCP socket
- Specify IP address, port number of server process
- When **client creates socket**: client TCP establishes connection to server TCP
Socket Programming Using TCP

• When contacted by client,
  • TCP Server creates new socket
  • Does this for each client
  • Allows server to talk with multiple clients
  • Source port numbers used to distinguish clients
Client/Server socket interaction: TCP

Server (running on hostid)
- create socket, port=50, for incoming request:
  welcomeSocket = ServerSocket()
- wait for incoming connection request
  connectionSocket = welcomeSocket.accept()
- read request from connectionSocket
- write reply to connectionSocket
- close connectionSocket

Client
- create socket, connect to hostid, port=50
  clientSocket = Socket()
- send request using clientSocket
- read reply from clientSocket
- close clientSocket
Stream jargon

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

Example: Uppercase Converter

1) Client reads line from standard input \texttt{inFromUser} stream, sends to server via socket \texttt{outToServer} stream

2) Server reads line from socket

3) Server converts line to uppercase, sends back to client

4) Client reads and then prints modified line from socket \texttt{inFromServer} stream
Client Side Character Converter

• First, we will look at code for the Client side of the Upper Case Converter ..... 
• Next slide defines some Java Socket methods client side
Java Socket programming with TCP

• `java.net.socket`
  – Java package that defines socket programming

• **Socket** object is Java representation of TCP connection
  - When socket created on the client side, a connection is opened to a destination

• **Methods**
  - Two most important methods:
    - `getInputStream()` and `getOutputStream()`
  - Return stream objects used to communicate through the socket
  - The `close()` method tells underlying operating system to terminate the connection
Client Uppercase Converter

- Code Logic
  - Accept a string to be converted from the keyboard
  - Attach itself to an **Uppercase Server**
  - Send the string through the socket
  - Read the string back from the socket
  - Print the string to the terminal
  - Close the connection
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());

        String sentence = inFromUser.readLine();
        modifiedSentence = processSentence(sentence);

        outToServer.writeUTF(modifiedSentence);
    }

    public static String processSentence(String sentence) {
        // Implement sentence processing logic
        return modifiedSentence;
    }
}
```
Example: Java client (TCP), cont.

```java
BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();
```
Java Server (TCP)

• ServerSocket represents listening TCP connection
  • Once an incoming connection is requested,
  • ServerSocket object will return a Socket object representing the connection

• Methods
  • Most important method is accept()
  • Returns a Socket, connected to a client
  • The close() method tells operating system to stop listening for requests on the socket
  • Other Methods are also provided
Server Uppercase Converter

- **Code Logic**
  - Set up a `ServerSocket`
  - Listen for a client on the `ServerSocket` and spawn a **new socket** for the client
  - Attach input and output streams to **new socket**
  - Read the string from the socket and convert it to uppercase
  - Send the uppercase string through the socket
  - Close the connection to the new socket
  - Return to listening for a new client
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()));

            // Create welcoming socket at port 6789
            // Waits, on welcoming socket for contact by client
            // Create input stream, attached to socket
            // Socket at port 6789
        }
    }
}
```
Example: Java server (TCP), cont

Create output stream, attached to socket

Read in line from socket

Write out line to socket

```
DataOutputStream outToClient =
    new DataOutputStream(connectionSocket.getOutputStream());

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence);

connectionSocket.close();
```

End of while loop, loop back and wait for another client connection
Running the Client/Server
Example of Server Creating Socket Object

• Demo a Server that listens on a port for a connection, and creates a “socket” object for talking to client
• Has Ephemeral port numbers
• Can use telnet to play the role of a client
• Steps:
  1. Start my Server: HelloServer 5555
  2. telnet localhost 5555
I am running my server, HelloServer on 5555
Java .net Package
Java Sockets Programming

• The package java.net provides support for socket programming (and more)
• Docs for Java Version 10 are here
  https://docs.oracle.com/javase/10/docs/api/java/net/InetAddress.html

• Typically you import everything defined in this package with:
  ```
  import java.net.*;
  ```
Classes

InetAddress

Socket

ServerSocket
InetAddress Class

- InetAddress class is Java’s high-level representation of an IP address,
  - Both IPv4 and IPv6.
- Used by most of other networking classes,
  - Socket, ServerSocket, URL, DatagramSocket, DatagramPacket, and more.
  - Usually, it includes both a hostname and an IP address.

Nice explanation of InetAddress Class

InetAddress Class

- InetAddress has static factory methods that connect to a DNS server to resolve a hostname
- The most common is `InetAddress.getByName()`
- `getByName` actually makes a connection to the local DNS server to look up the name and the numeric address
- Throws unknown host exception
InetAddress class

- Other static methods you can use to create new InetAddress objects
  
  ```java
  InetAddress addr = InetAddress.getByName( "www.ewu.edu" );
  ``

- Throws `UnknownHostException`

Example

```java
InetAddress addr = InetAddress.getByName( "www.ewu.edu" );
```

- Throws `UnknownHostException`
Sample Code: Lookup.java

- Uses InetAddress class to lookup hostnames found on command line.

> java Lookup www.ewu.edu
www.ewu.edu:  146.187.224.198

> java Lookup www.yahoo.com
www.yahoo.com:  209.131.36.158
try {
    InetAddress a = InetAddress.getByName(hostname);
    System.out.println (hostname + "::" + a.getHostAddress());
} catch (UnknownHostException e) {
    System.out.println("No address found for " +  hostname);
}
InetAddress Class

- You can also do a reverse lookup by IP address
- For example, if you want hostname for address 208.201.239.100, pass the dotted quad address to InetAddress.getByName():

Example

```java
InetAddress address = InetAddress.getByName("208.201.239.100");
System.out.println(address.getHostName());
```
Server Classes

- Two classes of socket used for TCP
  1. **java.net.ServerSocket** class
     - Used by server applications to obtain port and listen for client requests
   - **ServerSocket class – some constructors**
     - `public ServerSocket(int port) throws IOException`
       - Attempts to create a server socket bound to specified port. An exception happens if port is already bound by another application
     - `public ServerSocket(int port, int backlog) throws IOException`
       - Similar to previous constructor, backlog parameter specifies how many incoming clients to store in wait queue
Server Classes

• Here are common methods of ServerSocket class:

  public Socket accept() throws IOException
    • Waits for an incoming client
    • Method blocks until either client connects to server on specified port or socket times out, assuming a time-out value has been set using setSoTimeout() method
    • Otherwise, this method blocks indefinitely

  public void setSoTimeout(int timeout)
    • Sets time-out value for how long the server socket waits for a client during accept()
Server and Client Socket Class

2. **java.net.Socket** class is socket both client and server use to communicate
   - Client obtains Socket object by instantiating one
   - Server obtains Socket object from return value of accept() method

• **Constructor:**
  
  ```java
  public Socket(String host, int port) throws UnknownHostException, IOException
  ```

• This method attempts to connect to specified server at specified port

• If this constructor does not throw an exception, connection is successful and client is connected to server
Server and Client Class

- Two most important methods of Socket class are:
  - `public InputStream getInputStream() throws IOException`
    - Returns input stream of socket
    - Input stream is connected to the output stream of the remote socket
  - `public OutputStream getOutputStream() throws IOException`
    - Returns output stream of the socket
    - Output stream is connected to input stream of remote socket
Summary

• Covered briefly, client-server programming in Java environment
  • Uses sockets – virtual pipe connections across the Internet

• Lets you read/write data to processes on local or very remote hosts

• Will explore more complicated client/server applications in programming assignments
Next … Java UDP Sockets