Application Development with TCP/IP

Brian S. Mitchell
Drexel University
Agenda

- TCP/IP Application Development Environment
- Client/Server Computing with TCP/IP
- Sockets
- Port Numbers
- The TCP/IP Application Programming Interface (API)
- Example: A Distributed File Viewer
- Java Socket Implementation
TCP/IP applications are constructed using the sockets interface

- Open standard
  - Extensions to support the Windows (3.1, 95, NT) environment have been developed by an open committee (Winsock)
TCP/IP provides peer to peer communication between two application processes
- The peer applications can execute on the same or different machines
- Client application initiates the conversation
- Server application accepts the conversation
- Client applications generally request services whereas server applications generally provide services
  - Example: CETS
Application Responsibilities

- Client Applications
  - Locate and request a service from a server
  - Can run on a single (DOS, Windows 3.x) or a multi-process (Windows 95, NT, OS2, UNIX, MVS) operating system

- Server Applications
  - Server has additional responsibilities beyond providing a service to a client
    - **Authentication** - verifying the identity of the client
    - **Authorization** - determining if a given client is permitted to access the service that the server supplies
    - **Data Security** - guaranteeing that the data that is exchanged between the client and the server is not revealed or compromised
Application Responsibilities (con’t)

- **Privacy** - keeping information about an individual (e.g. user ID and password) from unauthorized access
- **Protection** - guaranteeing that network applications cannot abuse system resources
- Requires a multiprocessing operating system to concurrently support more than one client.
Application Development with TCP/IP

- Checklist
  - ✔ IP Address
  - ✔ Socket
  - ✔ Port Number
  - ✔ Socket Verbs (Functions)
Sockets

A socket is a 16-bit number that is used as an index into a socket descriptor table.

- Each entry in this table references a socket data structure.
- The socket data structure contains all of the information that is required to setup and exchange data over an IP conversation.
- The socket model has many advantages:
  - Support for different communication protocols (Raw IP, UDP/IP, TCP/IP)
  - Makes receiving and sending data over a network synonymous with performing read and write operations on a local file.

### Socket Descriptor Table

<table>
<thead>
<tr>
<th>Socket</th>
<th>Socket Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
</tr>
<tr>
<td>9654</td>
<td></td>
</tr>
<tr>
<td>3156</td>
<td></td>
</tr>
<tr>
<td>9512</td>
<td></td>
</tr>
</tbody>
</table>

### Socket Data Structure

- Family: PF_INET
- Service: SOCK_STREAM (TCP)
- Local IP: 172.21.100.8
- Remote IP: 172.21.100.45
- Local Port: 3571
- Remote Port: 6358
When a socket is first created it does not contain enough information to initiate a network conversation.

- Special socket functions (TCP/IP API’s) must be called in order to populate the socket data structure.
Port Numbers

- A port number is a 16-bit integer that represents the logical endpoint of a conversation
- The mechanism that enables a computer to support more than one IP conversation at a time
- Port numbers can (and most likely will) be different for the same conversation on the client and server computers
A Client application can obtain the server application port number in one of two ways:

- Must know the port number in advance
  - Provided by the server’s documentation
- Can determine the port number dynamically by sending a message to the port mapper service
  - All services that want their port number to be found by name must register (and deregister) with the port mapper service
  - Port number is obtained by calling the getportbyname (...) API
TCP/IP Client/Server Conversation

**Client Program**
- Create a socket - `socket()`
- Connect to the server - `connect()`
- Send data to the server - `write()`
- Receive data from the server - `read()`
- Terminate the conversation - `close()`

**Server Program**
- `socket()` - create a socket
- `bind()` - bind program to a port number
- `listen()` - get ready to accept client requests
- `accept()` - wait for a client request
- `read()` - Receive data from the client
- `write()` - Send data to the client
- `close()` - Terminate the conversation

TCP/IP socket programs are controlled by a series of API’s:
- Socket, connect, accept, read, write, close, listen, bind, accept
Supporting concurrent clients with Socket Servers

Multiprocessing Socket Server

s1 = socket ()  // Create a socket
bind (s1, port_number)  // Bind the socket to a port
listen (s1)  // Place the socket into a listening mode
s2 = accept (s1)  // Wait for a client connection

accept( ) returns a new socket to handle the client connection

create_process(client_handler, s2)  // Create a new process to service the client request

client_handler (client_socket)
read(client_socket)
write(client_socket)
close(client_socket)

- The server creates a new process or thread to handle the clients request
- Client connection is serviced by the new socket that was returned by the accept( ) call
- The original socket is freed up to handle another client request
The distributed file service will consist of the following components:

- **Client Program**
  - Connect to the remote server program
  - Request a file by name
  - Receive an image of the actual file from the server program
  - Display the file image for the user to view

- **Server Program**
  - Wait for a client request
  - Read a file from the local hard disk based on the client request
  - Send an image of the file to the remote client

---

**Example: A Distributed File Viewer Service**

- **Client Program**
- **Server Program**
- **Hard Disk**

- Get me file ‘c:\test.txt’
- Image of the ‘c:\test.txt’ file
Example: A Distributed File Server (con’t)

- Read the file ‘c:\test.txt’ on the server given
  - Client IP = 172.21.100.7
  - Server IP = 172.21.100.45
  - File read service port number = 1031
Sockets in Java

- The java.net package provides several classes that enable socket-based communication in Java
- The Java socket classes are:
  - java.net.Socket
  - java.net.ServerSocket
  - java.net.DatagramSocket
  - java.net.SocketImpl
- See the Java documentation for details on these classes
Parameters in Java Class

Constructors

- **String host**: String host can be specified either as machine name or IP address. For queen.mcs.drexel.edu, machine name is "queen.mcs.drexel.edu", IP address is "129.25.6.176". In unix, if you know the machine name, then you can use "nslookup" to identify the IP address of the machine if the machine is connected to the network.

- **int port**: integer port number

- **boolean stream**: It specify whether communication through the socket based on TCP protocol, or UDP datagram protocol. The default protocol is TCP.
Parameters in Java Class
Constructors (con’t)

InetAddress address: InetAddress is the class that represents an Internet Protocol (IP) address. It does not have a constructor, but instead it has several methods which return the InetAddress objects. Method getLocalHost() returns an InetAddress for the local host. GetAllByName() returns an array of InetAddress that represents all of the available addresses for a host specified by name; getByName() returns the InetAddress of a host specified by name. The InetAddress class definition is shown on the next slide.
The InetAddress Class

public final class InetAddress extends Object {
    // Methods
    public boolean equals(Object obj);
    public byte[] getAddress();
    public static InetAddress[] getAllByName(String host);
    public static InetAddress getByName(String host);
    public String getHostName();
    public static InetAddress getLocalHost();
    public int hashCode();
    public String toString();
}
Implementing A Server

- Create a socket, use the ServerSocket class
  - ServerSocket s = new ServerSocket(4444)
  - Socket listens to port 4444
- Accept a connection request by a client
  - Socket client_socket = s.accept();
  - Creates a socket that communicates with the client
- Open the input and output stream to/from the client
  - DataInputStream in = new DataInputStream(client_socket.getInputStream());
  - PrintStream out = new PrintStream(client_socket.getOutputStream());
Implementing a Server

- Close the sockets
  - s.close();
  - client_socket.close();
  - in.close();
  - out.close();
- Closing the sockets and streams terminates the session between the server process and the client process
Implementing a Client

- Create a socket
  - Specify server host name or IP address and port number
  - Socket c = new Socket("queen.mcs.drexel.edu", 4444)
  - If you only have a single machine you can use the “127.0.0.1” local loopback address

- Open the input and output stream on the socket
  - DataInputStream in = new DataInputStream(c.getInputStream());
    // get input stream
  - PrintStream out = new PrintStream(c.getOutputStream());
    // get output stream
Implementing the Client

- Close the sockets
  - `in.close();  // close the input stream`
  - `out.close();  // close the output stream`
  - `c.close();    // close the client socket`
class EchoServer
{
    public static void main(String[] args)
    {
        try {
            ServerSocket s = new ServerSocket(4444);
            Socket client_socket = s.accept();
            DataInputStream in = new DataInputStream(client_socket.getInputStream());
            PrintStream out = new PrintStream(client_socket.getOutputStream());
            ...
        }
    }
}
Sample Server

...   
   out.println("Hello! Enter BYE to exit. \r");
   boolean done = false;

   while (!done) {
      String str = in.readLine();
      if (str == null)
         done = true;
      else {
         out.println("Echo: " + str + "\r");
         if (str.trim().equals("BYE"))
            done = true;
      }
   }
}...
Sample Server

```java
    client_socket.close();
}
catch (Exception e)
{
    System.out.println(e);
}
}  //end of class EchoServer
```
class EchoClient {
    public static void main(String[] args) {
        try {
            Socket c = new Socket("queen.mcs.drexel.edu", 4444);
            DataInputStream in = new DataInputStream(c.getInputStream());
            boolean more = true;
            while (more) {
                String str = in.readLine();
                if (str == null)
                    more = false;
                else
                    System.out.println(str);
            }
        } ...
    }
}
Sample Client

```java
} catch (IOException e) {
  System.out.println("Error" + e);
}

} //end of class EchoClient
```