Network Programming using sockets

TCP/IP layers

<table>
<thead>
<tr>
<th>Layers</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Messages (UDP) or Streams (TCP)</td>
</tr>
<tr>
<td>Transport</td>
<td>UDP or TCP packets</td>
</tr>
<tr>
<td>Internet</td>
<td>IP datagrams</td>
</tr>
<tr>
<td>Network interface</td>
<td>Network specific frames</td>
</tr>
<tr>
<td>Underlying network</td>
<td></td>
</tr>
</tbody>
</table>
The programmer’s conceptual view of a TCP/IP Internet

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API
- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket
- a host-local, application-created/owned, OS-controlled interface (a “door”) into which application process can both send and receive messages to/from another (remote or local) application process
Sockets and ports

Berkeley Sockets (1)

- Socket primitives for TCP/IP.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>Create a new communication endpoint</td>
</tr>
<tr>
<td>Bind</td>
<td>Attach a local address to a socket</td>
</tr>
<tr>
<td>Listen</td>
<td>Announce willingness to accept connections</td>
</tr>
<tr>
<td>Accept</td>
<td>Block caller until a connection request arrives</td>
</tr>
<tr>
<td>Connect</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>Send</td>
<td>Send some data over the connection</td>
</tr>
<tr>
<td>Receive</td>
<td>Receive some data over the connection</td>
</tr>
<tr>
<td>Close</td>
<td>Release the connection</td>
</tr>
</tbody>
</table>
Socket programming with TCP

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-local TCP socket
- specifying IP address, port number of server process

- When client creates socket, client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Example client-server app:
- client reads line from standard input (inFromUser stream), sends to server via socket (outToServer stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (inFromServer stream)

Input stream: sequence of bytes into process
Output stream: sequence of bytes out of process
**Client/server socket interaction: TCP**

**Server (running on hostid)**

- create socket, port=x, 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=x:
  - `clientSocket = Socket()`
- send request using `clientSocket`
- read reply from `clientSocket`
- close `clientSocket`

---

**Berkeley Sockets (2)**

- **Server**
  - `socket` → `bind` → `listen` → `accept` → `read` → `write` → `close`
  - Synchronization point
  - Communication

- **Client**
  - `socket` → `connect` → `write` → `read` → `close`

- Connection-oriented communication pattern using sockets.
Sockets used for streams

Requesting a connection

- `s = socket(AF_INET, SOCK_STREAM, 0)`
- `connect(s, ServerAddress)`
- `write(s, "message", length)`

Listening and accepting a connection

- `s = socket(AF_INET, SOCK_STREAM, 0)`
- `bind(s, ServerAddress);`
- `listen(s, 5);`
- `sNew = accept(s, ClientAddress);`
- `n = read(sNew, buffer, amount)`

ServerAddress and ClientAddress are socket addresses

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

---

**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()));
```
Example: Java server (TCP), cont

Create output stream, attached to socket

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

Read in line from socket

clientSentence = inFromClient.readLine();

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

Write out line to socket

outToClient.writeBytes(capitalizedSentence);
Client/server socket interaction: UDP

Server (running on hostid)

- create socket, port=x, for incoming request: `serverSocket = DatagramSocket()`
- read request from `serverSocket`
- write reply to `serverSocket` specifying client host address, port number

Client

- create socket, `clientSocket = DatagramSocket()`
- Create, address (hostid, port=x), send datagram request using `clientSocket`
- read reply from `clientSocket`
- close `clientSocket`

Sockets used for datagrams

Sending a message

```python
s = socket(AF_INET, SOCK_DGRAM, 0)
binding(s, ClientAddress)
sendto(s, "message", ServerAddress)
```

Receiving a message

```python
s = socket(AF_INET, SOCK_DGRAM, 0)
binding(s, ServerAddress)
amount = recvfrom(s, buffer, from)
```

*ServerAddress* and *ClientAddress* are socket addresses
Example: Java client (UDP)

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

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, 9876);
        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 client (UDP), cont.

Create input stream
Create client socket
Translate hostname to IP address using DNS
Create datagram with data-to-send, length, IP addr, port
Send datagram to server
Read datagram from server
**Example: Java server (UDP)**

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

class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        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);
        }
    }
}
```

**Example: Java server (UDP), cont**

```java
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);
```