COSC 6377
Computer Networks

Socket Programming
Md. Tarikul Islam
What is socket

Network based system consist of Client, Server and Media for Communication

Need connection between client and server.

Sockets are a protocol independent method of creating a connection between processes.
• Transport layer is involved.

• TCP is a connection-oriented protocol that provides a reliable flow of data between two computers. Example applications that use such services are HTTP, FTP, and Telnet.

• UDP is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival and sequencing. Example: Multimedia, voice over IP
Concept of Port Numbers

- The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer. Port is represented by a positive (16-bit) integer value.
- Port numbers are used to identify “entities” on a host.
- Port numbers can be:
  - Well-known (port 0-1023)
  - Dynamic or private (port 1024-65535)
- Servers/daemons usually use well-known ports.
  - Any client can identify the server/service
  - HTTP = 80, FTP = 21, Telnet = 23, ...
  - `/etc/service` defines well-known ports.
- Clients usually use dynamic ports.
  - Assigned by the kernel at run time.
Host Identification

• Every computer on the Internet is identified by a unique, 4-byte IP address.

• Humans prefer to deal with names not addresses
  – DNS provides mapping of name to address
  – Name based on administrative ownership of host
Communication

• Server has a socket bound to a specific port.
• Server listen to the socket for a client connection request.
• If every thing is ok then server accept the connection.
Socket Programming in java

Java provides a set of classes, defined in a package called java.net.
http://download.oracle.com/javase/1.4.2/docs/api/java/net/package-summary.html

The Classes

• ContentHandler
• DatagramPacket
• DatagramSocket
• DatagramSocketImpl
• HttpURLConnection
• InetAddress
• MulticastSocket
• ServerSocket
• Socket
• SocketImpl
• URL
• URLConnection
• URLEncoder
• URLStreamHandler

The Interfaces
• ContentHandlerFactory
• FileNameMap
• SocketImplFactory
• URLStreamHandlerFactory
Exceptions

- BindException
- ConnectException
- MalformedURLException
- NoRouteToHostException
- ProtocolException
- SocketException
- UnknownHostException
- UnknownHostException
- UnknownHostException
TCP/IP socket programming (server)

1. Open the Server Socket:
   
   ServerSocket server = new ServerSocket(PORT);

2. Wait for the Client Request:

   Socket client = server.accept();

3. Create I/O streams for communicating to the client

   DataInputStream is = new DataInputStream(client.getInputStream());
   DataOutputStream os = new DataOutputStream(client.getOutputStream());

4. Perform communication with client

   Receive from client: String line = is.readLine();
   Send to client: os.writeBytes("Hello\n");

5. Close socket:

   client.close();
import java.net.*;
import java.io.*;
public class SimpleServer {
    public static void main(String args[]) throws IOException {
        // Register service on port 1254
        ServerSocket s = new ServerSocket(1254);
        Socket s1 = s.accept(); // Wait and accept a connection
        // Get a communication stream associated with the socket
        OutputStream s1out = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream(s1out);
        // Send a string!
        dos.writeUTF("Hi there");
        // Close the connection, but not the server socket
        dos.close();
        s1out.close();
        s1.close();
    }
}
TCP/IP-client

1. Create a Socket Object:
   Socket client = new Socket(server, port_id);
2. Create I/O streams for communicating with the server.
   is = new DataInputStream(client.getInputStream());
   os = new DataOutputStream(client.getOutputStream());
3. Perform I/O or communication with the server:
   Receive data from the server: String line = is.readLine();
   Send data to the server: os.writeBytes("Hello\n");
4. Close the socket when done:
   client.close();
import java.net.*;
import java.io.*;
public class SimpleClient {
    public static void main(String args[]) throws IOException {
        // Open your connection to a server, at port 1254
        Socket s1 = new Socket("localhost",1254);
        // Get an input file handle from the socket and read the input
        InputStream s1In = s1.getInputStream();
        DataInputStream dis = new DataInputStream(s1In);
        String st = new String (dis.readUTF());
        System.out.println(st);
        // When done, just close the connection and exit
        dis.close();
        s1In.close();
        s1.close();
    }
}
UDP Socket programming

• Using datagram packet.
• Datagram packet format
  | Msg | length | Host | serverPort |

Key Methods:
byte[] getData() - Returns the data buffer.
int getLength() - length of data send or received
void setData(byte[] buf) - set the data buffer
void setLength(int length) - set the length of packet
UDP Socket programming

• DatagramSocket is used for send or receive data.
• Key methods:
  void send(DatagramPacket p)- Sends a datagram packet from this socket.
  void receive(DatagramPacket p)- Receives a datagram packet from this socket.
Sample server

// UDPServer.java: A simple UDP server program.
import java.net.*;
import java.io.*;
public class UDPServer {
    public static void main(String args[]){
        DatagramSocket aSocket = null;
        if (args.length < 1) {
            System.out.println("Usage: java UDPServer <Port Number>");
            System.exit(1);
        }
        try {
            int socket_no = Integer.valueOf(args[0]).intValue();
            aSocket = new DatagramSocket(socket_no);
            byte[] buffer = new byte[1000];
            while(true) {
                DatagramPacket request = new DatagramPacket(buffer,
                    buffer.length);
            }
        } catch (Exception e) {
            System.out.println("Error: ");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
Server Cont.

aSocket.receive(request);
DatagramPacket reply = new
    DatagramPacket(request.getData(), request.getLength(), request.getAddress(), request.getPort());
aSocket.send(reply);

} catch (SocketException e) {
System.out.println("Socket: "+ e.getMessage());
}
catch (IOException e) {
System.out.println("IO: "+ e.getMessage());
}
finally {
if (aSocket != null)
aSocket.close();
}
Sample client

// UDPClient.java: A simple UDP client program.
import java.net.*;
import java.io.*;
public class UDPClient {
    public static void main(String args[])
    {
        // args give message contents and server hostname
        DatagramSocket aSocket = null;
        if (args.length < 3) {
            System.out.println("Usage: java UDPClient <message> <Host name> <Port number>");
            System.exit(1);
        }
        try {
            aSocket = new DatagramSocket();
            byte [] m = args[0].getBytes();
            InetAddress aHost = InetAddress.getByName(args[1]);
            int serverPort = Integer.valueOf(args[2]).intValue();
            DatagramPacket request =
                new DatagramPacket(m, args[0].length(), aHost, serverPort);
            aSocket.send(request);
            byte[] buffer = new byte[1000];
DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
aSocket.receive(reply);
System.out.println("Reply: " + new String(reply.getData()));
}
catch (SocketException e) {
System.out.println("Socket: " + e.getMessage());
}
catch (IOException e) {
System.out.println("IO: " + e.getMessage());
}
finally {
if (aSocket != null)
aSocket.close();
}
Socket Programming in C


- **socket**: creates a socket of a given domain, type, protocol
- (buy a phone)
- **bind**: assigns a name to the socket (get a telephone number)
- **listen**: specifies the number of pending connections that can be queued for a server socket. (call waiting allowance)
- **accept**: server accepts a connection request from a client
- (answer phone)
- **connect**: client requests a connection request to a server
- (call)
- **send, sendto**: write to connection (speak)
- **recv, recvfrom**: read from connection (listen)
- **shutdown**: end the call
Socket Address

Struct sockaddr_in{
    sa_family_t sin_family;  // =AF_INET
    in_port_t sin_port;      // is a port number
    struct in_addr sin_addr; // an ip address
}
Server Action

Server performs the following actions

- **socket**: create the socket
- **bind**: give the address of the socket on the server
- **listen**: specifies the maximum number of connection requests that can be pending for this process
- **accept**: establish the connection with a specific client
- **send,recv**: stream-based equivalents of read and write (repeated)
- **shutdown**: end reading or writing
- **close**: release kernel data structures
Sample server TCP/IP

sockaddr_in serverAddr;
sockaddr &serverAddrCast=(sockaddr &serverAddr);
// get a tcp/ip socket
int listenFd=socket(AF_INET,SOCK_STREAM,0);
bzero(&serverAddr,sizeof(serverAddr));
serverAddr.sin_addr.s_addr=htonl(INADDR_ANY);
serverAddr.sin_port=htons(13);
bind(listenFd,&serverAddrCast,sizeof(serverAddr));
listen(listenFd,5);
for(: : )
{
    int connectFd=accept(listenFd,(sockaddr*)NULL,NULL);
    // read and write operation here
    shutdown(connectFd,2);
    close(connectFd);
}
Client Action

Client performs the following actions

• socket: create the socket
• connect: connect to a server
• send,recv: (repeated)
• shutdown
• close
sockaddr_in serverAddr;
sockaddr &serverAddrCast=(sockaddr &)serverAddr;
//get a tcp/ip socket
int sockFd=socket(AF_INET,SOCK_STREAM,0);
bzero(&serverAddr,sizeof(serverAddr));
serverAddr.sin_family=AF_INET;
//host IP# in dotted decimal format!
inet_pton(AF_INET,serverName,serverAddr.sin_addr);
serverAddr.sin_port=htons(13);
connect(sockFd,serverAddrCast,sizeof(serverAddr));
//read write operation here
shutdown(sockFd.2);
close(sockFd);
• http://www.slideshare.net/sunfacepriya/socket-programming-tutorial-1227317798640739-8?src=related_normal&rel=10496


• http://www.cs.usfca.edu/~parrt/course/601/lectures/sockets.html