Sockets

HY556 – Distributed Systems

Computer Science Department

What is a Socket

- A socket is one end-point of a two way communication link between two programs running on the network
- It represents the connection between a client program and a server program
- Imagine the analogous of a telephone device!

Socket types

- Stream socket
  - Streaming, bidirectional connection
  - Ordered, reliable delivering of packets
- Datagram socket
  - There is not a connection. Each packet is sent independently from the others
  - Unordered, unreliable delivering of packets
- Raw sockets
  - Out of our scope

Stream Socket lifetime

- Creation
  - Unbound ‘file descriptor’ of STREAM socket type
- Binding
  - Assigning a name to the socket – until a name is assigned, no messages may be received. Communicating processes are bound by an *association*, which in Internet is composed of local and foreign addresses, and local and foreign ports.
  - The ***bind()*** system call specifies half of an association {local address, local port}, while the ***connect*** and ***accept*** primitives complete the association {foreign address, foreign port}.

Stream Socket lifetime (cont.)

- Connection
  - Connection establishment is usually asymmetric, between a **server** and a **client**
  - The server ***binds*** a socket to a well known address and passively ***listens***, which means that he waits for a client to ***connect***
  - When a client connects, the server ***accepts*** the connection.
- Data transfer
  - When the two process are connected, data flow may begin between them
- Discard
  - When the communication ends, the sockets must be ***closed***, to enable the system to release resources, especially the bounded names (e.g. local ports) because they cannot be reused until they are available from any possible association.

Stream sockets: How does it work (synopsis)

<u>Client</u>	<u>Server</u>
▪ Create the socket	▪ Create the socket
	▪ Bind to local port
▪ Connect to Server	▪ Listen
	▪ Accept
▪ Communicate	▪ Communicate

Stream sockets: Java Example

- package java.net
- Classes
  - InetAddress
  - Socket
  - ServerSocket

Class **ServerSocket**

- Constructor
  - **ServerSocket**(int port)
  - **ServerSocket**(int port, int backlog)
  - **ServerSocket**(int port, int backlog, InetAddress bindAddr)
- Methods
  - Socket **accept**()
  - void **close**()
  - InetAddress **getInetAddress**()
  - int **getLocalPort**()

Class **Socket**

- Constructor
  - **Socket** (InetAddress address, int port)
  - **Socket** (String host, int port)
- Methods
  - InputStream **getInputStream**()
  - OutputStream **getOutputStream**()
  - void **close**()
  - InetAddress **getInetAddress**()
  - int **getLocalPort**()

Class **InetAddress**

- Methods
  - *static* InetAddress **getByName**(String host)
  - *static* InetAddress **getLocalHost**()
  - String **getHostAddress**()
  - String **getHostName**()

Java Client example

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

public class EchoClient {
    public static int serverPort = 12345;
    public static String serverHost = “levantes”;
    public static void main(String[] args) throws Exception {
            /* create the socket – connect to the server */
             Socket echoSocket = new Socket(serverHost, serverPort);

            /* create the input-output streams */
             PrintWriter out = new PrintWriter(echoSocket.getOutputStream(), true);
             BufferedReader in = new BufferedReader(
                         new InputStreamReader( echoSocket.getInputStream() ));
```

Java Client example (cont.)

```
            BufferedReader stdIn = new BufferedReader(
                                  new InputStreamReader(System.in));

            /* communicate */
            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
                      out.println(userInput);
                      System.out.println("echo: " + in.readLine());
            }

            /* release resources */
            out.close();
            in.close();
            echoSocket.close();
      }
}
```

Java Server example

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

public class EchoServer {
      public static String PORT = 12345;
      public static void main(String[] args) throws Exception {
            /* create socket */
            ServerSocket mainSocket = new ServerSocket(PORT);

            /* listen for incoming connection, accept it
             * and create a new Socket for this connection */
            Socket clientSocket = mainSocket.accept();
```

Java Server example (cont.)

```
/* create input and output streams for this connection */
 BufferedWriter out = new BufferedWriter(new
              OutputStreamWriter(clientSocket.getOutputStream()));
 BufferedReader in = new BufferedReader(new
              InputStreamReader(clientSocket.getInputStream()));

/* communicate */
int character=0;
while (character != -1) {
              character = in.read();
              out.write(character);
}
```

Java Server example (cont.)

```
                /* release resources */
                out.close();
                in.close();
                clientSocket.close();
                mainSocket.close();
        }
}
```

Datagram Socket lifetime

- Creation
  - Same as STREAM sockets
- Binding
  - Same as STREAM sockets
- Communication (send packet, receive packet)
  - In DGRAM sockets there is not a connection between the two process.
  - Each packet is sent independently, it may be routed differently and may arrive out-of-order with previous packets, it may arrive duplicate or it may not arrive at all.
  - If it arrives, the data are guaranteed that will not be corrupted.
- Discard
  - Same as STREAM sockets, all resources must be released in order to be available again.

Datagram sockets: How does it work (synopsis)

Client

- Create socket
- Create packet
- Send packet

Server

- Create socket
- Bind to local port
- Receive packet

Datagrams: Java Client example

- package java.net
- Classes:
  - DatagramPacket
  - DatagramSocket

Datagrams: Java Client example

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

public class DatagramClient{
      public static int MAXBUFLEN=1024;
      public static String serverHost = “levantes”;
      public static int serverPort = 12345;

      public static void main(String args[]) throws Exception{
                /* create the socket */
                DatagramSocket ds = new DatagramSocket();
```

Datagrams: Java Client example

```
            /* create the packet */
            InetAddress serverAddress = InetAddress.getByName(serverHost);
            byte[] buf = new byte[MAXBUFLEN];
            // … add to "buf" array the data we want to send
            DatagramPacket dp = new DatagramPacket(buf,
                                   buf.length, serverAddress, serverPort);

            /* send the packet */
            ds.send(dp);

            /* release resources */
            ds.close();
    }
}
```

Datagrams: Java Server example

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

public class DatagramServer{
    public static int MAXBUFLEN=1024;
    public static int port = 12345;

    public static void main(String args[]) throws Exception{
        /* create the socket – bind it to a local port */
        DatagramSocket ds = new DatagramSocket(port);
```

Datagrams: Java Server example

```
            /* create the buffer-packet where the received data
            *  will be written */
            byte[] buf = new byte[MAXBUFLEN];
            DatagramPacket dp = new DatagramPacket(buf, buf.length);

            /* receive the packet */
            ds.receive(dp);
            byte[] data = dp.getData();
            // ... do something with the data

            /* release resources */
            ds.close();
        }
    }
```

References

- http://java.sun.com/j2se/1.3/docs/api
- http://java.sun.com/docs/books/tutorial
- http://java.sun.com/docs/books/tutorial/networking/
- http://www.ecst.csuchico.edu/~beej/guide/net/
- http://www.csd.uoc.gr/~hy556/notes/Advanced-IPC.pdf