Distributed Systems

3. Access to the Transport Layer

Werner Nutt

Access to the Transport Layer

- Processes issue requests to the transport layer
  (i.e., the application takes the initiative, not the transport layer)
- Applications access the transportation layer via APIs
  – creation and manipulation of sockets

Access to the Transport Layer

2.1 Socket API for UDP

1. **Socket API for UDP**
2. Socket API for TCP

Java API for Internet Addresses

- Class **`InetAddress`**
  - uses DNS (Domain Name System)

```
InetAddress serverAdd =
 InetAddress.getByName("www.inf.unibz.it");
```

  - throws **`UnknownHostException`**
  - encapsulates details of IP address
    (4 bytes for IPv4 and 16 bytes for IPv6)

UDP Packet Structure

UDP = User Datagram Protocol

16-bit source port number	16-bit destination port number
16-bit UDP packet length	16-bit UDP checksum
Payload	

Java API for UDP

- Simple send/receive
  - with messages possibly lost/out of order

Payload (= array of bytes)	Payload length	Destination IP address	Destination Port

- Class **`DatagramPacket`**
  - packets may be transmitted between sockets
  - packets are truncated if too long
  - provides **`getData, getPort, getAddress, getLength`**

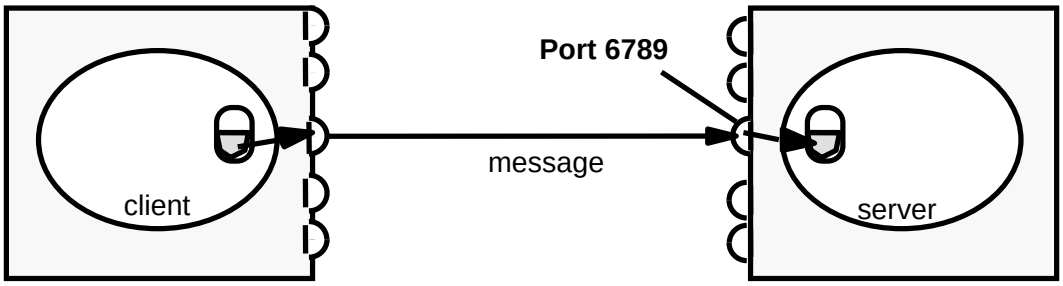
Java API for UDP Sockets

Class **`DatagramSocket`**

- *socket constructor*
  - bound to free port if no arg
  - arguments ***`InetAddress`***, ***`Port`***
- **`send`** a ***`DatagramPacket`***, non-blocking
- **`receive`** ***`DatagramPacket`***, blocking
- **`setSoTimeout`** (receive blocks for time T and throw ***`InterruptedIOException`***)
- **`close`** ***`DatagramSocket`***
- throws **`SocketException`** if port unknown or in use
- **`connect`** and **`disconnect`** (!!??)
- **`setReceiveBufferSize`** and **`setSendBufferSize`**

In the Following Example …

- UDP Client
  - sends a message and gets a reply
- UDP Server
  - repeatedly receives a request and sends it back to the client



See website of textbook for Java code (www.cdk4.net)

UDP Client Example

```
public class UDPClient{
public static void main(String args[]){
// args give message contents and server hostname
     DatagramSocket aSocket = null;
     try { aSocket = new DatagramSocket();
               byte [] m = args[0].getBytes();
               InetAddress aHost = InetAddress.getByName(args[1]);
               int serverPort = 6789;
               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);
          } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
} catch (IOException e){System.out.println("IO: " + e.getMessage());}
  finally {if (aSocket != null) aSocket.close(); }
}}
```


UDP Server Example

```
public class UDPServer{
    public static void main(String args[]){
          DatagramSocket aSocket = null;
          try {aSocket = new DatagramSocket(6789);
               byte[] buffer = new byte[1000];
               while(true) {
               DatagramPacket request = new DatagramPacket(buffer,
                                                                  buffer.length);
               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();}
}
```

Access to the Transport Layer

2.2 Socket API for TCP

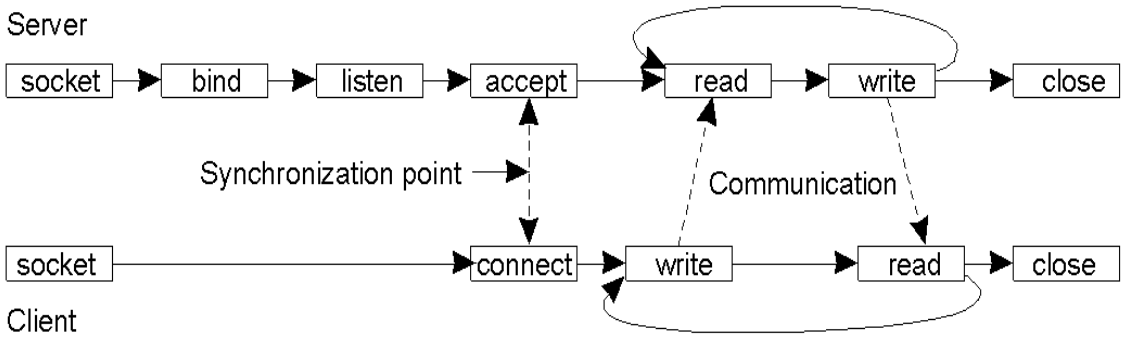
1. Socket API for UDP
2. **Socket API for TCP**

Socket Primitives for TCP/IP

System Calls	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Sockets appeared first in Berkeley UNIX as an interface to the transport layer

Life Cycle of Berkeley TCP Sockets



Java API for TCP

- Data stream abstraction
  - enables reliable transfer *(send can be blocking)*
  - marshaling/unmarshaling of data
  - access to TCP parameters: ReceiveBufferSize, SendBufferSize
- Classes **`Socket`** and **`ServerSocket`**
  - **`Socket`** asks for connection
  - **`ServerSocket`** listens and returns **`Socket`** when contacted
- Port numbers
  - explicit for **`ServerSocket`**, transparent for **`Socket`**

Java API for TCP

Class **`ServerSocket:`**

- **`bind`** to a **`SocketAddress`** if unbound
- **`accept`**: listen and return a **`Socket`**
  when a connection request arrives (blocking)
- **`close`**

Java API for TCP

Class **`Socket:`**

- **`connect`** to ***`SocketAddress`***
- **`getRemoteSocketAddress`** since that was chosen by the TCP system on the other side
- **`getInputStream`**, **`getOutputStream`**
  - use them for reading and writing
  - which is/may be blocking
- **`DataInputStream`**, **`DataOutputStream`**:
  - wrapper classes for streams
  - have methods for marshaling/ unmarshaling
- **`isConnected`**
- **`close`**

TCP Client Example

```
public class TCPClient {
    public static void main (String args[]) {
    // arguments supply message and hostname of destination
        Socket s = null;
        try{ int serverPort = 7896;
             s = new Socket(args[1], serverPort);
             DataInputStream in = new DataInputStream( s.getInputStream());
             DataOutputStream out = new DataOutputStream(
                                                    s.getOutputStream());
              out.writeUTF(args[0]); // UTF is a string encoding
              String data = in.readUTF();
              System.out.println("Received: "+ data) ;
              s.close();
            } catch (UnknownHostException e){
                                    System.out.println("Sock: "+e.getMessage());
            } catch (EOFException e){System.out.println("EOF: "+e.getMessage());
            } catch (IOException e){System.out.println("IO: "+e.getMessage());}
             } finally {if(s!=null} try {s.close();} catch (IOException e)….}
}
```


TCP Server Example

```
public class TCPServer {

    public static void main (String args[]) {
            try{
                int serverPort = 7896;
                ServerSocket listenSocket = new ServerSocket(serverPort);
                while(true) {
                    Socket clientSocket = listenSocket.accept();
                    Connection c = new Connection(clientSocket);
                }
             } catch(IOException e) {System.out.println("Listen: " +
                                                    e.getMessage());}
    }
}

// this figure continues on the next slide
```

Example Server (cntd.)

```
class Connection extends Thread {
    DataInputStream in;
    DataOutputStream out;
    Socket clientSocket;
    public Connection (Socket aClientSocket) {
        try {
            clientSocket = aClientSocket;
            in = new DataInputStream(clientSocket.getInputStream());
            out = new DataOutputStream( clientSocket.getOutputStream());
            this.start();
        } catch(IOException e) {System.out.println("Connection: "+e.getMessage());}
    }
    public void run(){
        try { // an echo server
            String data = in.readUTF();
            out.writeUTF(data);
        } catch(EOFException e) {System.out.println("EOF: "+e.getMessage());
                } catch(IOException e) {System.out.println("IO:s a"+e.getMessage());}
        } finally {try {clientSocket.close();}catch (IOException e)…..}
}
```


References

In preparing the lectures I have used several sources.

The main ones are the following:

Books:

- Coulouris, Dollimore, Kindberg. Distributed Systems – Concepts and Design (CDK)
- Kurose/Ross. Computer Networking: A Top-Down Approach

Slides:

- Kurose/Ross, Material for lecturers
- Andrew Tanenbaum, Slides from his website
- CDK Website