JAVA Socket Programming

What is a socket?

- Socket
  - The combination of an IP address and a port number. (RFC 793 ,original TCP specification)
  - The name of the Berkeley-derived *application programming interfaces* (APIs) for applications using TCP/IP protocols.
  - Two types
    - Stream socket : reliable two-way connected communication streams
    - Datagram socket

- Socket pair
  - Specified the two end points that uniquely identifies each TCP connection in an internet.
  - 4-tuple: (client IP address, client port number, server IP address, server port number)

Client-server applications

- Implementation of a protocol standard defined in an RFC. (FTP, HTTP, SMTP…)
  - Conform to the rules dictated by the RFC.
  - Should use the port number associated with the protocol.

- Proprietary client-server application.
  - A single developer(or team) creates both client and server program.
  - The developer has complete control.
  - Must be careful not to use one of the well-known port number defined in the RFCs.

  * well-known port number : managed by the Internet Assigned Numbers Authority(IANA)

Socket Programming with TCP

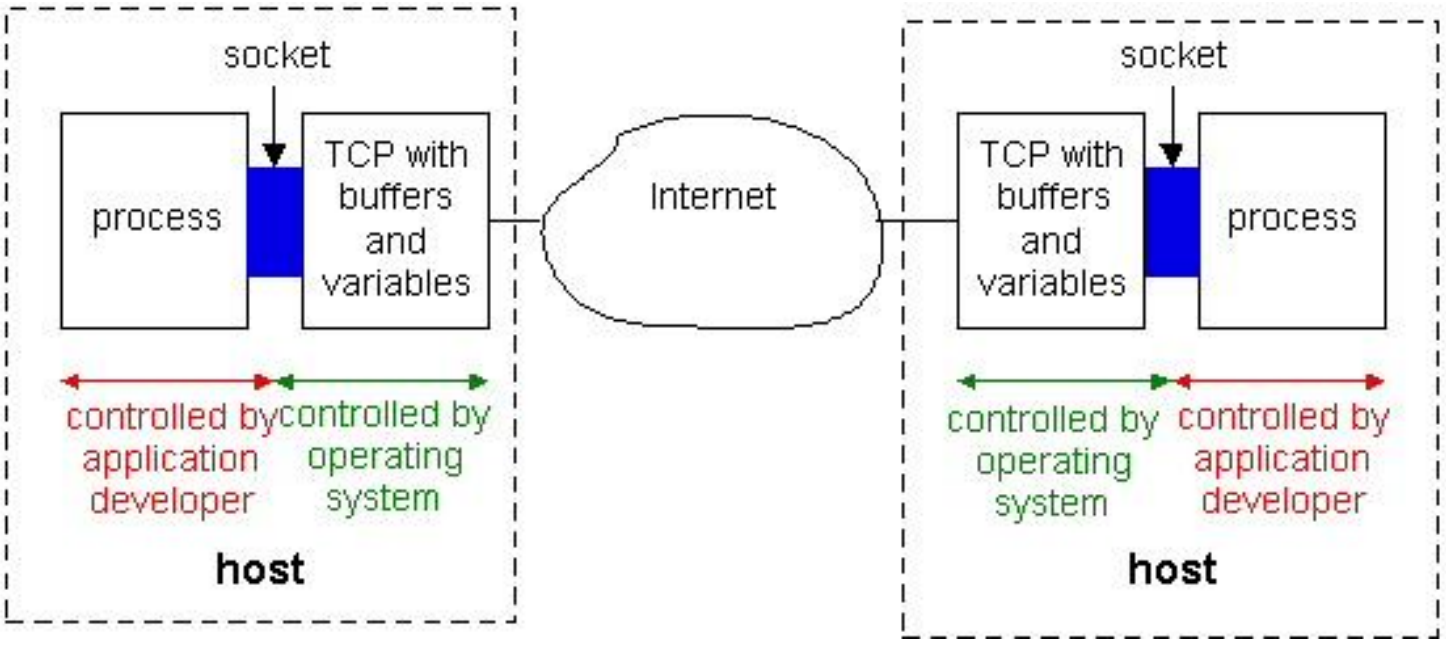


Figure 2.6-1: Processes communicating through TCP sockets

The application developer has the ability to fix a few TCP parameters, such as maximum buffer and maximum segment sizes.

Sockets for server and client

- Server
  - Welcoming socket
    - Welcomes some initial contact from a client.
  - Connection socket
    - Is created at initial contact of client.
    - New socket that is dedicated to the particular client.
- Client
  - Client socket
    - Initiate a TCP connection to the server by creating a socket object. (Three-way handshake)
    - Specify the address of the server process, namely, the IP address of the server and the port number of the process.

Socket functional calls

- socket (): Create a socket
- bind(): bind a socket to a local IP address and port #
- listen(): passively waiting for connections
- connect(): initiating connection to another socket
- accept(): accept a new connection
- Write(): write data to a socket
- Read(): read data from a socket
- sendto(): send a datagram to another UDP socket
- recvfrom(): read a datagram from a UDP socket
- close(): close a socket (tear down the connection)

Sockets

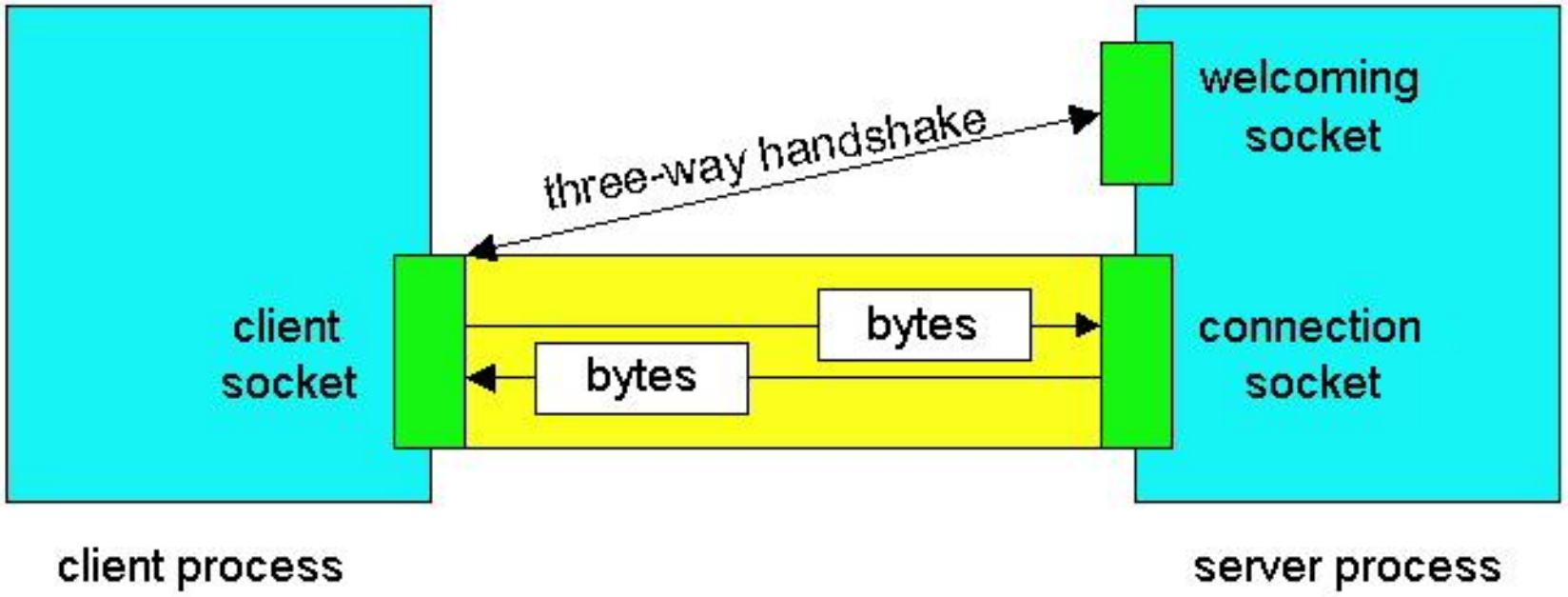
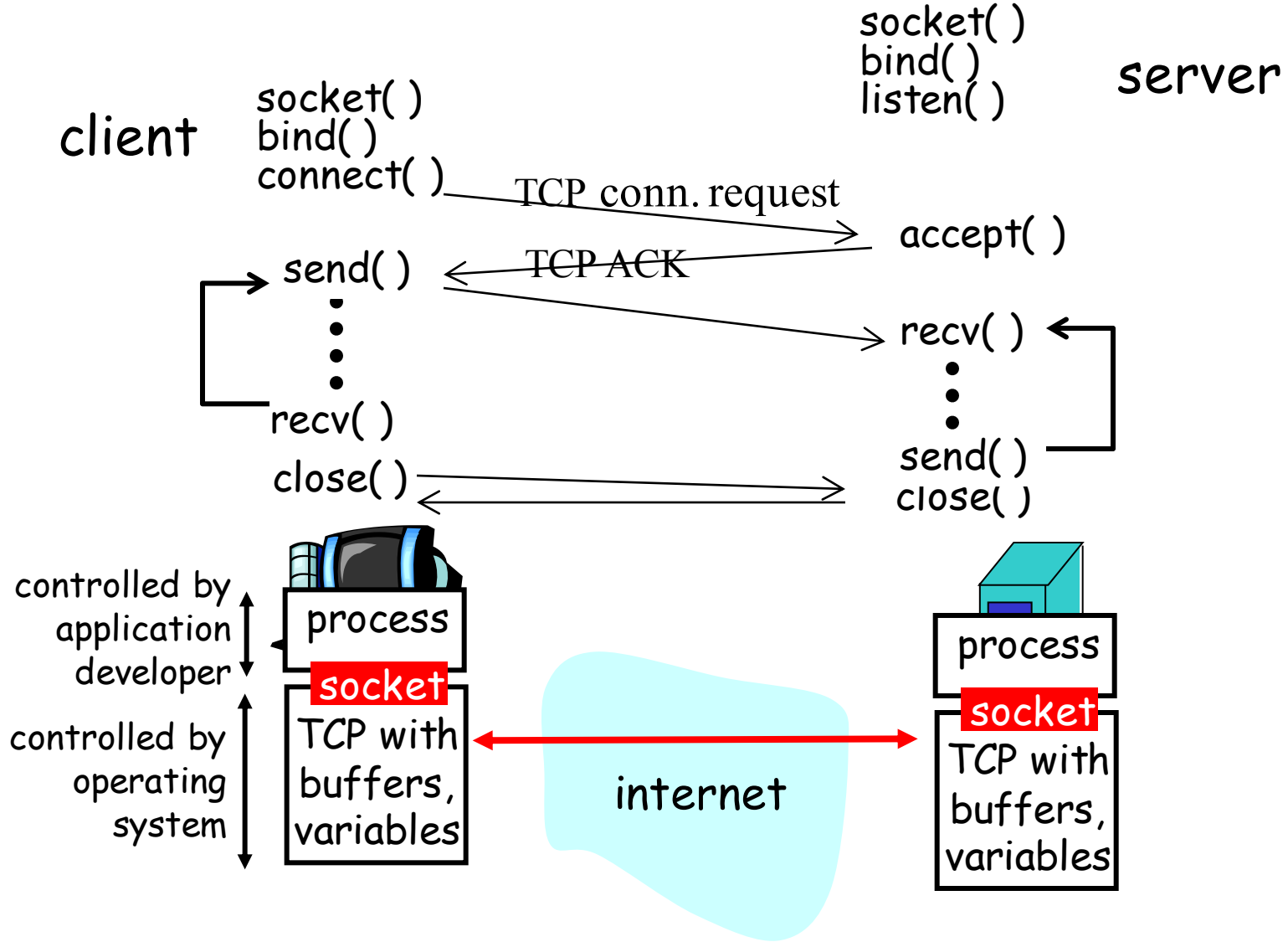


Figure 2.6-2: Client socket, welcoming socket and connection socket

Socket-programming using TCP

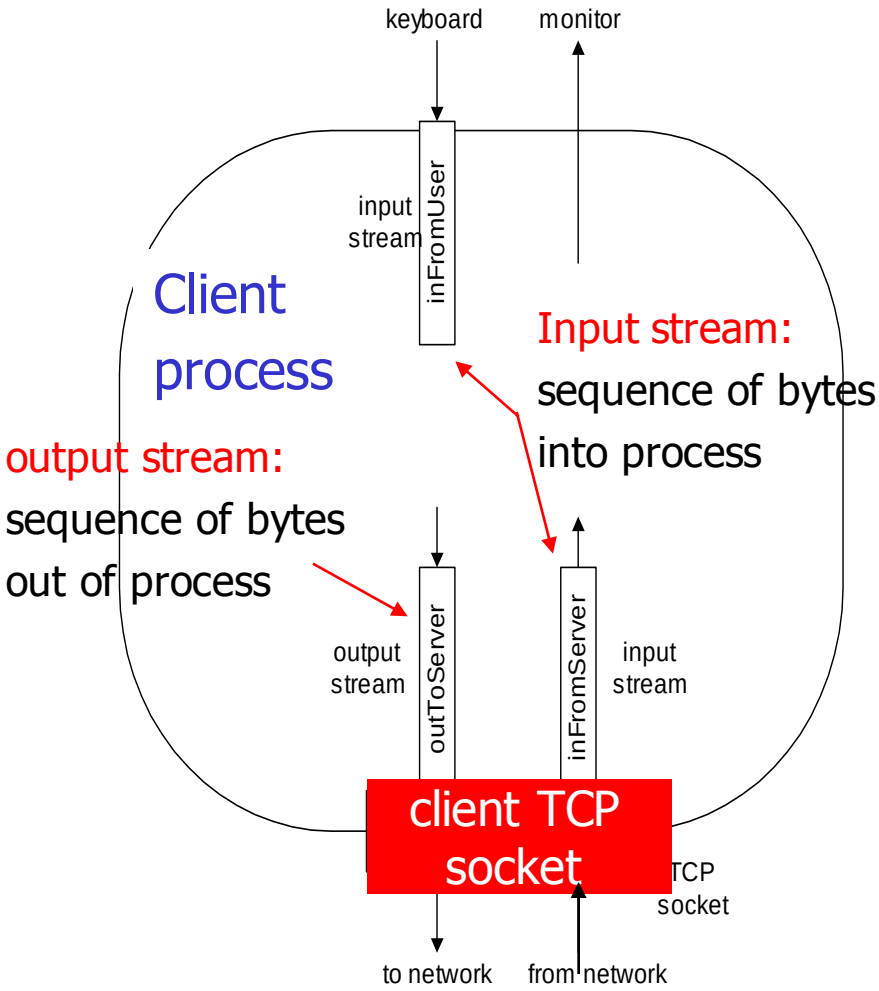
TCP service: reliable byte stream transfer



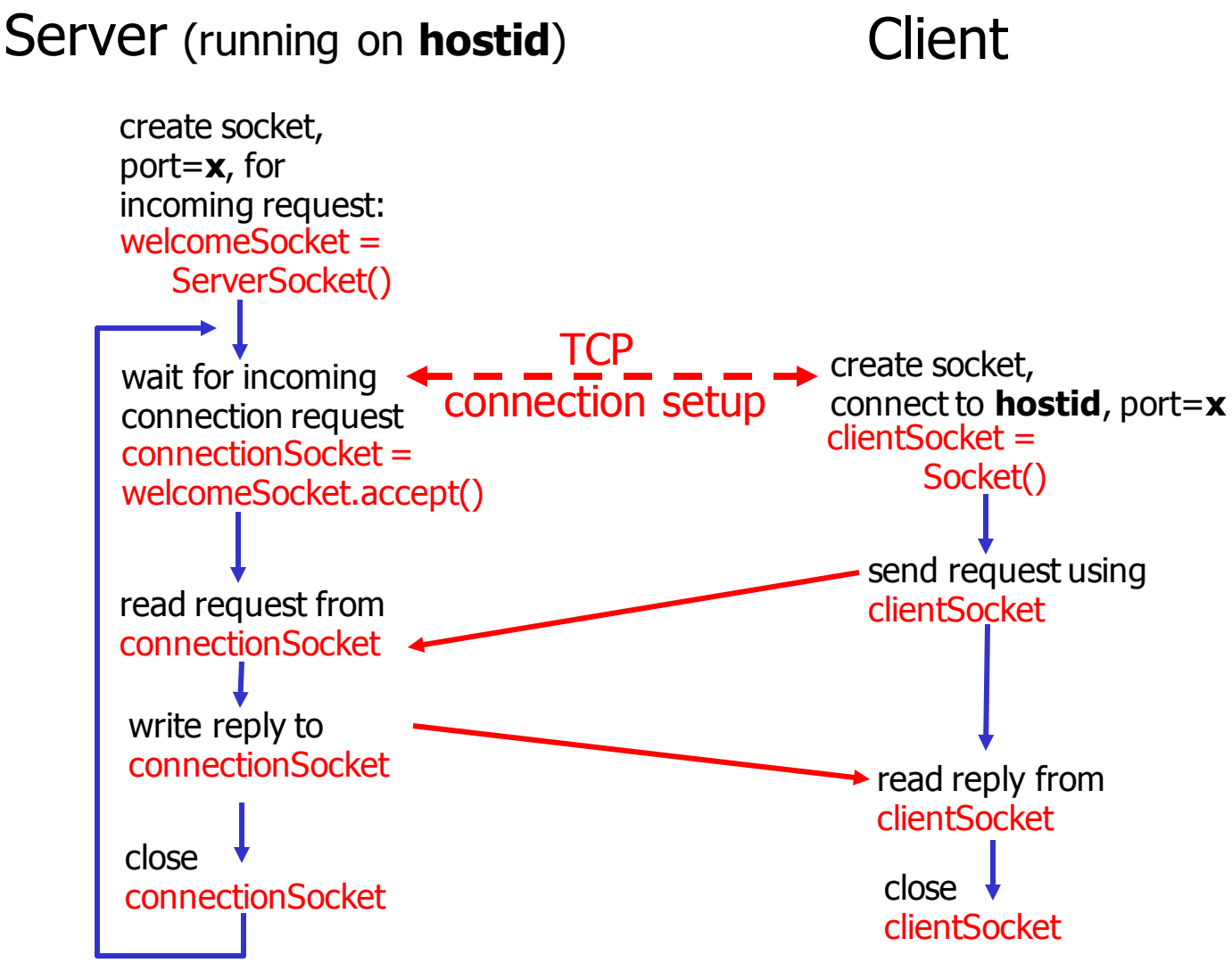
Socket programming with TCP

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)



Client/server socket interaction: TCP



JAVA TCP Sockets

- In Package java.net
  - java.net.Socket
    - Implements client sockets (also called just "sockets").
    - An endpoint for communication between two machines.
    - Constructor and Methods
      - Socket(String host, int port): Creates a stream socket and connects it to the specified port number on the named host.
      - InputStream getInputStream()
      - OutputStream getOutputStream()
      - close()
  - java.net.ServerSocket
    - Implements server sockets.
    - Waits for requests to come in over the network.
    - Performs some operation based on the request.
    - Constructor and Methods
      - ServerSocket(int port)
      - Socket Accept(): Listens for a connection to be made to this socket and accepts it. This method blocks until a connection is made.

TCPClient.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());
```

TCPClient.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();

    }
}
```

TCPServer.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()));
```

TCPServer.java

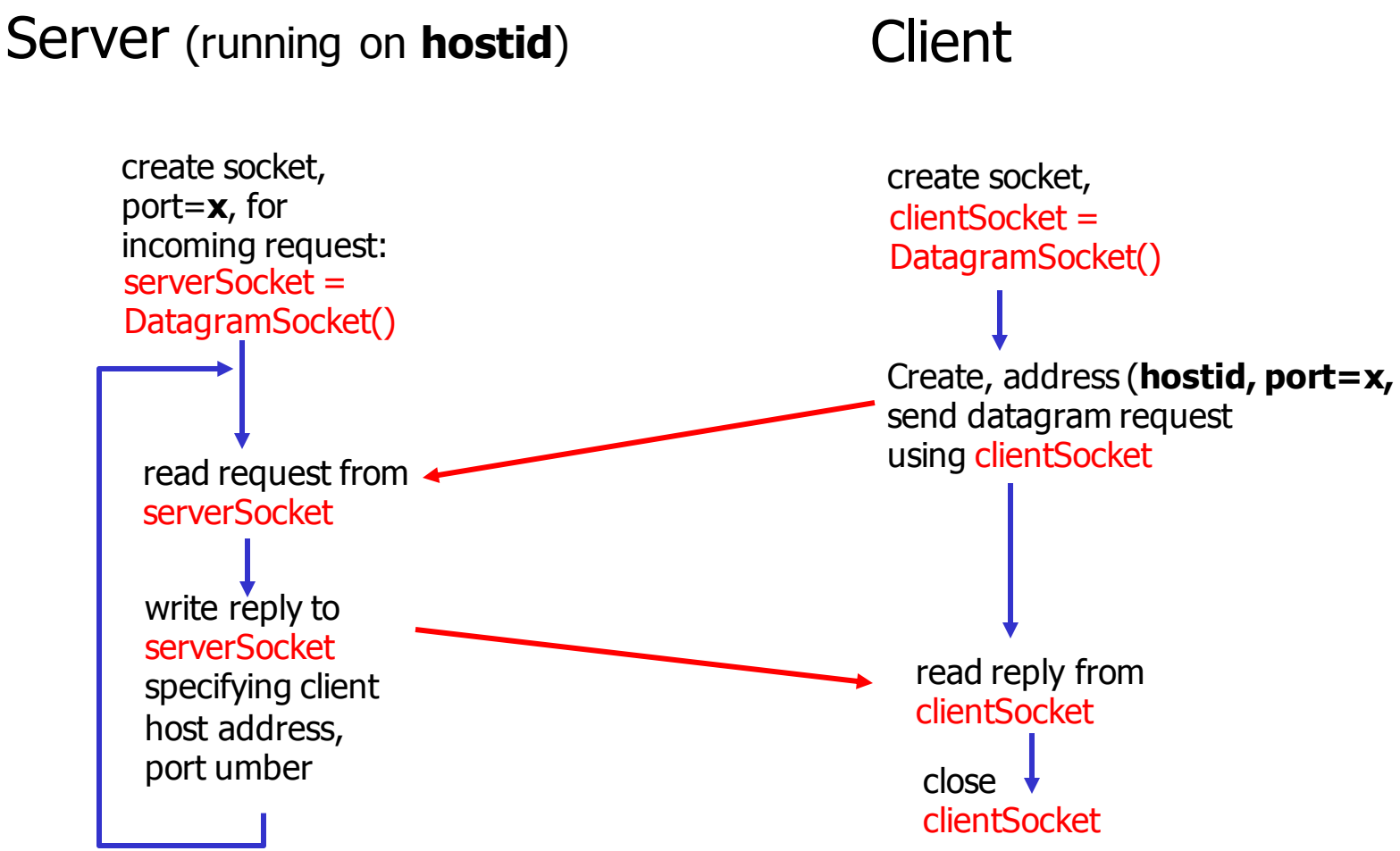
```
        DataOutputStream  outToClient =
             new DataOutputStream(connectionSocket.getOutputStream());

        clientSentence = inFromClient.readLine();

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

        outToClient.writeBytes(capitalizedSentence);

      }
    }
}
```

Socket Programming with UDP

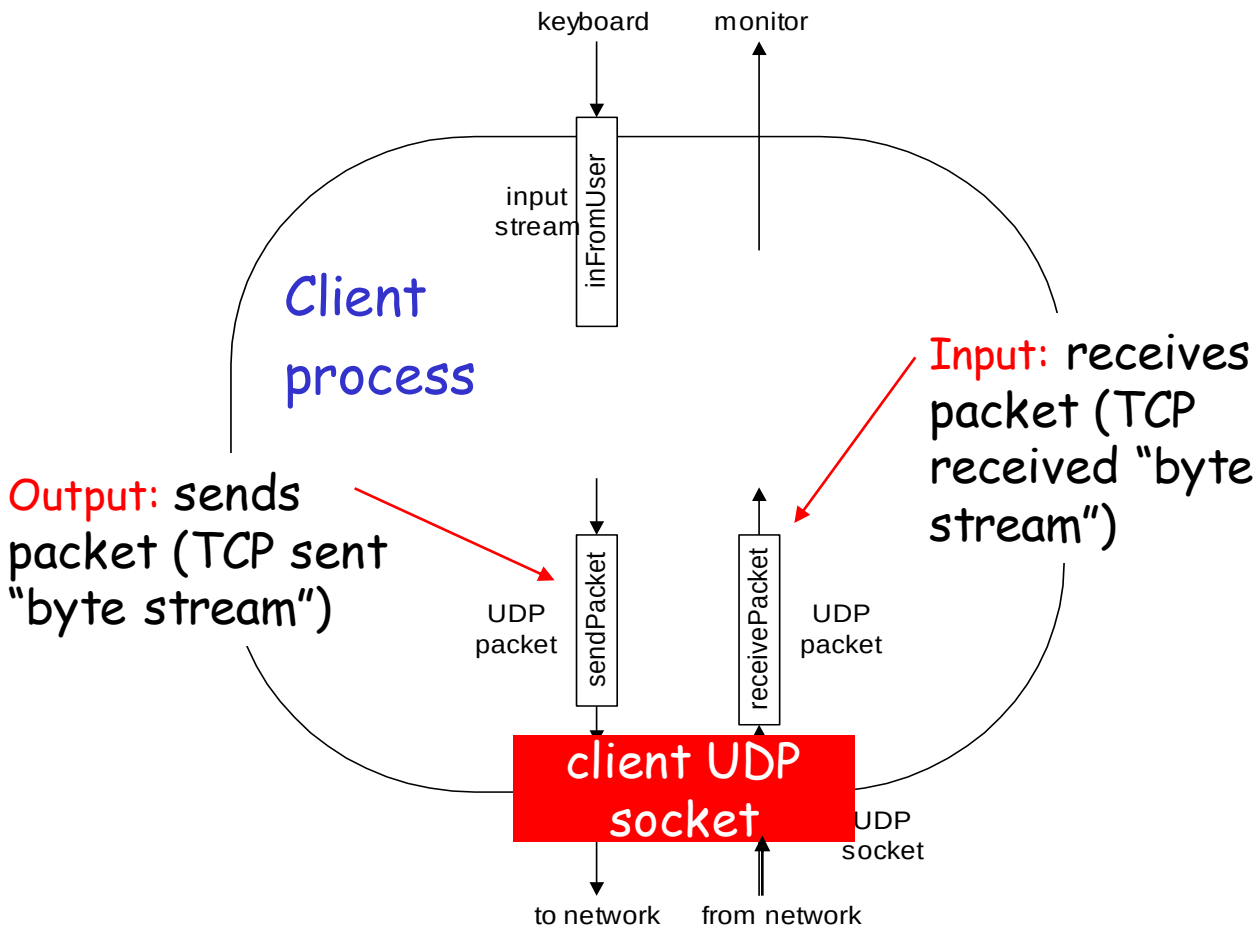
- UDP
  - Connectionless and unreliable service.
  - There isn't an initial handshaking phase.
  - Doesn't have a pipe.
  - transmitted data may be received out of order, or lost

- Socket Programming with UDP
  - No need for a welcoming socket.
  - No streams are attached to the sockets.
  - the sending hosts creates "packets" by attaching the IP destination address and port number to each batch of bytes.
  - The receiving process must unravel to received packet to obtain the packet's information bytes.

Client/server socket interaction: UDP



Example: Java client (UDP)



JAVA UDP Sockets

- In Package java.net
  - java.net.DatagramSocket
    - A socket for sending and receiving datagram packets.
    - Constructor and Methods
      - DatagramSocket(int port): Constructs a datagram socket and binds it to the specified port on the local host machine.
      - void receive(DatagramPacket p)
      - void send(DatagramPacket p)
      - void close()

UDPClient.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();
```

UDPClient.java

```
 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();

    }
}
```

UDPServer.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());
```

UDPServer.java

```
        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);

      }
   }
}
```

Building a Simple Web Server

- Handles only one HTTP request
- Accepts and parses the HTTP request
- Gets the required file from the server's file system.
- Creates an HTTP response message consisting of the requested file preceded by header lines
- Sends the response directly to the client

WebServer.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class WebServer{
    public static void main(String argv[]) throws Exception  {
          String requestMessageLine;
          String fileName;
          ServerSocket listenSocket = new ServerSocket(6789);
          Socket connectionSocket = listenSocket.accept();

          BufferedReader inFromClient =
                new BufferedReader(new
                      InputStreamReader(connectionSocket.getInputStream()));

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

WebServer.java

requestMessageLine = inFromClient.readLine();

```
StringTokenizer tokenizedLine =
    new StringTokenizer(requestMessageLine);

if (tokenizedLine.nextToken().equals("GET")){
        fileName = tokenizedLine.nextToken();
        if (fileName.startsWith("/") == true )
                fileName  = fileName.substring(1);

       File file = new File(fileName);
        int numOfBytes = (int) file.length();
         FileInputStream inFile  = new FileInputStream (fileName);
         byte[] fileInBytes = new byte[numOfBytes];

       inFile.read(fileInBytes);
```

WebServer.java

```
    outToClient.writeBytes("HTTP/1.0 200 Document Follows\r\n");

    if (fileName.endsWith(".jpg"))
        outToClient.writeBytes("Content-Type: image/jpeg\r\n");

    if (fileName.endsWith(".gif"))
        outToClient.writeBytes("Content-Type: image/gif\r\n");

    outToClient.writeBytes("Content-Length: " + numOfBytes + "\r\n");

    outToClient.writeBytes("\r\n");
    outToClient.write(fileInBytes, 0, numOfBytes);
     connectionSocket.close();
   }
    else System.out.println("Bad Request Message");
  }
}
```

Concurrent server

- Servers need to handle a new connection request while processing previous requests.
  - Most TCP servers are designed to be concurrent.
- When a new connection request arrives at a server, the server accepts and invokes a new process to handle the new client.

How to handle the port numbers

```
cosmos% netstat –a –n –f inet
Active Internet connections (including servers)
Proto     Recv-Q    Send-Q    Local Address          Foreign Address        (state)
tcp       0         0         *.23                   *.*                    LISTEN

cosmos%  netstat –a –n –f inet
Proto     Recv-Q    Send-Q    Local Address          Foreign Address        (state)
tcp       0         0         192.249.24.2.23        192.249.24.31.1029     ESTABLISHED
tcp       0         0         *.23                   *.*                    LISTEN

cosmos%  netstat –a –n –f inet
Proto     Recv-Q    Send-Q    Local Address          Foreign Address        (state)
tcp       0         0         192.249.24.2.23        192.249.24.31.1029     ESTABLISHED
tcp       0         0         192.249.24.2.23        192.249.24.31.1030     ESTABLISHED
tcp       0         0         *.23                   *.*                    LISTEN
```

Socket programming: references

C-language tutorial (audio/slides):

- "Unix Network Programming" (J. Kurose),

http://manic.cs.umass.edu/~amldemo/courseware/intro.html

Java-tutorials:

- "All About Sockets" (Sun tutorial), http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html
- "Socket Programming in Java: a tutorial," http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html