CIS430/530-INFORMATION TECHNOLOGY

SOCKET PROGRAMMING

SOCKET PROGRAMMING

- Socket programming with TCP

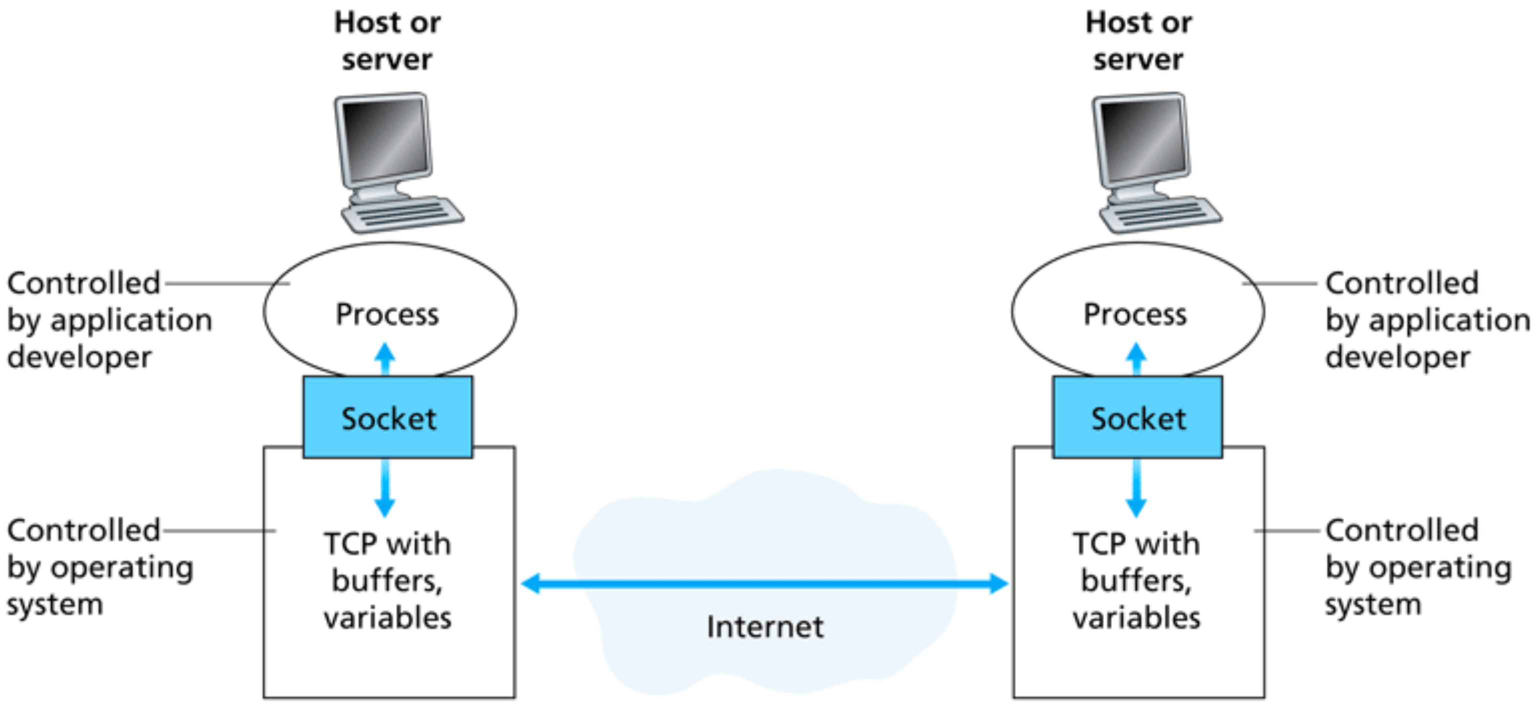

Host or server
Host or server
Controlled by application developer
Process
Socket
Controlled by operating system
TCP with buffers, variables
Internet
Process
Socket
TCP with buffers, variables
Controlled by application developer
Controlled by operating system

Figure 2.3 ♦ Application processes, sockets, and underlying transport protocol

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:
  - UDP
  - TCP

socket

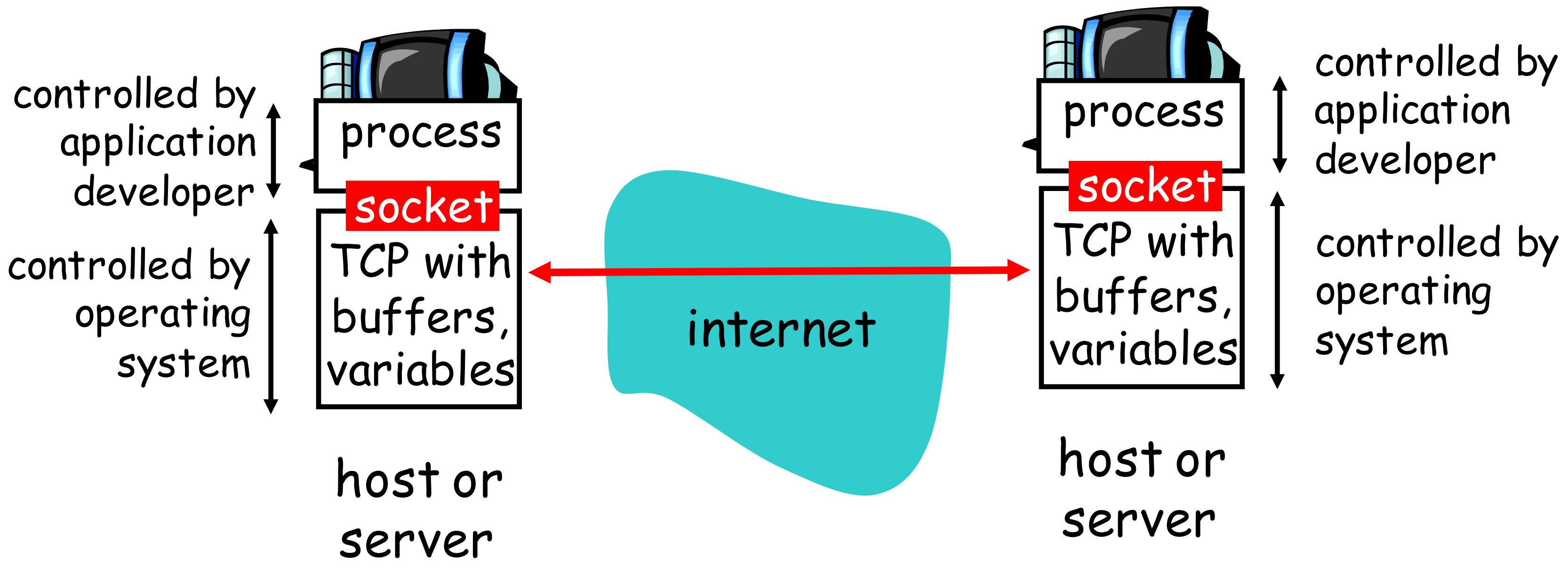
A *application-created, OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

SOCKET PROGRAMMING BASICS

- Server must be running before client can send anything to it.
- Server must have a socket (door) through which it receives and sends segments
- Similarly client needs a socket
- Socket is locally identified with a port number
  - Analogous to the apt # in a building
- Client needs to know server IP address and socket port number.

SOCKET-PROGRAMMING USING TCP

TCP service: reliable transfer of **bytes** from one process to another


controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
host or server
controlled by application developer
controlled by operating system

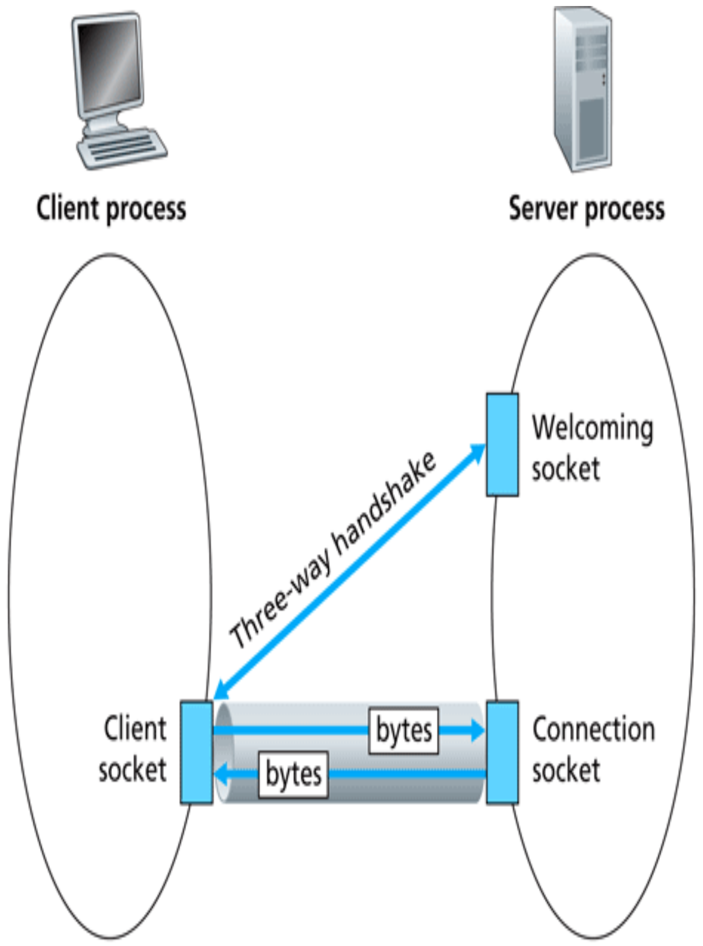


Figure 2.31 ♦ Client-socket, welcoming socket, and connection socket

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

SOCKET PROGRAMMING *WITH TCP*

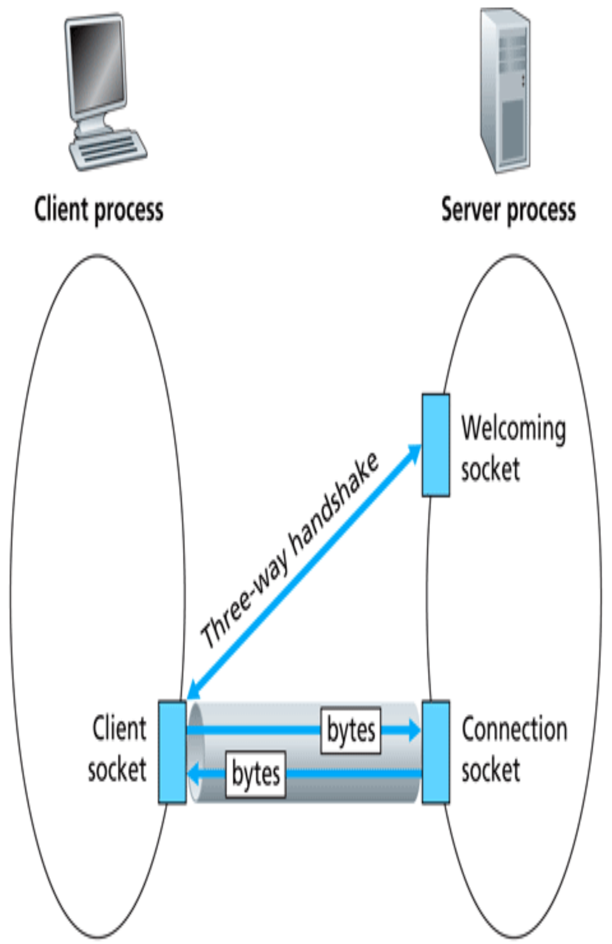


Figure 2.31 ♦ Client-socket, welcoming socket, and connection socket

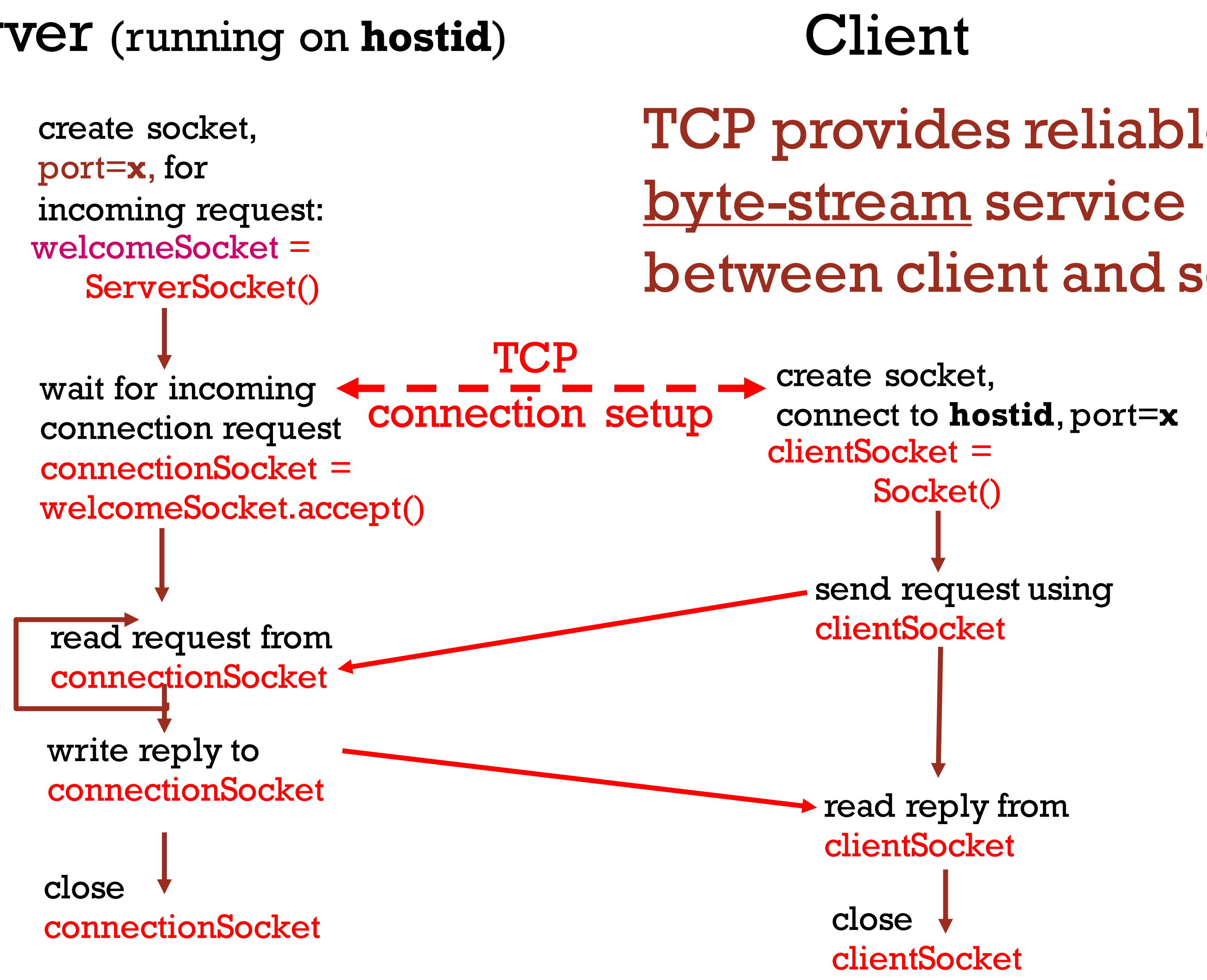
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more later)

application viewpoint

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

CLIENT/SERVER SOCKET INTERACTION: TCP

TCP provides reliable <u>byte-stream</u> service between client and server



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()

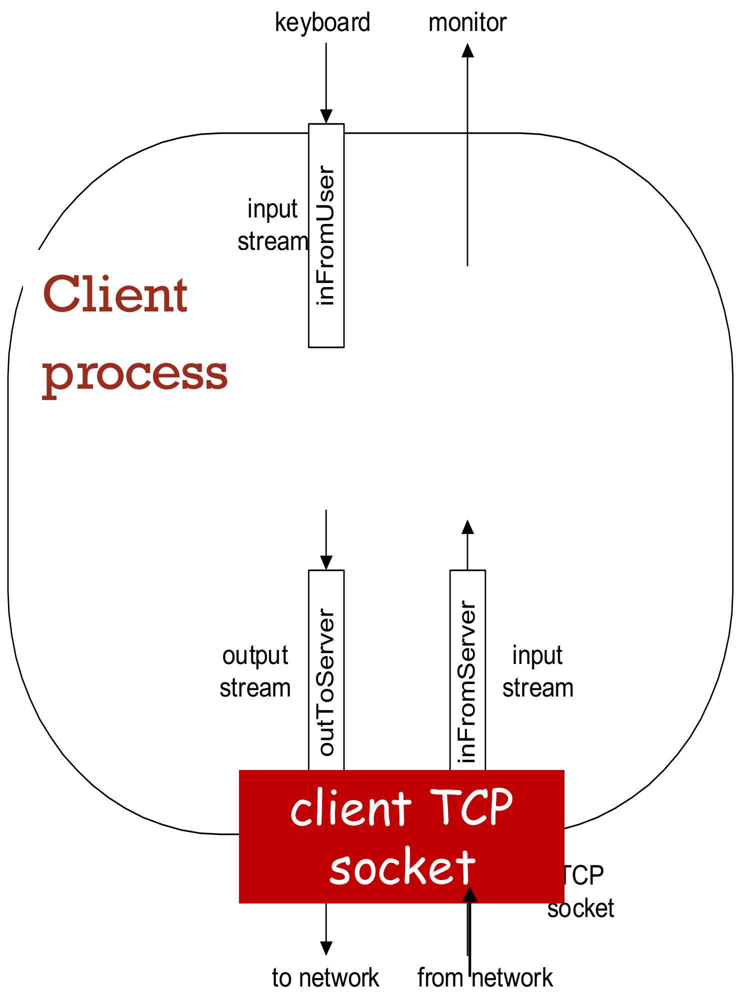
send request using clientSocket

read reply from clientSocket

close clientSocket

Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



SOCKET PROGRAMMING WITH TCP

Example client-server app:

1) client reads line from standard input (`inFromUser` stream) , sends to server via socket (`outToServer` stream)

2) server reads line from socket

3) server converts line to uppercase, sends back to client

4) client reads, prints modified line from socket (`inFromServer` stream)

EXAMPLE: JAVA CLIENT (TCP)

```
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());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

EXAMPLE: JAVA CLIENT (TCP), CONT.

Create input stream attached to socket

```
    BufferedReader inFromServer =
      new BufferedReader(new
      InputStreamReader(clientSocket.getInputStream()));

    sentence = inFromUser.readLine();
```

Send line to server

```
    outToServer.writeBytes(sentence + '\n');
```

Read line from server

```
    modifiedSentence = inFromServer.readLine();

    System.out.println("FROM SERVER: " + modifiedSentence);

    clientSocket.close();

  }
}
```

EXAMPLE: JAVA SERVER (TCP)

```
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()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

EXAMPLE: JAVA SERVER (TCP), CONT

Create output stream, attached to socket

Read in line from socket

Write out line to socket

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

clientSentence = inFromClient.readLine();

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

outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

End of while loop, loop back and wait for another client connection

TCP OBSERVATIONS & QUESTIONS

- Server has two types of sockets:
  - welcomeSocket and connectionSocket
- When client knocks on serverSocket's "door," server creates connectionSocket and completes TCP conx.
- Dest IP and port are not explicitly attached to segment.
- Can multiple clients use the server?

SOCKET PROGRAMMING *WITH UDP*

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination to each segment
- OS attaches IP address and port of sending socket to each segment
- Server can extract IP address, port of sender from received segment

application viewpoint

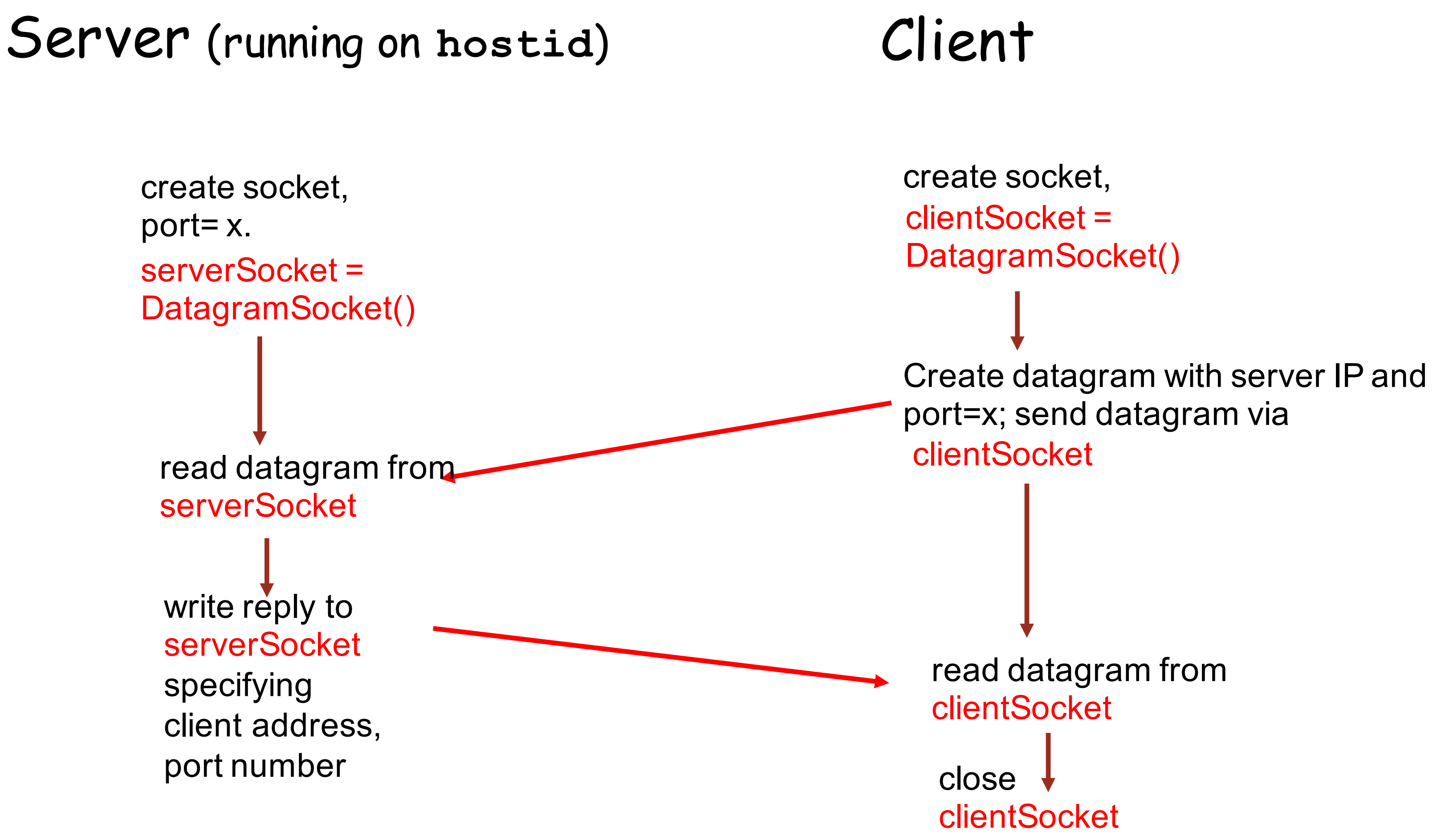
UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Note: the official terminology for a UDP packet is "datagram". In this class, we instead use "UDP segment".

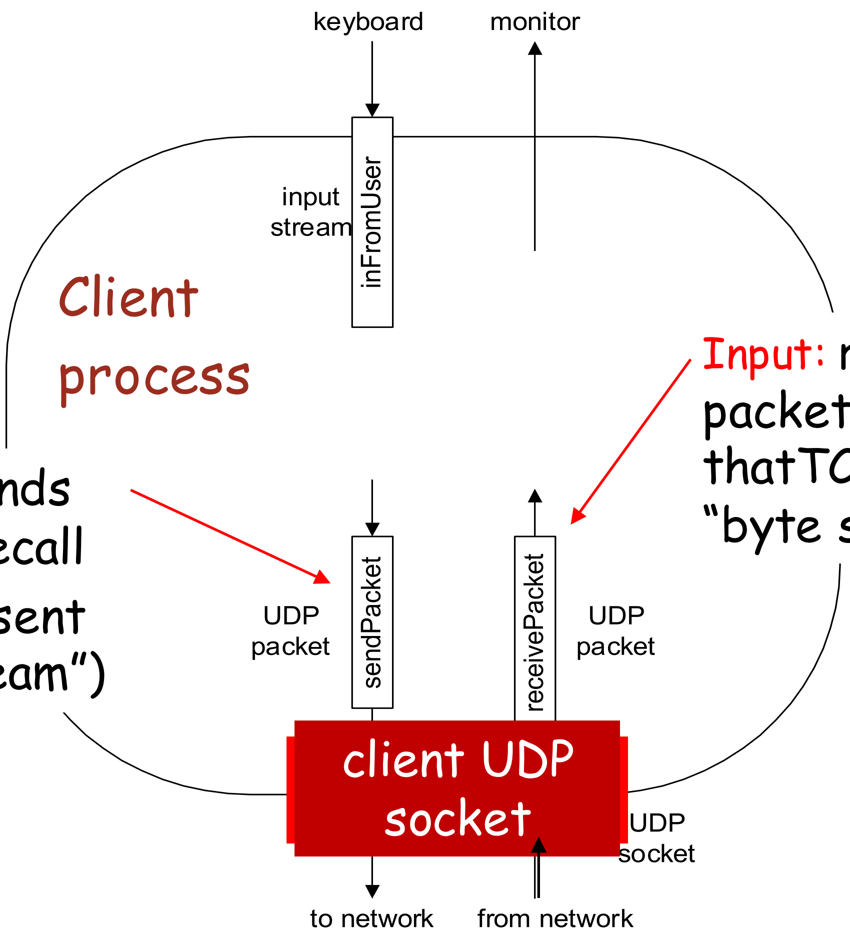
RUNNING EXAMPLE

- Client:
  - User types line of text
  - Client program sends line to server
- Server:
  - Server receives line of text
  - Capitalizes all the letters
  - Sends modified line to client
- Client:
  - Receives line of text
  - Displays

CLIENT/SERVER SOCKET INTERACTION: UDP


Server (running on hostid)
Client
create socket, port= x. serverSocket = DatagramSocket()
create socket, clientSocket = DatagramSocket()
Create datagram with server IP and port=x; send datagram via clientSocket
read datagram from serverSocket
write reply to serverSocket specifying client address, port number
read datagram from clientSocket
close clientSocket

EXAMPLE: JAVA CLIENT (UDP)



Output: sends packet (recall that TCP sent "byte stream")

Input: receives packet (recall thatTCP received "byte stream")

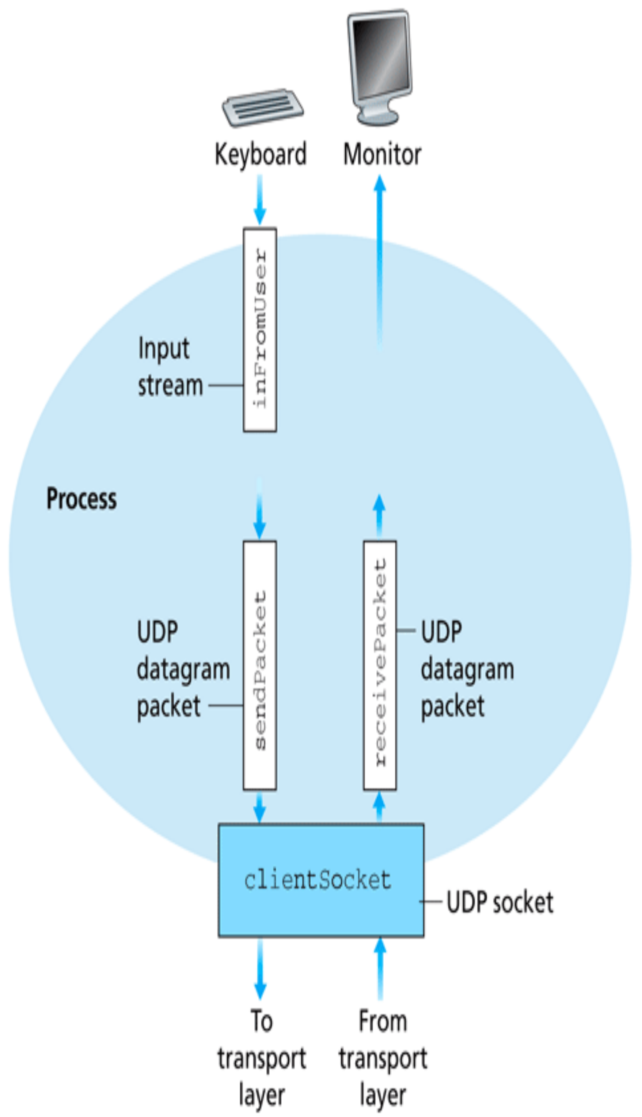


Figure 2.35 ♦ UDPClient has one stream; the socket accepts packets from the process and delivers packets to the process.

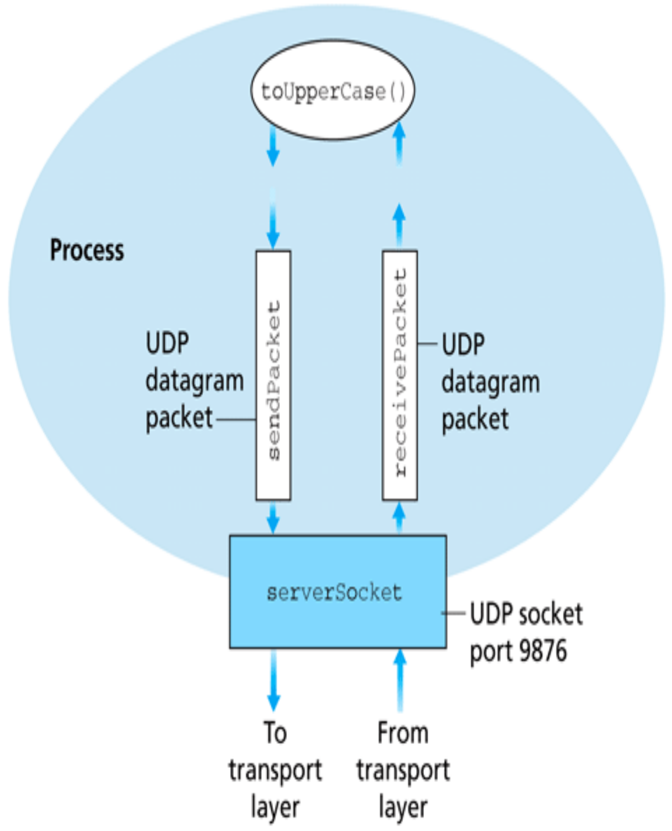


Figure 2.36 ♦ UDPServer has no streams; the socket accepts packets from the process and delivers packets to the process.

EXAMPLE: JAVA CLIENT (UDP)

Create input stream

Create client socket

Translate hostname to IP address using DNS

```
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();
```

EXAMPLE: JAVA CLIENT (UDP), CONT.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
    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 Server (UDP)

```
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);
```

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

EXAMPLE: JAVA SERVER (UDP), CONT

```
         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);
        }
     }
  }
```

Get IP addr port #, of sender

Create datagram to send to client

Write out datagram to socket

End of while loop, loop back and wait for another datagram

UDP OBSERVATIONS & QUESTIONS

- Both client server use DatagramSocket
- Dest IP and port are explicitly attached to segment.
- Can the client send a segment to server without knowing the server's IP address and/or port number?
- Can multiple clients use the server?

SUMMARY

- Socket programming with TCP
- Socket programming with UDP