Socket Programming

- Socket Programming Overview
  - Socket Programming with TCP
  - Socket Programming with UDP
- Python Socket Programming
- Java Socket Programming

Readings: Chapter 2: Sections 2.7

Recap: Client-Server Communication Paradigm

Typical network app has two pieces: *client* and *server*

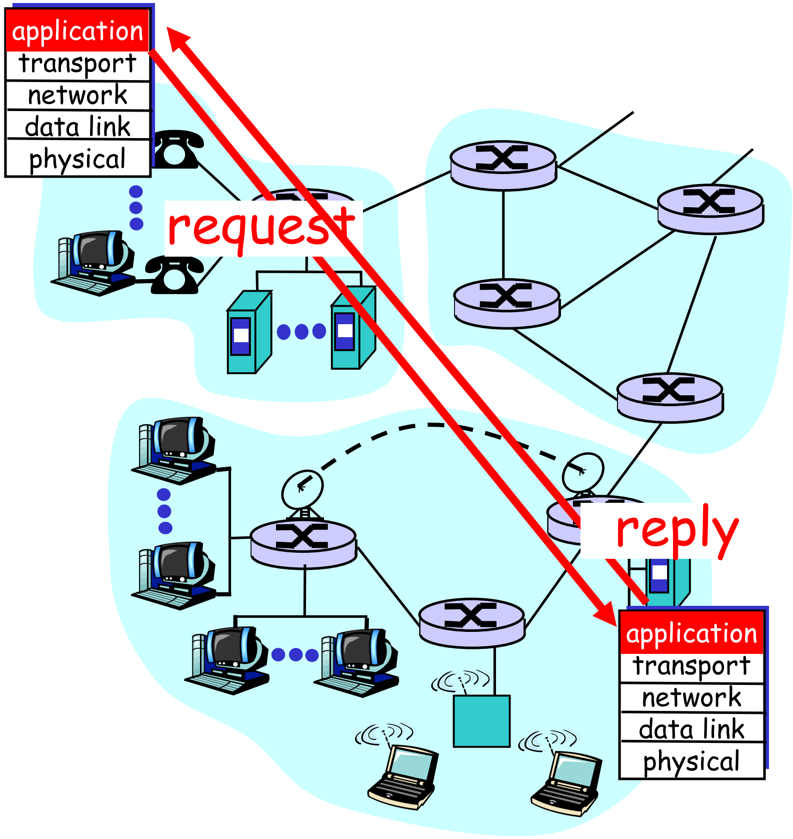
Client:

initiates contact with server ("speaks first")

typically requests service from server

Server:

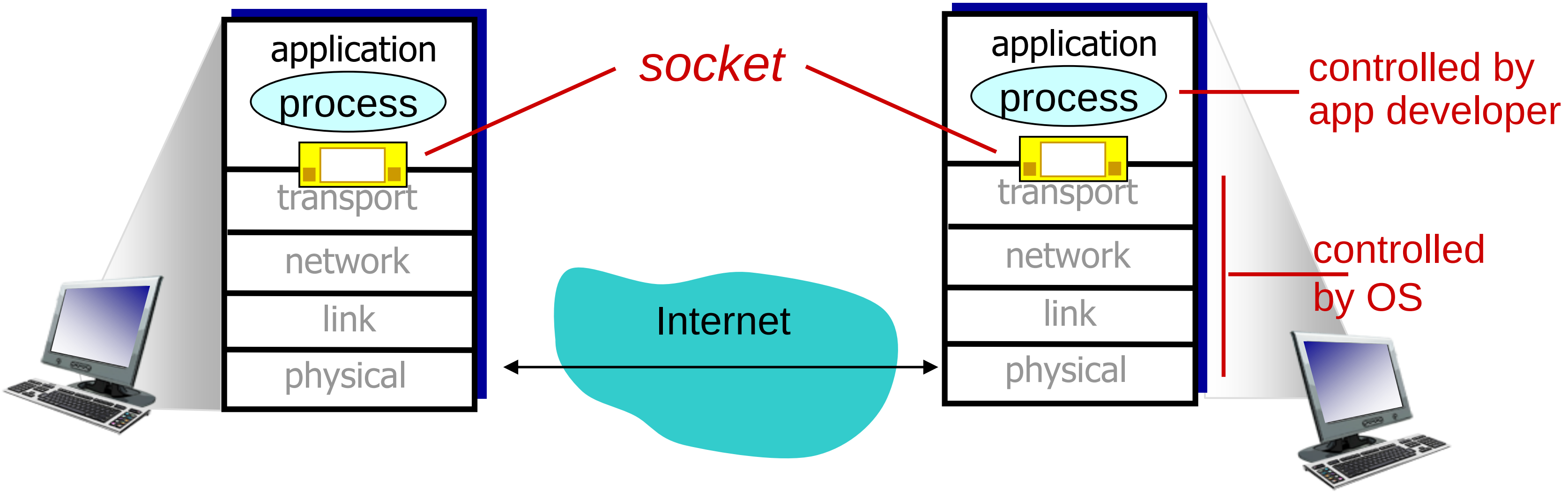
provides requested service to client



Socket Programming

Goal: learn how to build client/server applications that communicate using sockets

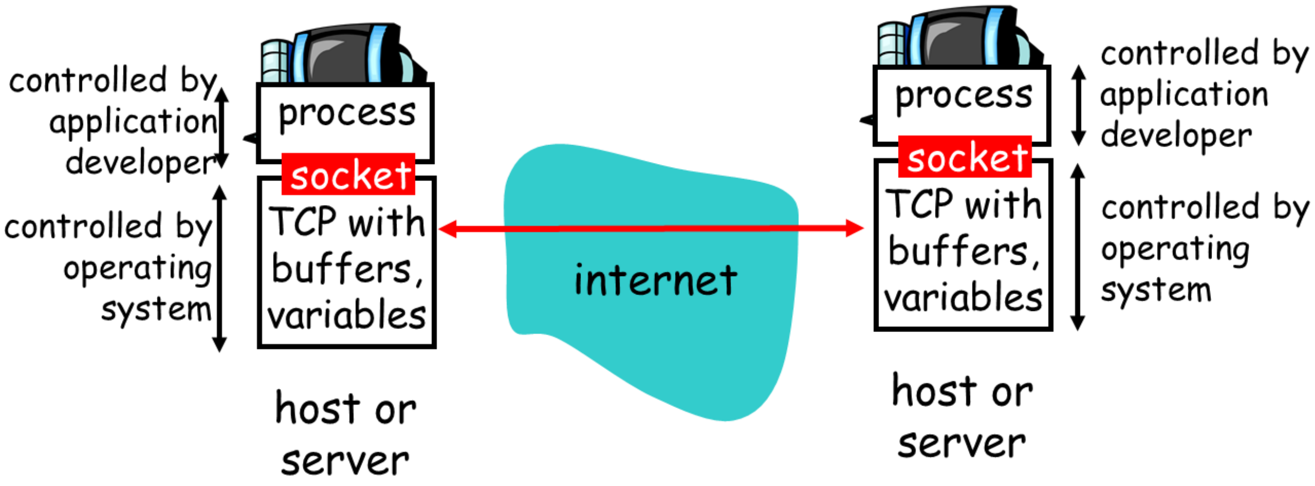
Socket: door between application process and end-end-transport protocol



Socket Programming

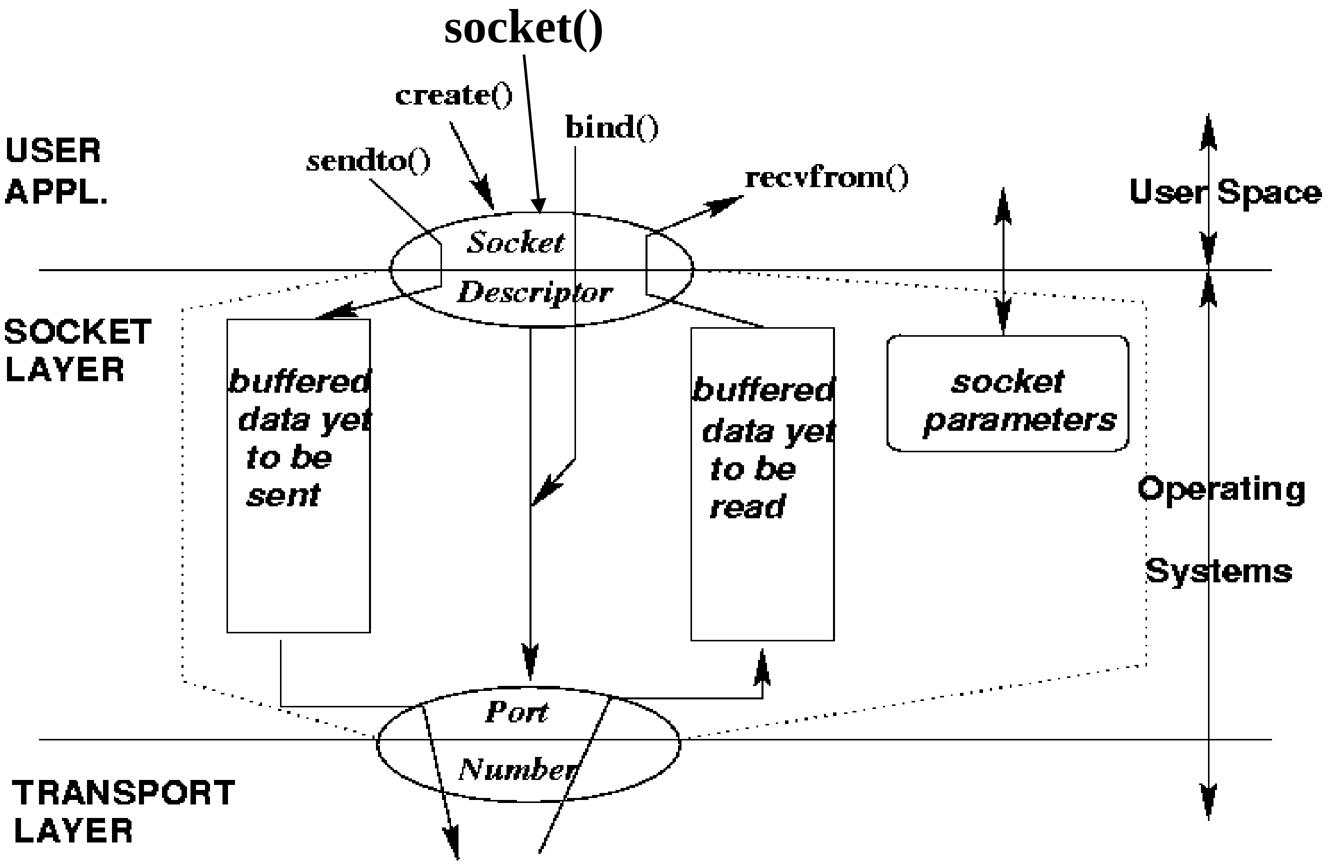
Socket:

- Interface between the application layer and the transport layer within a host
- Analogous to a door:
  - The sending process which is created in an application, shoves the messages out of the "door"


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

Socket Programming

Socket: conceptual view



Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

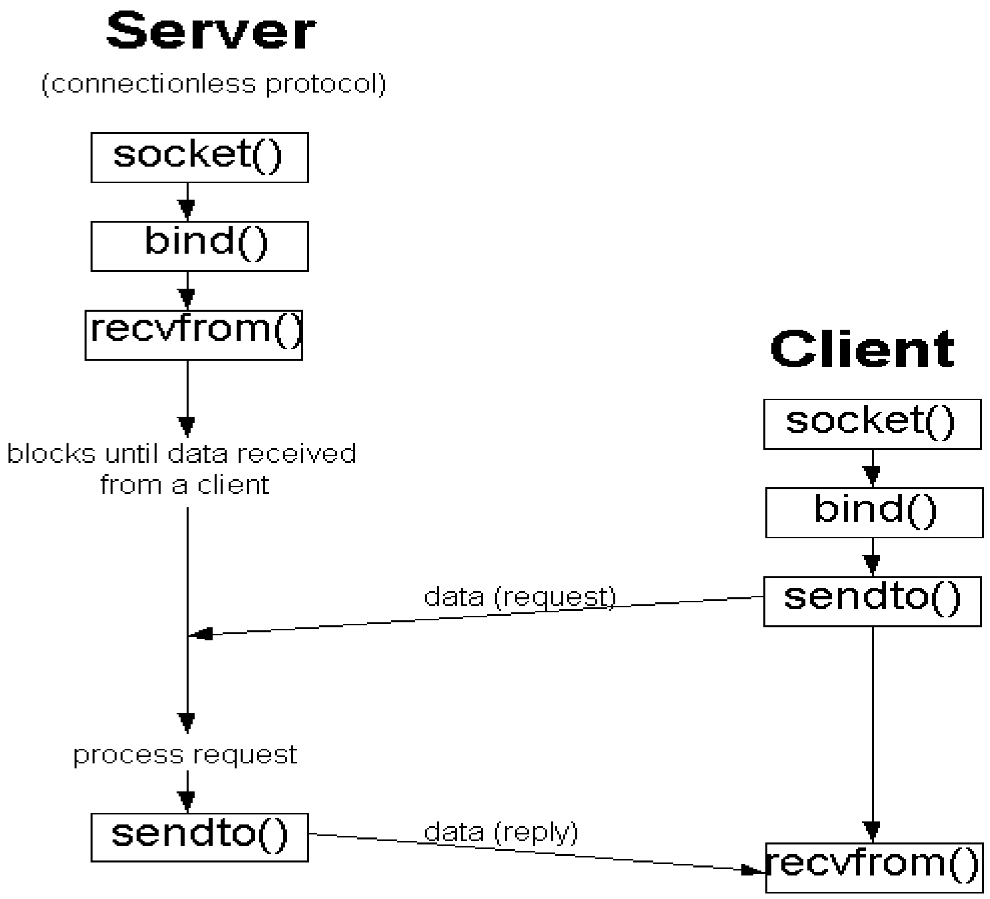
1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Socket Programming: Basics

- The server application must be running before the client can send anything.
- The server must have a socket through which it sends and receives messages. The client also need a socket.
- Locally, a socket is identified by a port number.
- In order to send messages to the server, the client needs to know the IP address and the port number of the server.

Port number is analogous to an apartment number. All doors (sockets) lead into the building, but the client only has access to one of them, located at the provided number.

BSD Socket Programming (connectionless)



Socket programming *with UDP*

UDP: no "connection" between client & server

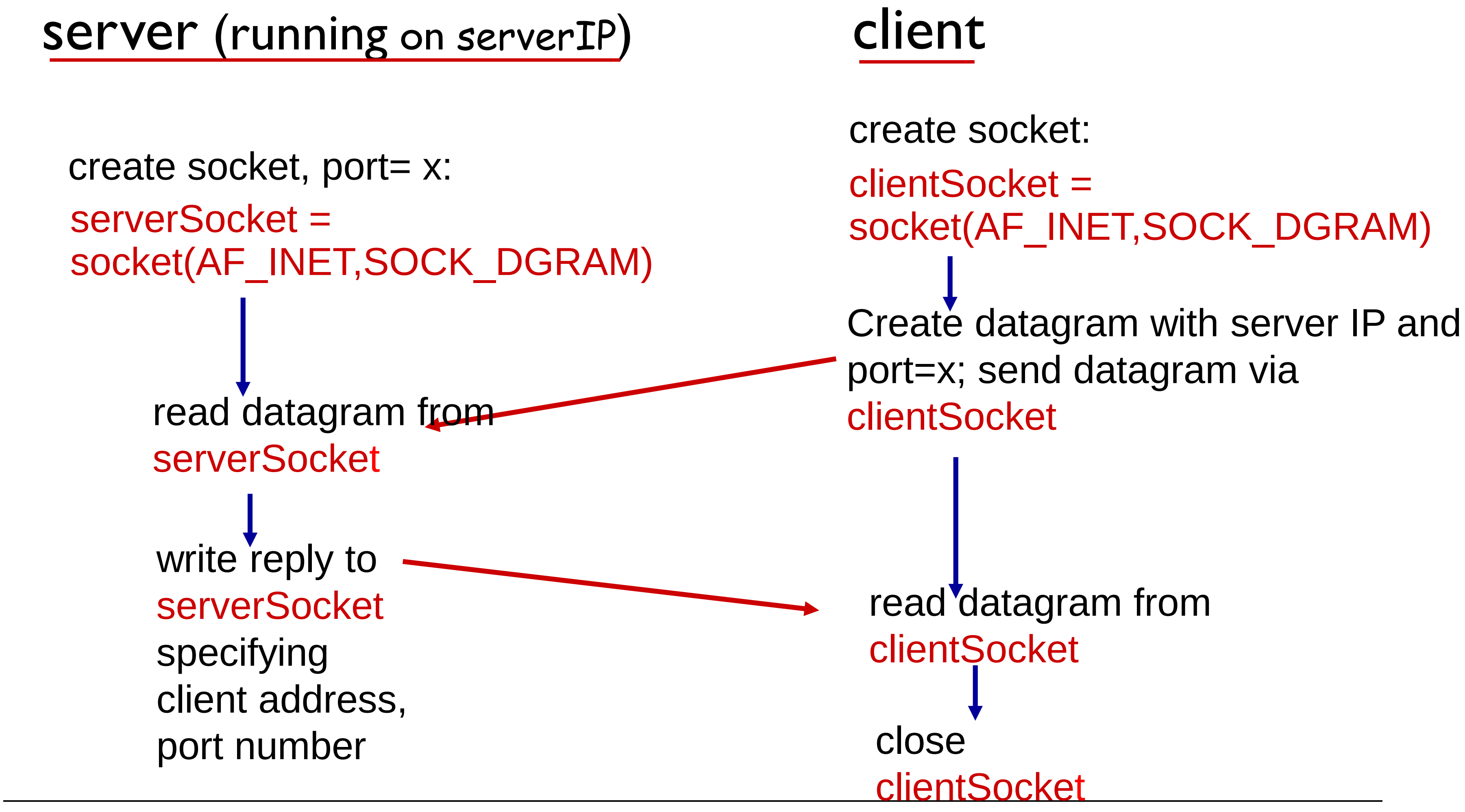
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

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

Client/server socket interaction: UDP



server (running on serverIP)

create socket, port= x:

serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:

clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind((", serverPort))
print (“The server is ready to receive”)
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                                   clientAddress)
```

include Python’s socket library

create UDP socket

bind socket to local port number 12000

Buffer size

Read from UDP socket into message, getting client’s address (client IP and port)

send upper case string back to this client

Example app: UDP client

Python UDPClient

include Python’s socket library

```
from socket import *
serverName = ‘servername’
serverPort = 12000
```

create UDP socket for server

```
clientSocket = socket(AF_INET,
                          SOCK_DGRAM)
```

get user keyboard input

```
message = raw_input(’Input lowercase sentence:’)
```

Attach server name, port to message; send into socket

```
clientSocket.sendto(message.encode(),
                         (serverName, serverPort))
```

Buffer size

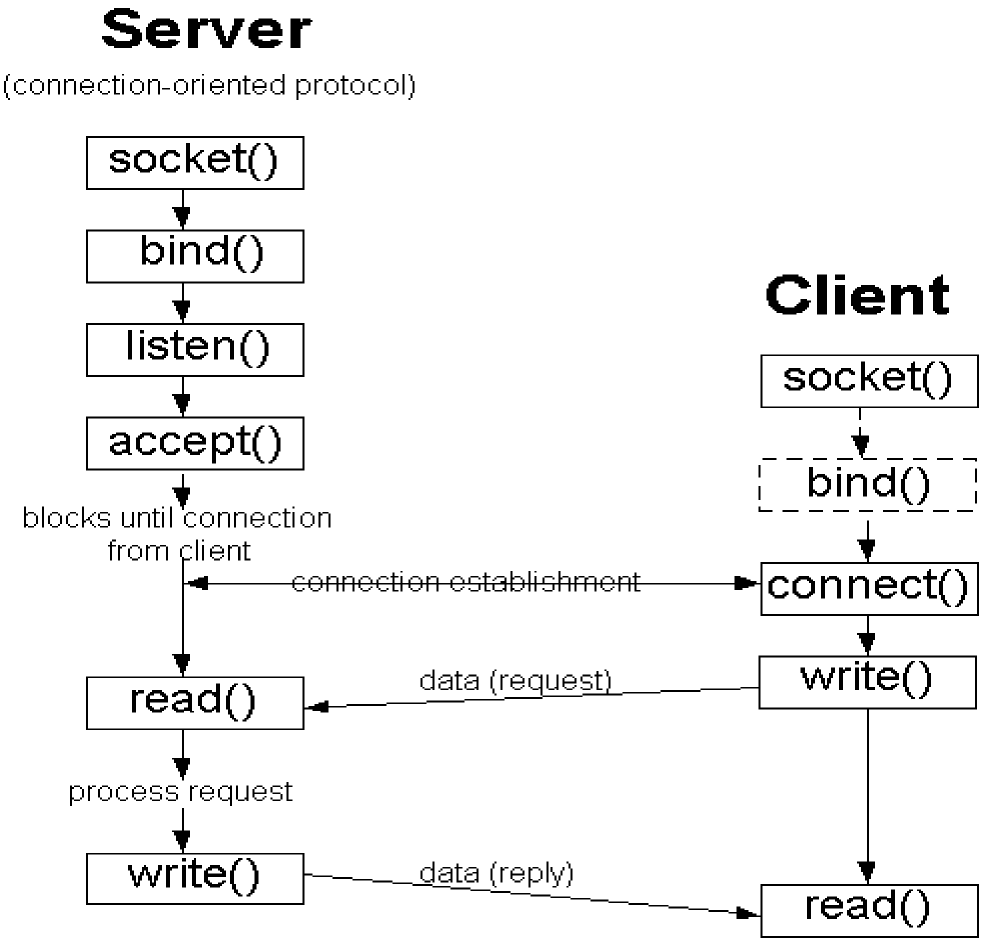
read reply characters from socket into string

```
modifiedMessage, serverAddress =
                          clientSocket.recvfrom(2048)
```

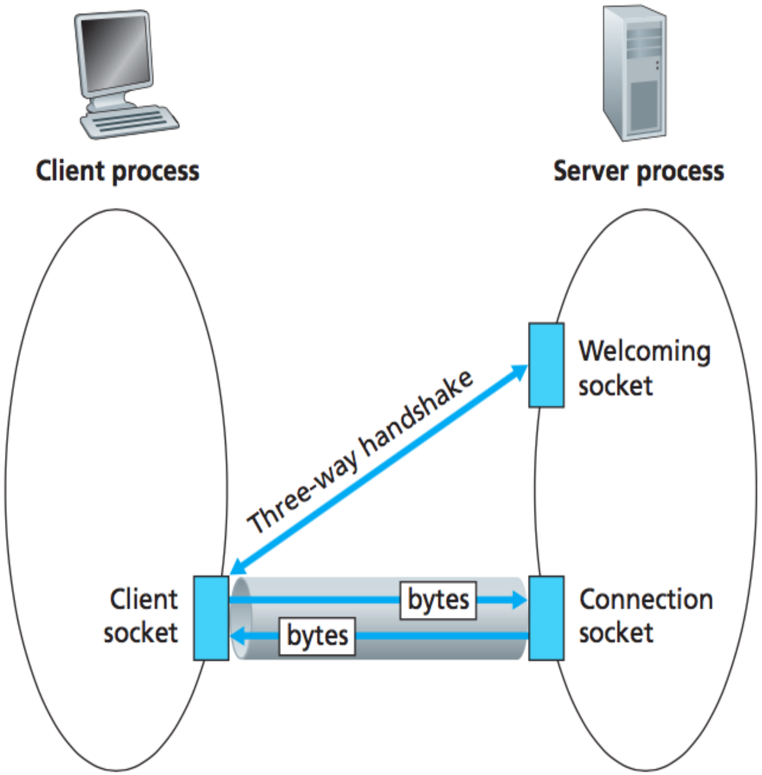
print out received string and close socket

```
print modifiedMessage.decode()
clientSocket.close()
```

BSD Socket Programming Flows (connection-oriented)



Socket programming *with TCP*



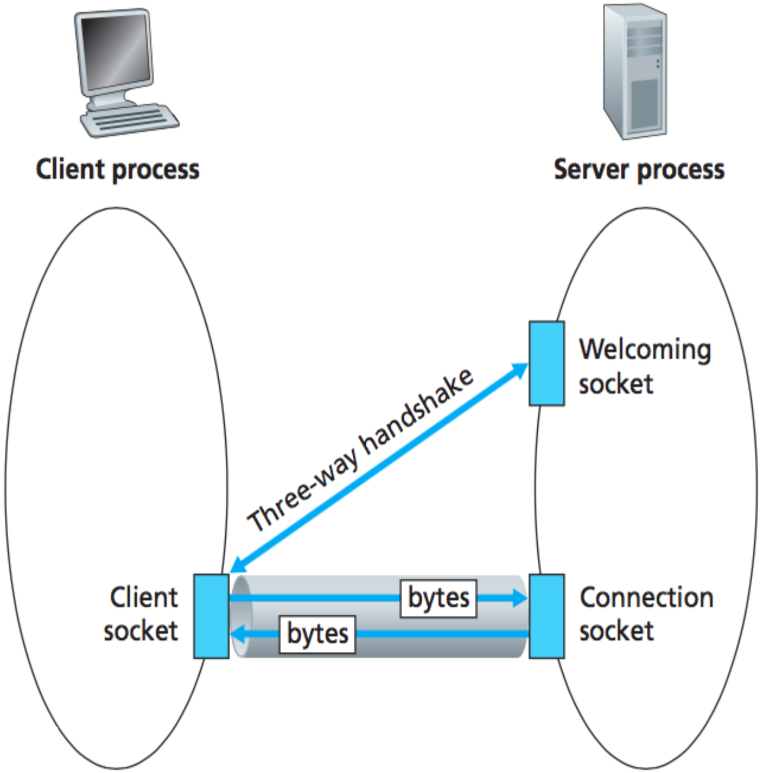
client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client’s contact

client contacts server by:

- Creating 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*

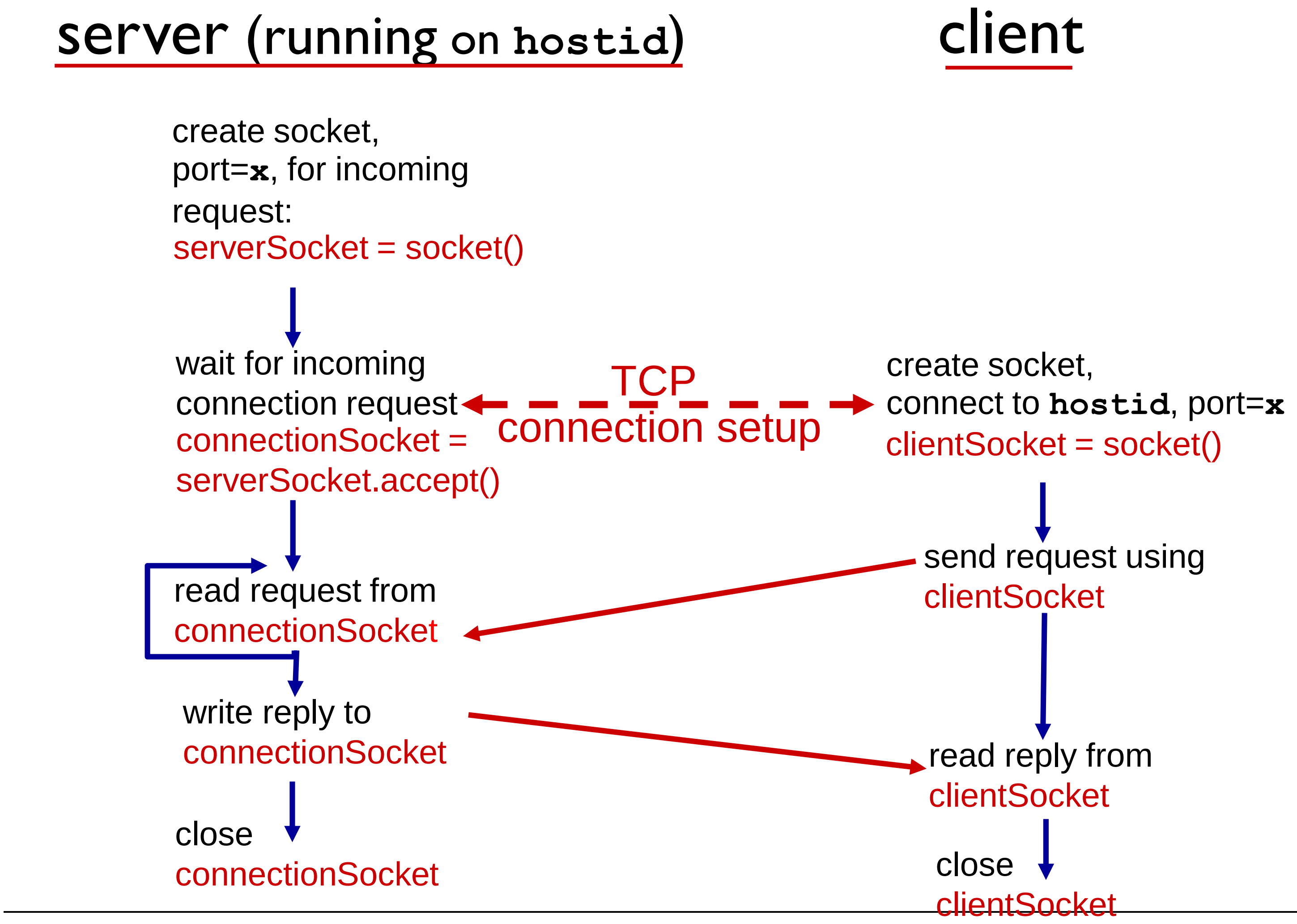


application viewpoint:

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

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

Client/server socket interaction: TCP



server (running on **hostid**)

create socket,
port=**x**, for incoming
request:
serverSocket = socket()

wait for incoming
connection request
connectionSocket =
serverSocket.accept()

read request from
connectionSocket

write reply to
connectionSocket

close
connectionSocket

TCP
connection setup

client

create socket,
connect to **hostid**, port=**x**
clientSocket = socket()

send request using
clientSocket

read reply from
clientSocket

close
clientSocket

Example app: TCP server

create TCP welcoming socket

server begins listening for incoming TCP requests

server waits on accept() for incoming requests, new socket created on return

read bytes from socket

close connection to this client (but *not* welcoming socket)

Python TCPServer

```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
print ‘The server is ready to receive’
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                        encode())
    connectionSocket.close()
```

Example app: TCP client

Python TCPClient

```
from socket import *
serverName = ’servername’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input(‘Input lowercase sentence:’)
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print (‘From Server:’, modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Java Socket Programming API

- Class ServerSocket
  - Connection-oriented server side socket
- Class Socket
  - Regular connection-oriented socket (client)
- Class DatagramSocket
  - Connectionless socket
- Class InetAddress
  - Encapsulates Internet IP address structure

Example: Java server (UDP)

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

class UDPServer {
  public static void main(String args[]) throws Exception
    {
```

Create datagram socket at port 9876

```
      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {
```

Create space for received datagram

```
          DatagramPacket receivePacket =
             new DatagramPacket(receiveData, receiveData.length);
```

Receive datagram

```
          serverSocket.receive(receivePacket);
```

Example: Java server (UDP), cont

```
String sentence = new String(receivePacket.getData());
```

Get IP addr port #, of sender

```
InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();
```

Create datagram to send to client

```
DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                     port);
```

Write out datagram to socket

```
serverSocket.send(sendPacket);
  }
 }
}
```

Example: Java client (UDP)

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

class UDPClient {
    public static void main(String args[]) throws Exception
    {
```

Create
input stream

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

Create
client socket

```
      DatagramSocket clientSocket = new DatagramSocket();
```

Translate
hostname to IP
address using DNS

```
      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

```
  DatagramPacket sendPacket =
    new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
```

Send datagram to server

```
  clientSocket.send(sendPacket);

  DatagramPacket receivePacket =
    new DatagramPacket(receiveData, receiveData.length);
```

Read datagram from server

```
  clientSocket.receive(receivePacket);

  String modifiedSentence =
     new String(receivePacket.getData());

  System.out.println("FROM SERVER:" + modifiedSentence);
  clientSocket.close();
  }
}
```

Example: Java Server (TCP)

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

```
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()));
```

Example: Java Server (TCP), cont

Create output stream, attached to socket

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

Read in line from socket

```
            clientSentence = inFromClient.readLine();

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

Write out line to socket

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

Example: Java Client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

  public static void main(String argv[]) throws Exception
  {
    String sentence;
    String modifiedSentence;
```

Create
input stream

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

Create
client socket,
connect to server

```
    Socket clientSocket = new Socket("hostname", 6789);
```

Create
output stream
attached to socket

```
    DataOutputStream outToServer =
      new DataOutputStream(clientSocket.getOutputStream());
```

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();

    }
}
```

Two Different Server Behaviors

- Iterative server
  - At any time, only handles one client request

```
for (;;) {
    accept a client request;
    handle it
}
```

- Concurrent server
  - Multiple client requests can be handled simultaneously
  - create a new process/thread to handle each request

```
for (;;) {
    accept a client request;
    create a new  process / thread  to
            handle request;
    parent process / thread continues
}
```

Example of Concurrent Server Python

```
import threading
def main():
        .......
        cx, addr = s.accept()
        server = threading.Thread(target=dns, args=[cx, addr] )
        server.start()
        .......

......

def dns(cx,addr):
        .......
```

Helpful Resources

- Python Socket Tutorial
  - https://docs.python.org/2/library/socket.html
  - https://docs.python.org/3.4/library/socket.html
- Java Socket Tutorial
  - http://download.oracle.com/javase/tutorial/networking/sockets/
- Computer Networking: A Top-Down Approach, 7th Edition. Section 2.7