BLG 433E – COMPUTER COMMUNICATIONS

Instructor: Prof. Sema F. OKTUĞ

Assistant: Mertkan AKKOÇ

SOCKET PROGRAMMING

OUTLINE

1) What is socket?

2) Socket Types

3) Programming

4) Show Time!

WHAT IS SOCKET?

MIND BLOWN

- How do two or more computer communicate with each other? What do they use?

- Where do all words go or where do they come from in a computer during communication?

- Don’t forget the role of Unix and C in communication !!!

WHAT IS SOCKET?

- ‘a way to **speak** to other **programs** using standard Unix **file descriptors.**’
- Unix **programs** does I/O by using **file descriptors**
- A **file descriptor** is simply an **integer** associated with an **open file**.
- If you want to communicate, you have to use a **file descriptor**.
- How to get **file descriptor?**
  - **socket()** system routine
  - Communicate with **send()** and **receive()**

SOCKET TYPES

- DARPA Internet addresses → **Internet Sockets**

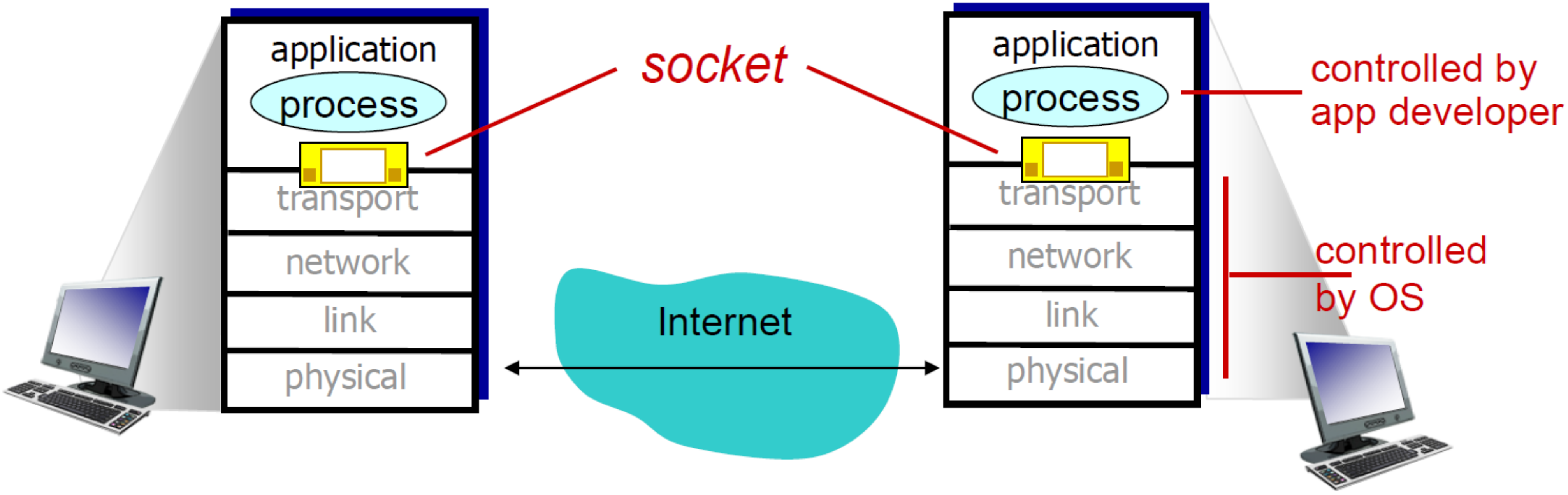
  1. Stream Sockets
  2. Datagram Sockets

- Path names on a local nodes → Unix Sockets

- CCITT X.25 addresses → X.25 Sockets

REMEMBER !!!

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



SOCKET TYPES

INTERNET SOCKETS

STREAM SOCKETS / SOCK_STREAM

- **Some applications:** HTTP, FTP

- reliable **two way connected**

- **reliable:** sequential data arrive and error-free. **HOW?**

TCP (Transmission Control Protocol)

✓ responsible data integrity

✓ just a bunch of rule

SOCKET TYPES

INTERNET SOCKETS

DATAGRAM SOCKETS / SOCK_DGRAM

- **Some applications:** DNS, VoIP, games, audio, video
- **connectionless:** No connection needed, build packet and send, don’t maintain an open connection
- **unreliable:** don’t care about there is a connection and dropped packets

UDP (User Datagram Protocol)

✓ just responsible to send data. Receiving control done by application

✓ just a bunch of rule

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 *with UDP*

UDP: no "connection" between client & server

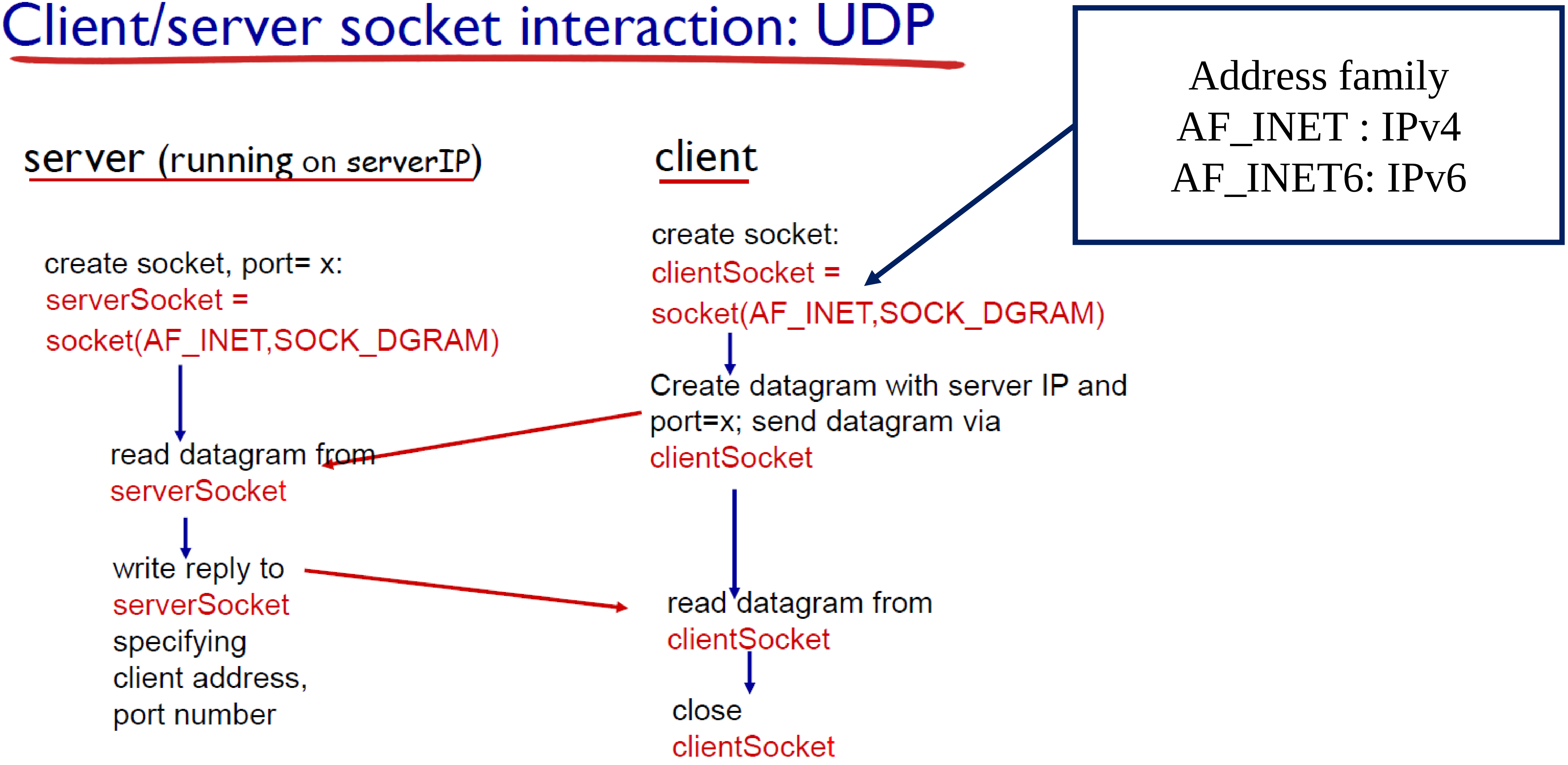
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

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

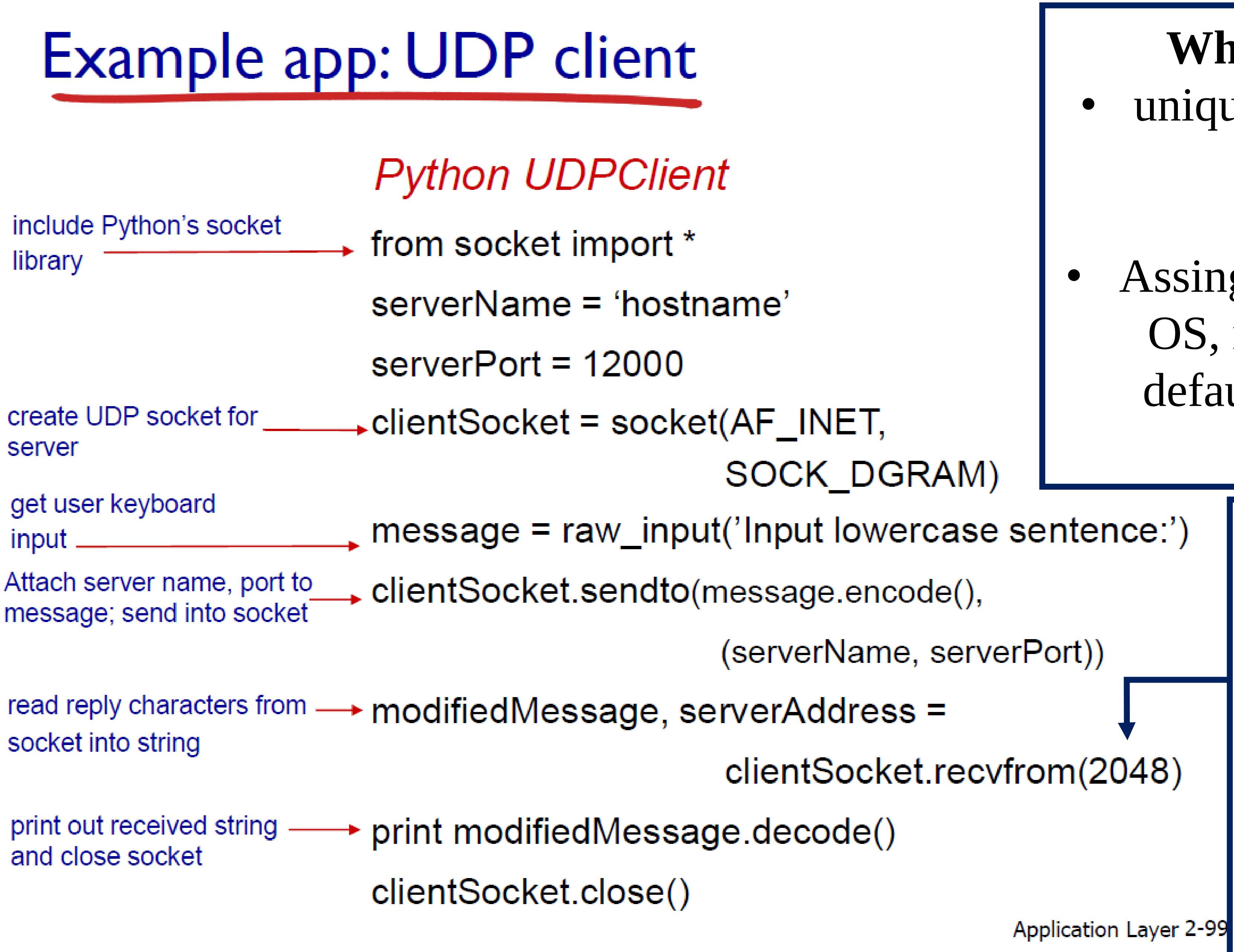
create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

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

read datagram from
clientSocket

close
clientSocket

Address family
AF_INET : IPv4
AF_INET6: IPv6



Example app: UDP client

Python UDPClient

include Python's socket library

```
from socket import *
serverName = ‘hostname’
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))
```

read reply characters from socket into string

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

print out received string and close socket

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

Application Layer 2-99

What is port number?

- uniquely identifies a network-based application
- 16-bit integer
- Assingned automatically by the OS, manually by the user, or default for some applications

- the number of bytes you want to accept
- 2048 is the maximum for UDP
- if coming message greater than you specify you get an error or message will be discarded

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)
```

create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)

bind socket to local port number 12000 → serverSocket.bind(('', serverPort))

loop forever → while True:

Read from UDP socket into message, getting client’s address (client IP and port) → message, clientAddress = serverSocket.recvfrom(2048)

send upper case string back to this client → serverSocket.sendto(modifiedMessage.encode(), clientAddress)

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:

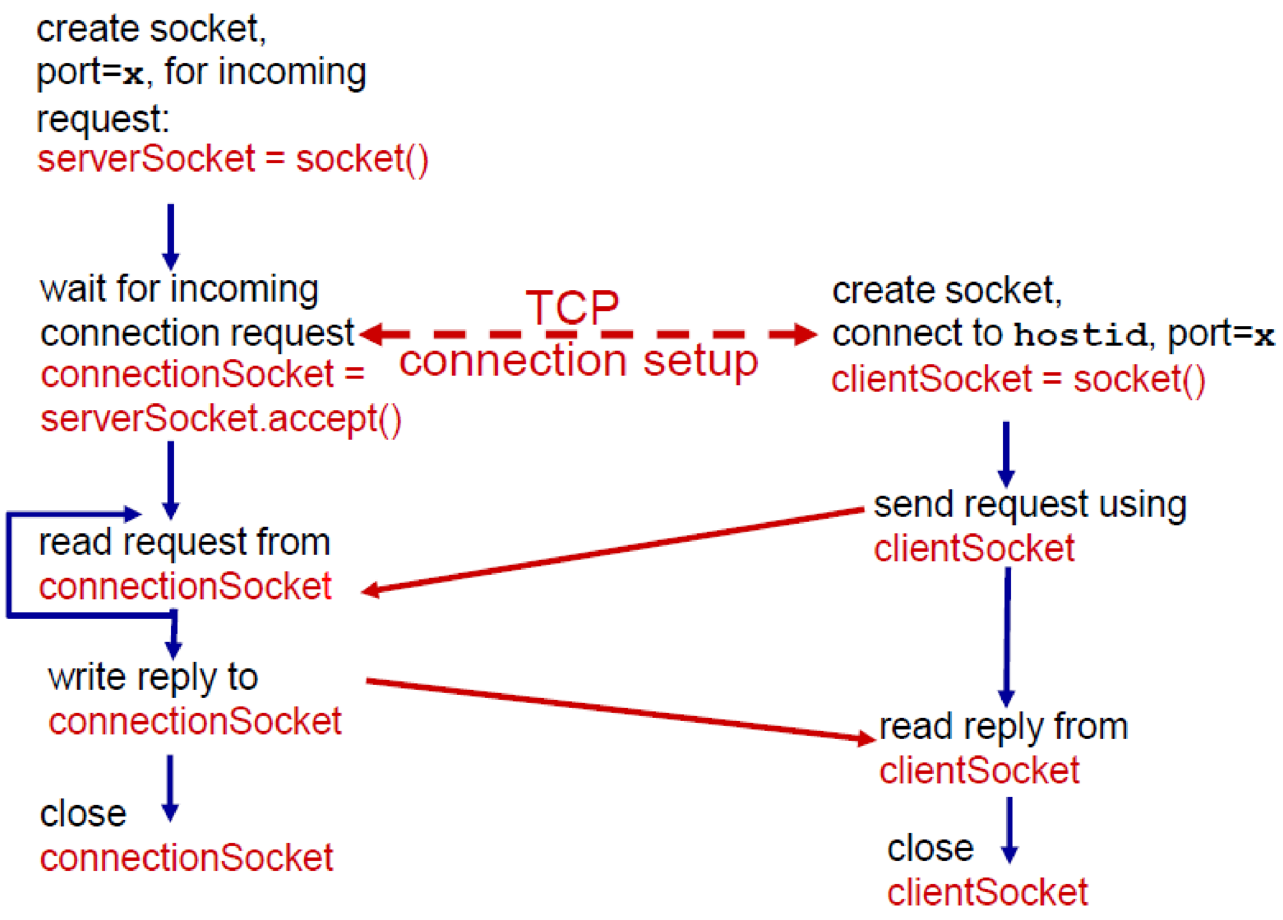
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- 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)

application viewpoint:

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

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

client

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

send request using
clientSocket

read reply from
clientSocket

close
clientSocket

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

- There is no strict rule like UDP

Example app: TCP server

Python TCPServer

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

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

read bytes from socket (but not address as in UDP)

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

```
 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()
```

REFERENCES

- Book Slides / Chapter 2

  (JAMES F. KUROSE, KEITH W. ROSS, COMPUTER NETWORKING: A TOP-DOWN APPROACH, 7TH EDITION, PEARSON, 2017)

- http://beej.us/guide/bgnet/html/multi/theory.html

  (Türkçe: http://www.belgeler.org/bgnet/bgnet.html)

SHOW TIME