HY335b
Computer Networks

Introduction to Socket Programming

Spring 2017

Author: Gigis Petros

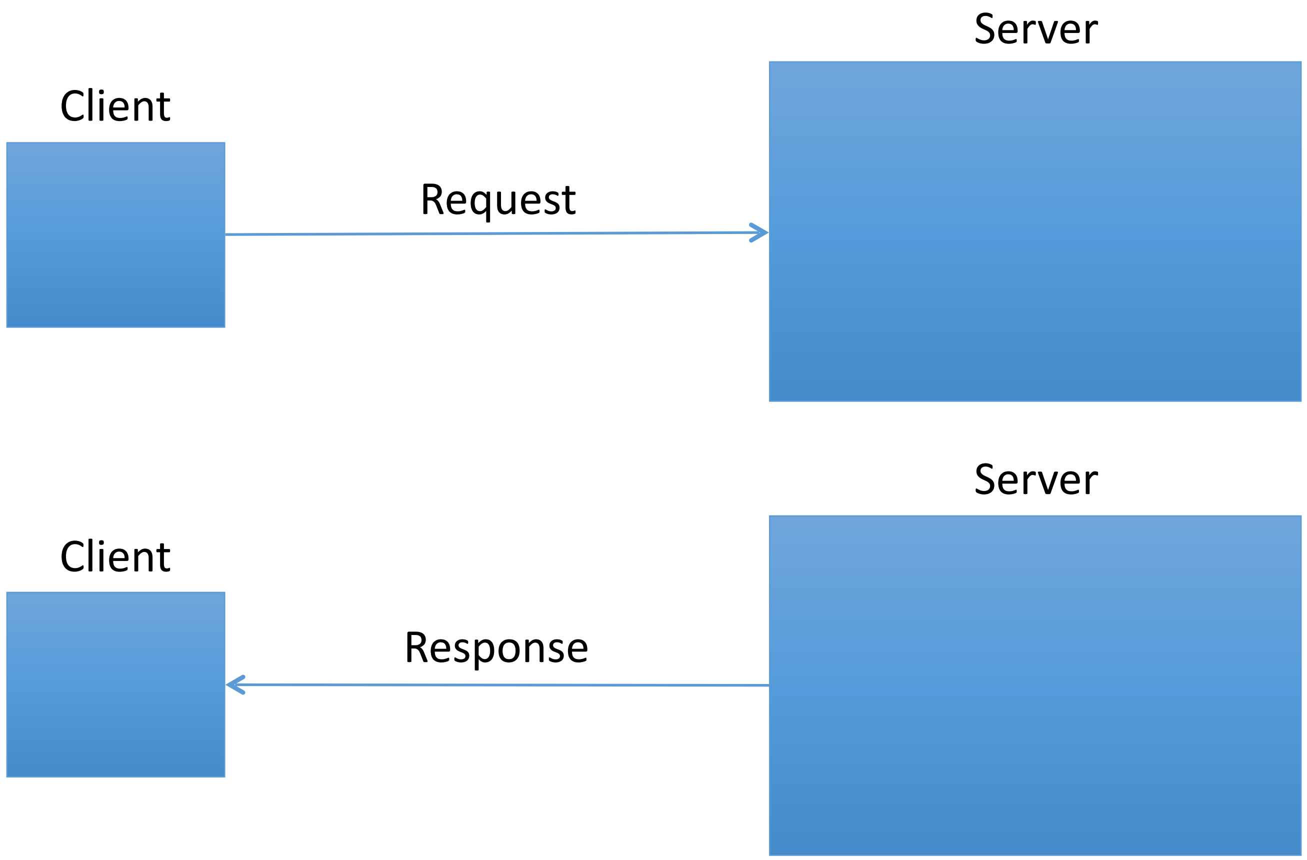
Presenter: Milolidakis Alex

08/03/2017

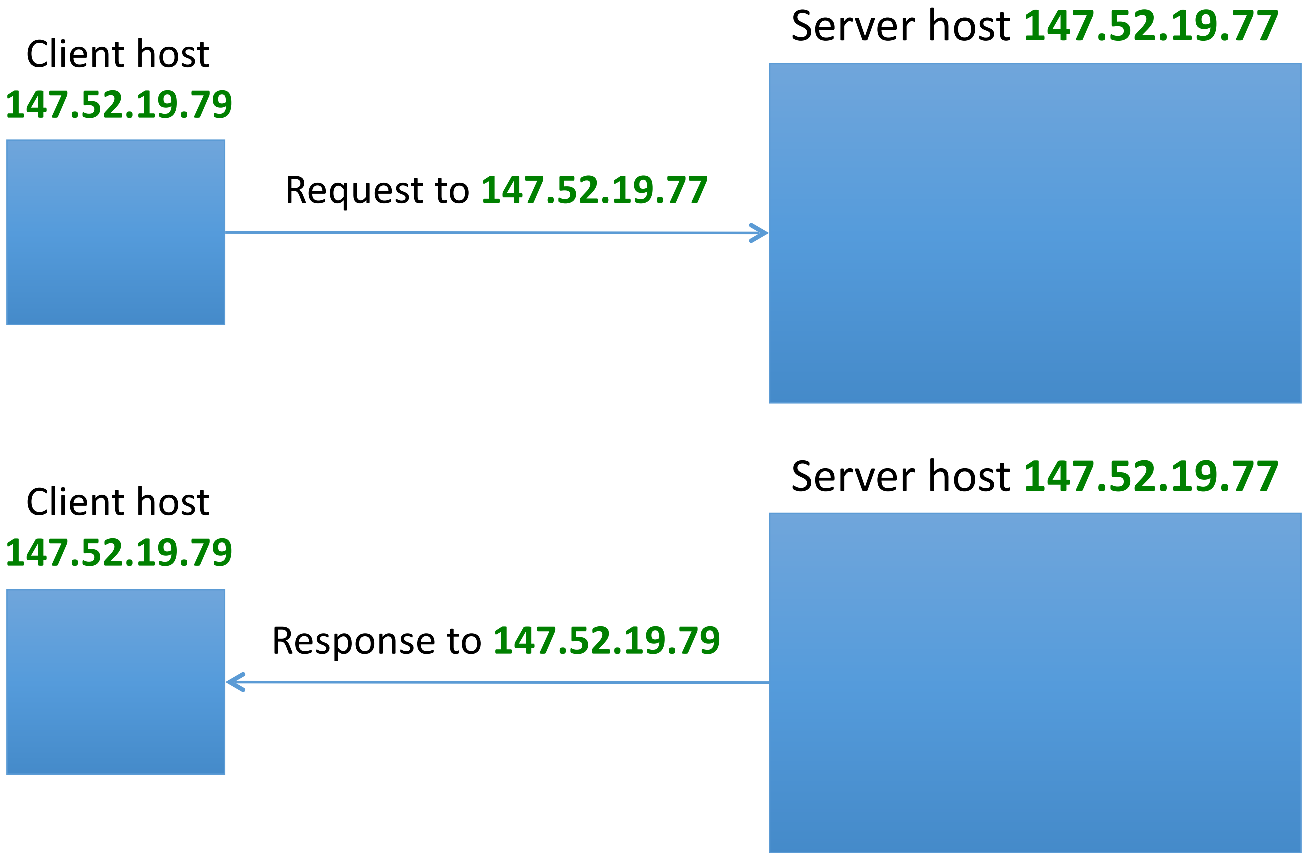
Outline

- Basics about sockets
- Flow diagram in socket communication
- TCP vs UDP case
- Examples in C
- Examples in Python
- Live demo

Basic Schema



Using addresses to identify Hosts



Client host **147.52.19.79**

Request to **147.52.19.77**

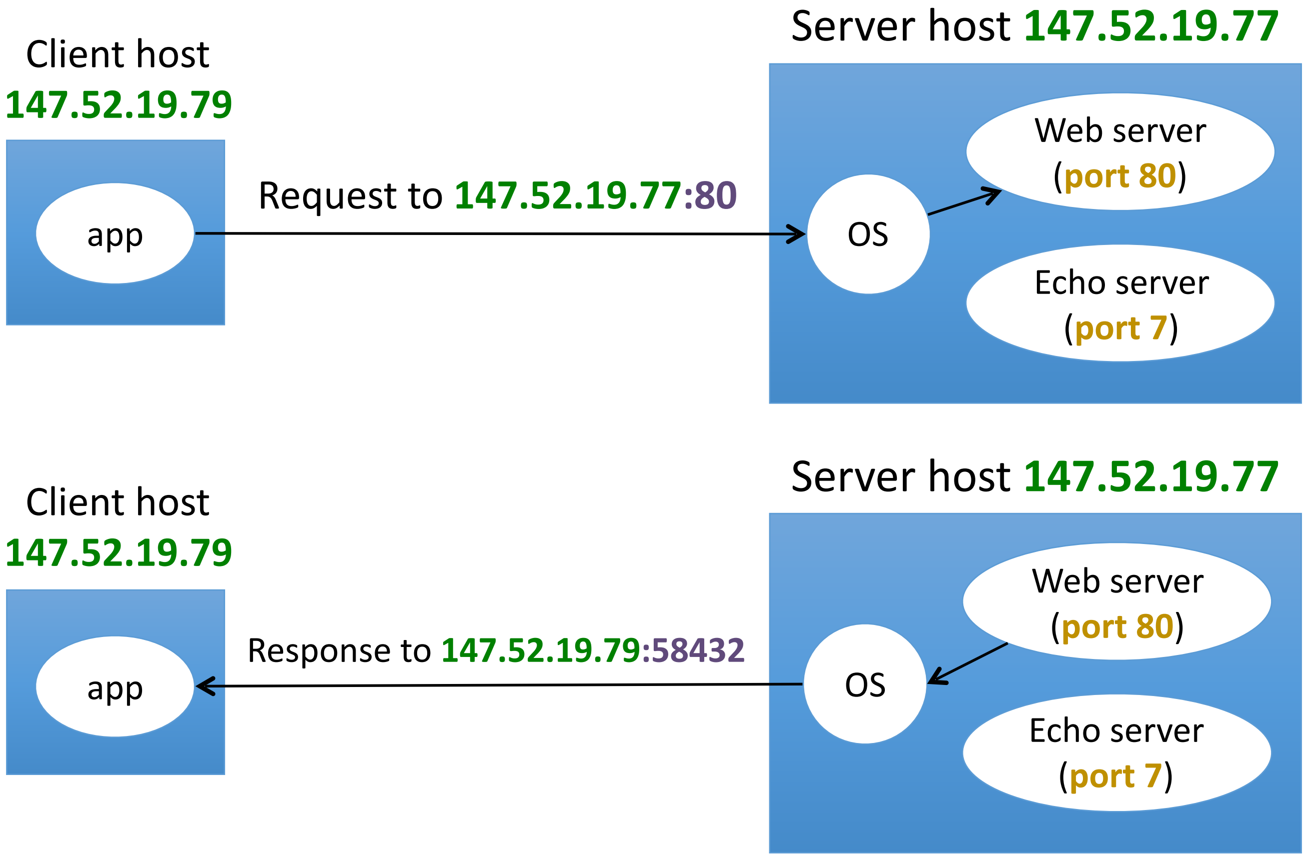
Server host **147.52.19.77**

Client host **147.52.19.79**

Response to **147.52.19.79**

Server host **147.52.19.77**

Using ports to identify Services



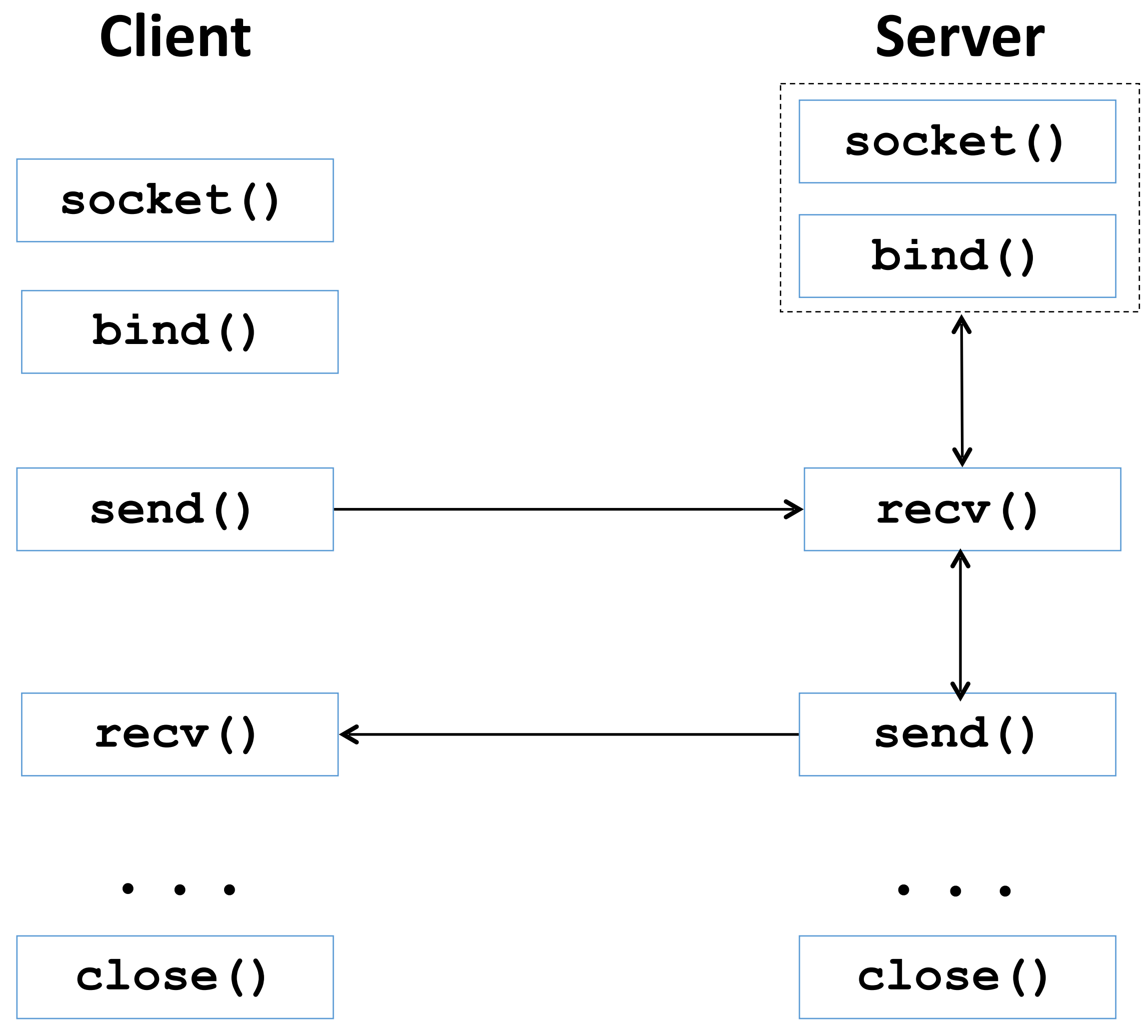
Choosing appropriate port

- There are 1024 well-known ports **reserved.** Should not be used.

- You can use any port number between range (1024, 65535]. If a port is used from another service, you should use another port number **randomly**.

- Some well known port examples are:
  - ➢HTTP: 80
  - ➢SSH: 22
  - ➢SMTP: 25
  - ➢DNS: 53

UDP Example


Client
Server
socket()
bind()
socket()
bind()
send()
recv()
recv()
send()
. . .
. . .
close()
close()

Socket programming with UDP

UDP: no "connection" between client & server

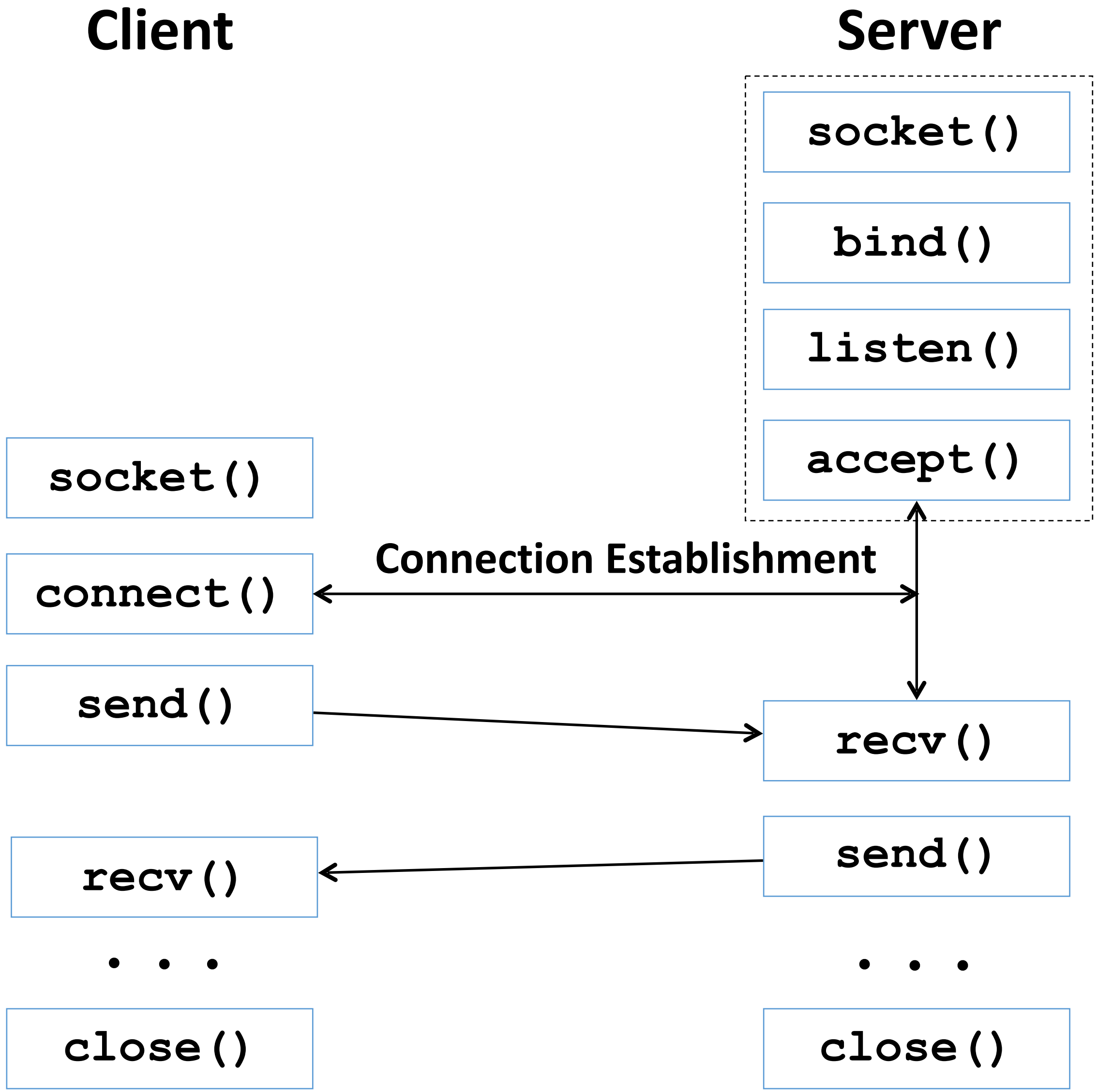
- *no handshaking before sending data*
- *sender explicitly attaches IP destination address and port # to each packet*
- *rcvr extracts sender IP address and port# from received packet*

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*

TCP Example


Client
Server
socket()
bind()
listen()
accept()
socket()
connect()
Connection Establishment
send()
recv()
send()
recv()
. . .
. . .
close()
close()

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

Application viewpoint:

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

Creating a Socket (C Programming)

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

Sockets allow communication between two different processes on the same or different machines

***socket()** creates a socket of a certain domain, type and protocol specified by the parameters*

- *Possible domains:*
  - ***AF_INET** for IPv4 internet protocols*
  - ***AF_INET6** for IPv6 internet protocols*

Creating a Socket (C Programming)

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- *Possible types:*
    - ***SOCK_STREAM*** *provides reliable two way connection-oriented byte streams (TCP)*
    - ***SOCK_DGRAM*** *provides connection-less, unreliable messages of fixed size (UDP)*

- *protocol depends on the domain and type parameters. In most cases 0 can be passed*

On success returns new socket descriptor, else –1

Bind a Socket (C Programming)

```
#include <sys/socket.h>

int bind(int socket, const struct sockaddr *address,
            int addr_len);
```

- ***bind()*** *assigns an open socket to a specific network interface and port*
- **socket** is the socket descriptor
- **address** is the local host address
- **addr_len** is the size of the address structure

On success returns 0, else –1

Listening for incoming connections (C Programming)

```
int listen(int socket, int backlog);
```

- *After binding to a specific port a TCP server can listen at this port for incoming connections*

- ***backlog** parameter specifies the maximum possible outstanding connections*

  Default value in most OS is 5

- *Clients can connect using the **connect()** call*

Accept() incoming connections (C Programming)

```
int accept(int socket, struct sockaddr * far_addr,
                   int far_addr_length);
```

- ***socket***: *The socket after listen function*
- ****** far_addr***: *Optional pointer to a buffer that fills with the client's address*
- ***far_addr_length***: *Size of far_addr*

On success returns new socket descriptor, else –1

connect() function (C Programming)

```
int connect(int socket, struct sockaddr * far_addr,
                int far_addr_length);
```

- ***socket***: *the unconnected socket*
- ***far_addr***: *the data structure that describes the server address*
- ***far_addr_length***: *size of far_addr*

On success returns new socket descriptor, else –1

send() & recv() functions (C Programming)

```
int send(int socket, char * message, int length, int flags);
```

```
int recv(int socket, char * message, int length, int flags);
```

- ***socket***: *specified socket*
- ***message***: *buffer to send/receive*
- ***length***: *buffer length*
- ***flags***: *use none => set to 0*

On success send() returns the total number of bytes sent, recv() returns total number of bytes received, else -1

closesocket() function (C Programming)

```
int closesocket (int socket);
```

socket: the socket to be closed

On success returns 0, else -1

Python examples (UDP & TCP)

Application Example:

- *Client reads a line of characters (data) from its keyboard and sends the data to the server*
- *The server receives the data and converts characters to uppercase*
- *The server sends the modified data to the client*
- *The client receives the modified data and displays the line on its screen*

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(socket.AF_INET,
                                   socket.SOCK_DGRAM)
```

get user keyboard input

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

Attach server name, port to message; send into socket

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

read reply characters from socket into string

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

print out received string and close socket

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

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 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

create UDP socket

bind socket to local port number 12000

loop forever

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

send upper case string back to this client

Example app: TCP client

Python TCPClient

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

create TCP socket for server, remote port 12000

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input(‘Input lowercase sentence:’)
```

No need to attach server name, port

```
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print ‘From Server:’, modifiedSentence
clientSocket.close()
```

Example app:TCP server

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 1:
     connectionSocket, addr = serverSocket.accept()

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

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)

Questions?

Google is your friend.
Before ask google it first.

Use Moodle forum for questions.