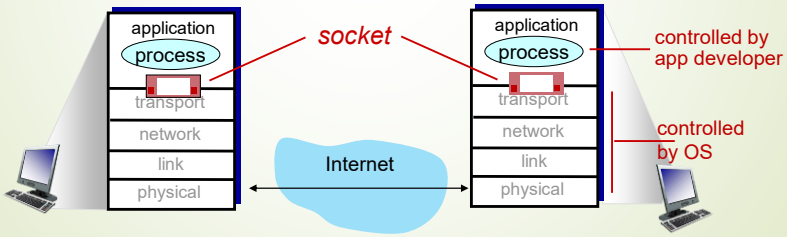
Socket Programming Python

Fall 2021

Sockets

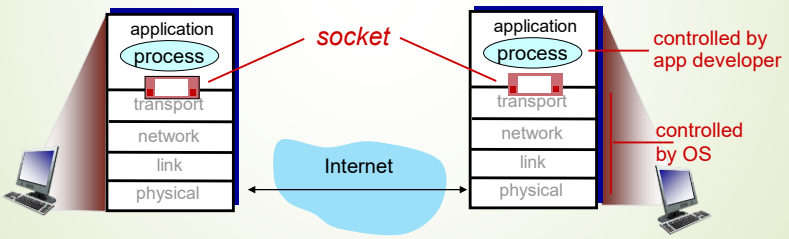
- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
  - two sockets involved: one on each side



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

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

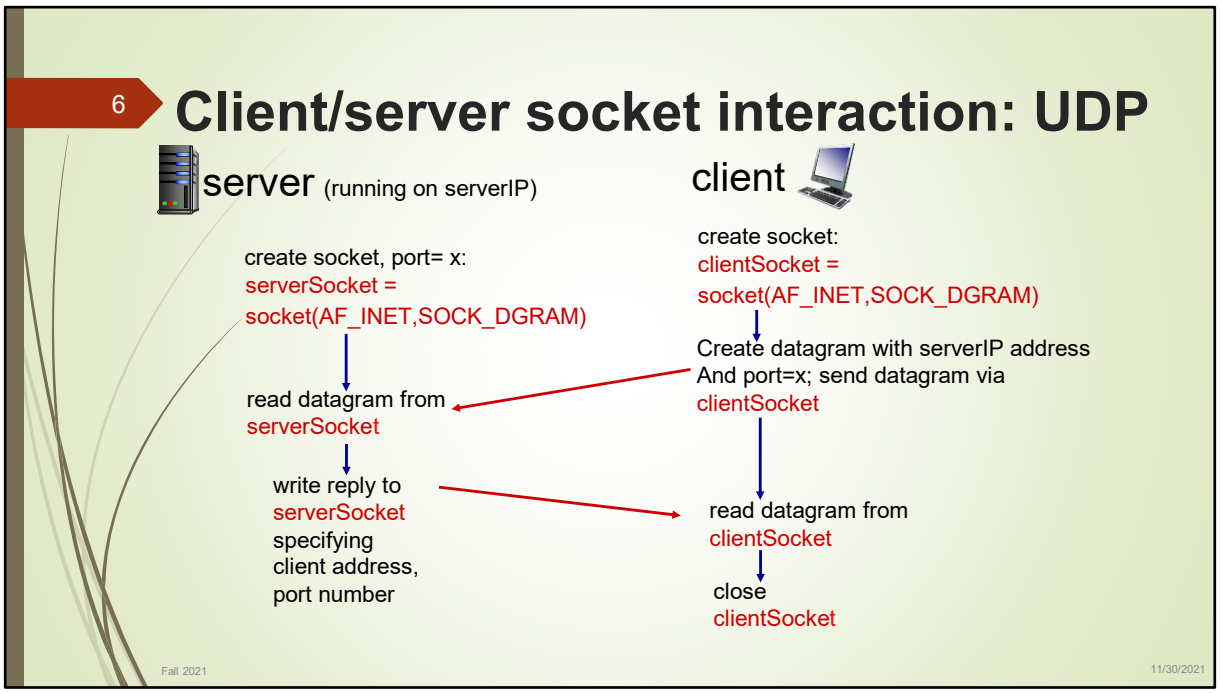
UDP: no “connection” between client and server:

- 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 processes



socket(AF_INET, SOCK_DGRAM)

1) **AF_INET:**

- The primary purpose of AF_INET was to allow for other possible network protocols or address families (AF is for address family; AF_INET is for the (IPv4) internet protocol family).
- Socket are characterized by their domain, type and transport protocol.
- Common domains are:
  - AF_UNIX: address format is UNIX pathname
  - AF_INET: address format is host and port number
  - (there are actually many other options which can be used here for specialized purposes).
  - Usually we use AF_INET for socket programming

2) **SOCK_DGRAM:** arguemt indicates socket type is udp

Socket.argument -> this argument indicates socket type either tcp or udp

- SOCK_STREAM = TCP
- SOCK_DGRAM = UDP

Example app: UDP client

Python UDPClient

include Python's socket library →
```
from socket import *
serverIP = '127.0.0.1'
serverPort = 12000
```
create UDP socket for server →
```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```
OS assign the port automatically →
```
clientSocket.bind( (' ', 0) )
```
get user keyboard input →
```
message = input('Input lowercase sentence: ')
```
attach server name, port to message; send into socket →
```
clientSocket.sendto(message.encode(),
                    (serverIP, 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()
```


1) socket.recvfrom – This method receives UDP message
2) socket.sendto – This method transmits UDP message
3) socket.close – This method closes socket
4) socket.gethostname – Returns a hostname
5) socket.bind(' ', port) – This method binds address hostname(' ‘ is translated to local host), port number to socket.

If port = 0 then the OS assigns a dynamic port number

Note: Line serverIP = '127.0.0.1' can be replaced by serverName = gethostname()

Example app: UDP server

Python UDPServer

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 →

```
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)
```

UDP Client and Server running on Local Host

- Open two distinct Command Prompts

Command Prompt - python UDPServer.py

```
Microsoft Windows [Version 10.0.22000.318]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd c:\Python Workspace

c:\Python Workspace>python UDPServer.py   1
The server is ready to receive
```

Command Prompt

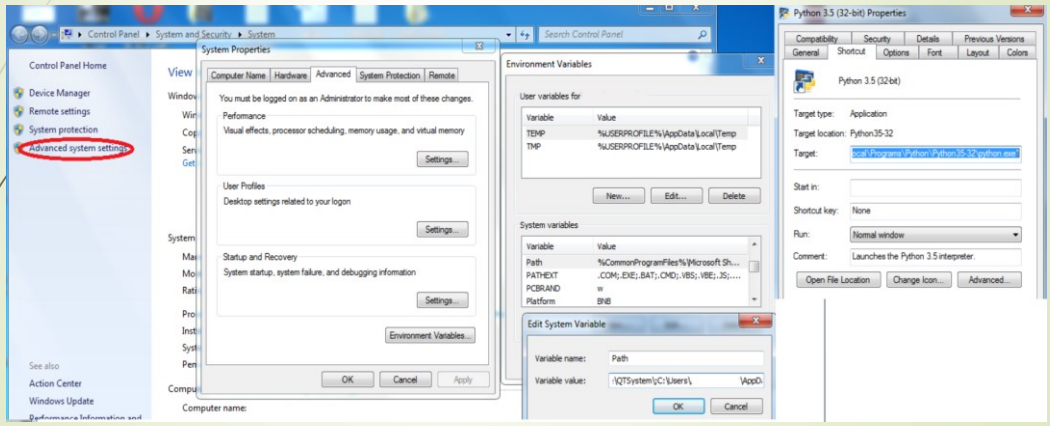
```
Microsoft Windows [Version 10.0.22000.318]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd c:\Python Workspace

c:\Python Workspace>Python UDPClient.py   2
Input lowercase sentence:"Hello from client"   3
"HELLO FROM CLIENT"

c:\Python Workspace>
```

UDP Client and Server running on Local Host *(Cont.)*

- Download Python from: Download Python | Python.org
- Make sure to select **Add Python to PATH** during the installation, to run the program from command window.

UDP Client and Server running on Local Host *(Cont.)*

- If not added during the installation, follow the following steps to add python to Windows environment variables.



12

UDP Using Arguments Task

- In the previous program, in the client side
  - The OS assigns the port number automatically.
  - The client sends a line of characters (data) from its keyboard to server.
- Update the client side of program so that it takes 2 arguments
  - The port it will bind to.
  - The message that will be sent to the server.
- Keep the server side of program as it is.

A simple code showing how to use arguments is sent with the materials named testArg, with example screenshot.

UDP Using Arguments Task Output

Command Prompt - python UDPServer-usingarg.py

```
Microsoft Windows [Version 10.0.22000.318]
(c) Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>cd C:\Python Workspace

C:\Python Workspace>python UDPServer-usingarg.py
The server is ready to receive
This message:  hello from client1
 is received from:  ('127.0.0.1', 1600)
This message:  hello from client 2
 is received from:  ('127.0.0.1', 1700)
```

1

Command Prompt

```
C:\Python Workspace>python UDPClient-usingarg.py 1600 "hello from client1"
HELLO FROM CLIENT1

C:\Python Workspace>
```

2

Select Command Prompt

```
C:\Python Workspace>python UDPClient-usingarg.py 1700 "hello from client 2"
HELLO FROM CLIENT 2

C:\Python Workspace>
```

3

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:

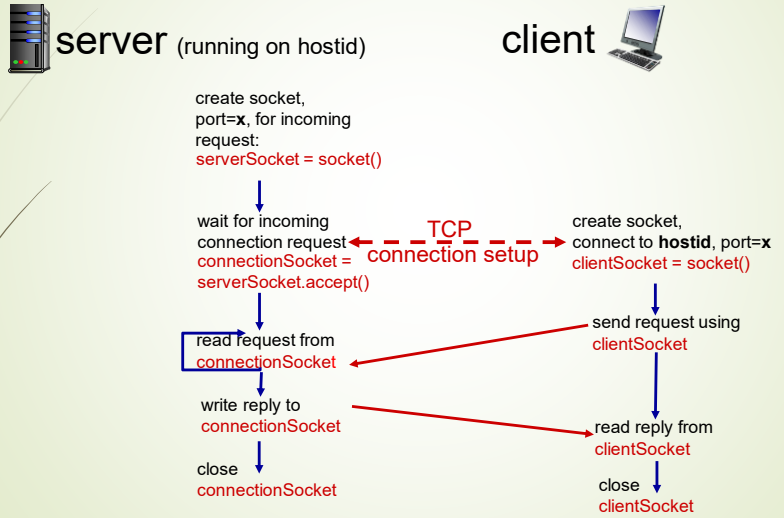
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- 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

Application viewpoint

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

Client/server socket interaction: TCP



Example app: TCP client

Python TCPClient

create TCP socket for server, remote port 12000

No need to attach server name, port

```
from socket import *
serverIP = '127.0.0.1'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverIP,serverPort))
sentence = input('Input lowercase sentence: ')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print (' From Server: ', modifiedSentence.decode())
clientSocket.close()
```


1) connect – This method actively initiates TCP server connection
2) socket.recv – This method receives TCP message
3) socket.send – This method transmits TCP message

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()
```


1) socket. bind(hostname_address, port) – This method binds address hostname, port number to socket.
2) socket.listen – This method setups and start TCP listener "keeps waiting for any incoming connection from clients".
3) socket.accept – This passively accepts client connection, waiting until connection arrives blocking.

TCP Client and Server running on Local Host

- Open two distinct Command Prompts

Command Prompt - python TCPServer.py

```
C:\Python Workspace>python TCPServer.py
The server is ready to receive
```

Command Prompt

```
C:\Python Workspace>python TCPClient.py
Input lowercase sentence:hello from tcp client
From Server: HELLO FROM TCP CLIENT

C:\Python Workspace>
```

TCP Task

- In the previous program, client sends only 1 message to server 'hello from tcp client' and server responds with the uppercase message.
- Update the program / make a simple chat program so that
  - Client can send/receive multiple messages to server.
  - A special exit message is used to disconnect 'Exit'.
- Each time client sends a message, server responds with a confirmation of receiving it and its length.

socket.getsockname()
Returns a pair representing the IP and Port assigned to the socket

TCP Task Output

Command Prompt - Python TCPServerTask.py

```
c:\Python Workspace>Python TCPServerTask.py
Listening at ('0.0.0.0', 1060)
The server now is connected to:  ('127.0.0.1', 49694)
Socket connects between  ('127.0.0.1', 1060)  and  ('127.0.0.1', 49694)
Received Message from Client:  Hi
Received Message from Client:  Testing Connection
Received Message from Client:  Message1
Received Message from Client:  Message2
Received Message from Client:  Exit
Reply sent, Server socket closed
Listening at ('0.0.0.0', 1060)
```

Command Prompt

```
C:\Python Workspace>Python TCPClientTask.py
Enter message to send or type Exit to disconnect: Hi
Received Message from Server:  Your data was 2 bytes
Enter message to send or type Exit to disconnect: Testing Connection
Received Message from Server:  Your data was 18 bytes
Enter message to send or type Exit to disconnect: Message1
Received Message from Server:  Your data was 8 bytes
Enter message to send or type Exit to disconnect: Message2
Received Message from Server:  Your data was 8 bytes
Enter message to send or type Exit to disconnect: Exit
Received Message from Server:  Disconnect
Now you are disconnected from the server

C:\Python Workspace>
```

Thanks