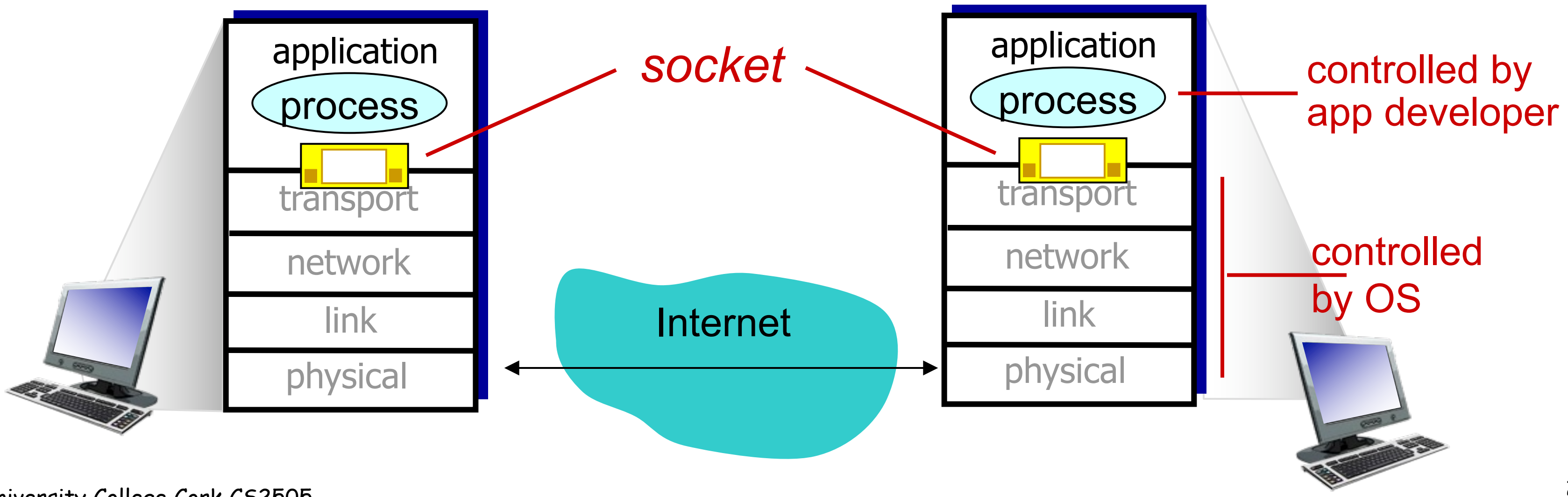
CS2505

Network Programming using the Socket Application Programming Interface (API)

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 API

- introduced in BSD4.1 UNIX, 1981
- client/server paradigm
- two types of transport service accessed via socket API:
  - UDP (unreliable datagram)
  - TCP (reliable byte-stream)
- Available on all operating systems and with libraries for all programming languages

socket

An *application-created, Operating System-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
- Similarly, each client needs a socket
- Socket is locally identified with a port number
  - *Analogous to the apt # in a building*
- Client needs to know Internet address (IP address) of server's computer and socket port number.

Example Application

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

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
- Operating System (OS) attaches IP address and port of sending socket to each segment
- receiver 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 course, we sometimes use "UDP segment" for consistency with TCP terminology

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 Application: UDP Client

Python UDPClient

include Python's socket library

create UDP socket for use by client

get user keyboard input

Attach server IP & port to message; send into socket

read reply characters from socket into string (buffer size 2048 bytes)

print out received string and close socket

```
from socket import *
serverName = ‘hostname’
serverPort = 12000
clientSocket = socket(socket.AF_INET,
                              socket.SOCK_DGRAM)
message = raw_input(’Input lowercase sentence:’)
clientSocket.sendto(message,(serverIP, serverPort))
modifiedMessage, serverAddress =
                              clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

Example App: UDP Server

Python UDPServer

	`from socket import *`
	`serverPort = 12000`
create UDP socket	`serverSocket = socket(AF_INET, SOCK_DGRAM)`
bind socket to local port number 12000	`serverSocket.bind(('', serverPort))`
	`print "The server is ready to receive"`
loop forever	`while 1:`
Read from UDP socket into message, getting client's address (client IP and port)	`    message, clientAddress = serverSocket.recvfrom(2048)`
	`    modifiedMessage = message.upper()`
send upper case string back to this client	`    serverSocket.sendto(modifiedMessage, clientAddress)`

```
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)
```

UDP observations & questions

- Both client & server use DatagramSocket (SOCK_DGRAM)
- Dest IP and port are explicitly attached to segment.
- What would happen if change both clientSocket and serverSocket to use name "mySocket"?
- 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?

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:

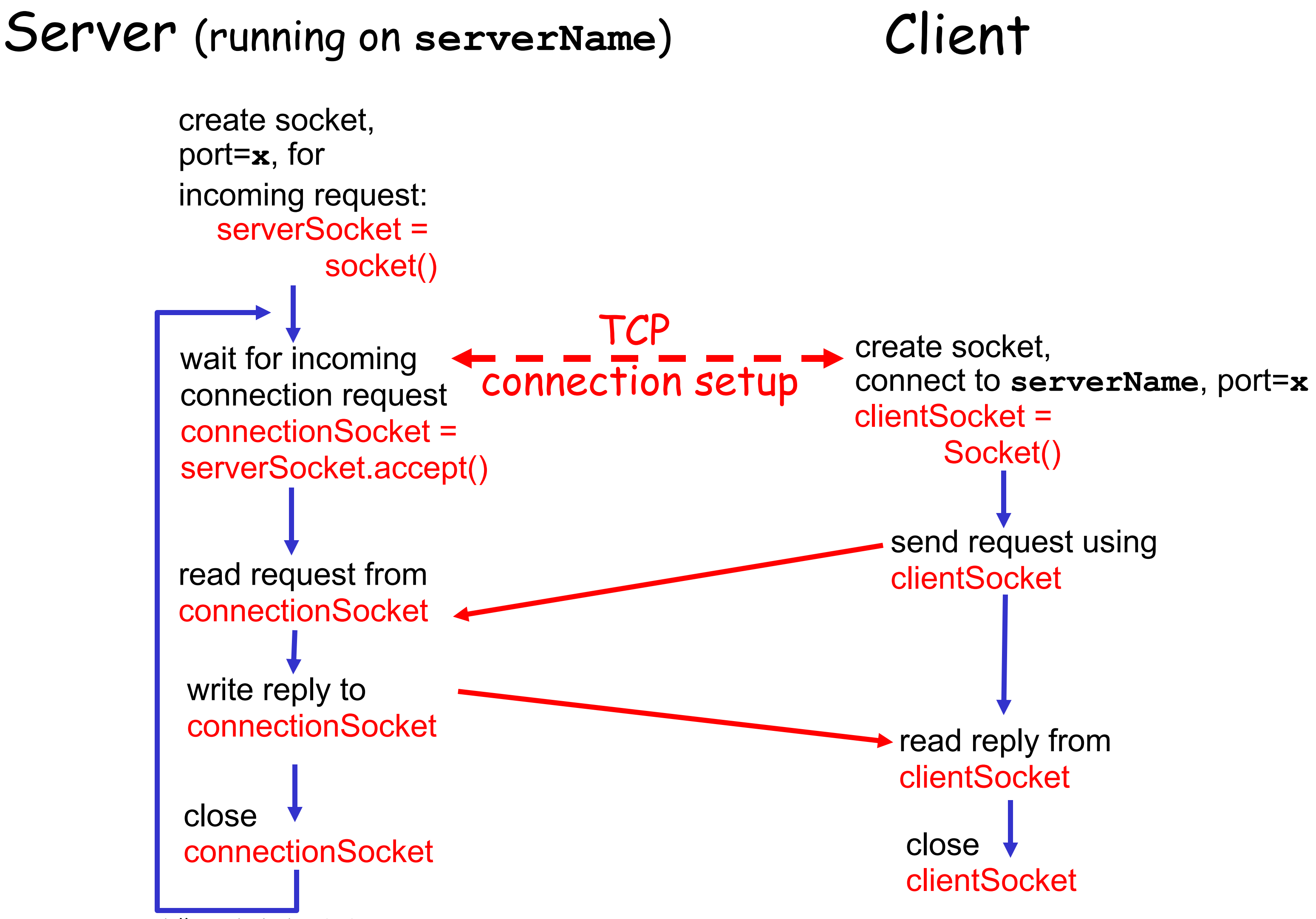
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

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

application viewpoint

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

Client/server socket interaction: TCP


Server (running on serverName)
Client
create socket, port=x, for incoming request:
serverSocket = socket()
wait for incoming connection request
connectionSocket = serverSocket.accept()
TCP connection setup
create socket, connect to serverName, port=x
clientSocket = Socket()
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

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

create TCP socket

Connect to server at port using 12000

No need to attach server name/port when sending

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 the 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 1:
    connectionSocket, addr = serverSocket.accept()

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

TCP observations & questions

- Server has two types of sockets:
  - ServerSocket and connectionSocket
- When client "knocks" on serverSocket's "door," server creates connectionSocket and completes TCP connection setup
- Can multiple clients use the server?

CS2505 labs

- The labs complement this section of the course by allowing you to write your own simple client/server programs using
  - Python
  - UDP
  - TCP