CIS 457 - Data Communications

Nathan Bowman

Images taken from Kurose and Ross book

Code taken from Kurose and Ross book with modifications

Socket Programming

Working in a layered architecture makes writing networked applications much more straightforward than it would at first seem

Choose transport protocol (usually TCP) and assume the messages will be sent correctly

Programmer's job is to:

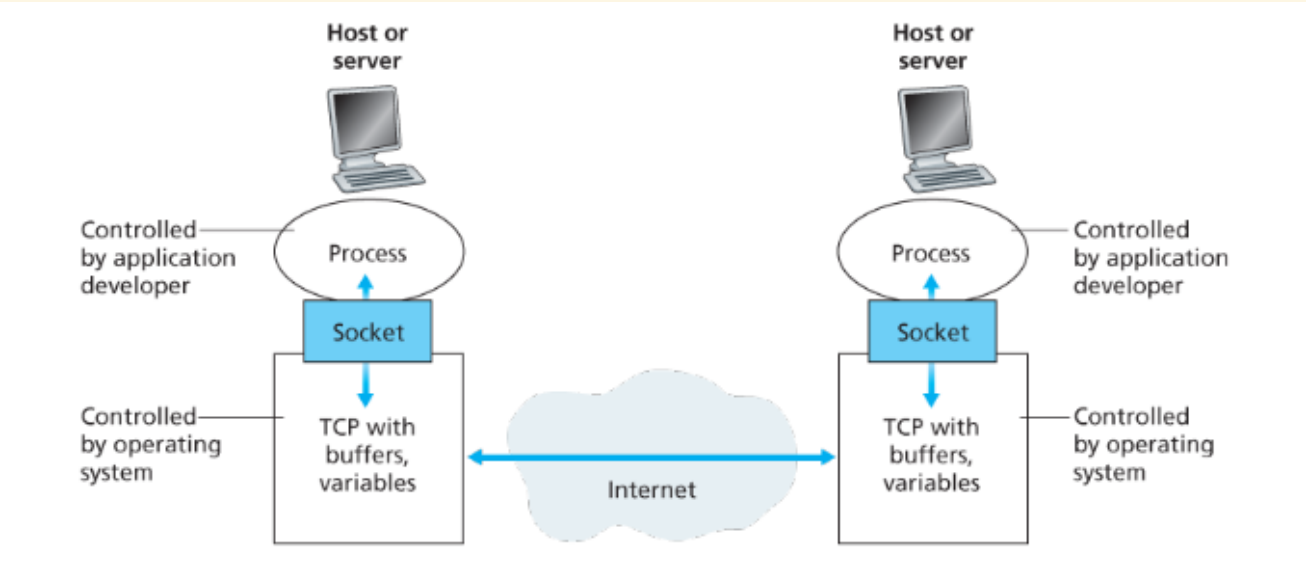
- create (or implement existing) application-layer protocol
- implement application logic

Recall that **socket** is interface between application and transport layers

Programming languages have APIs to implement sockets

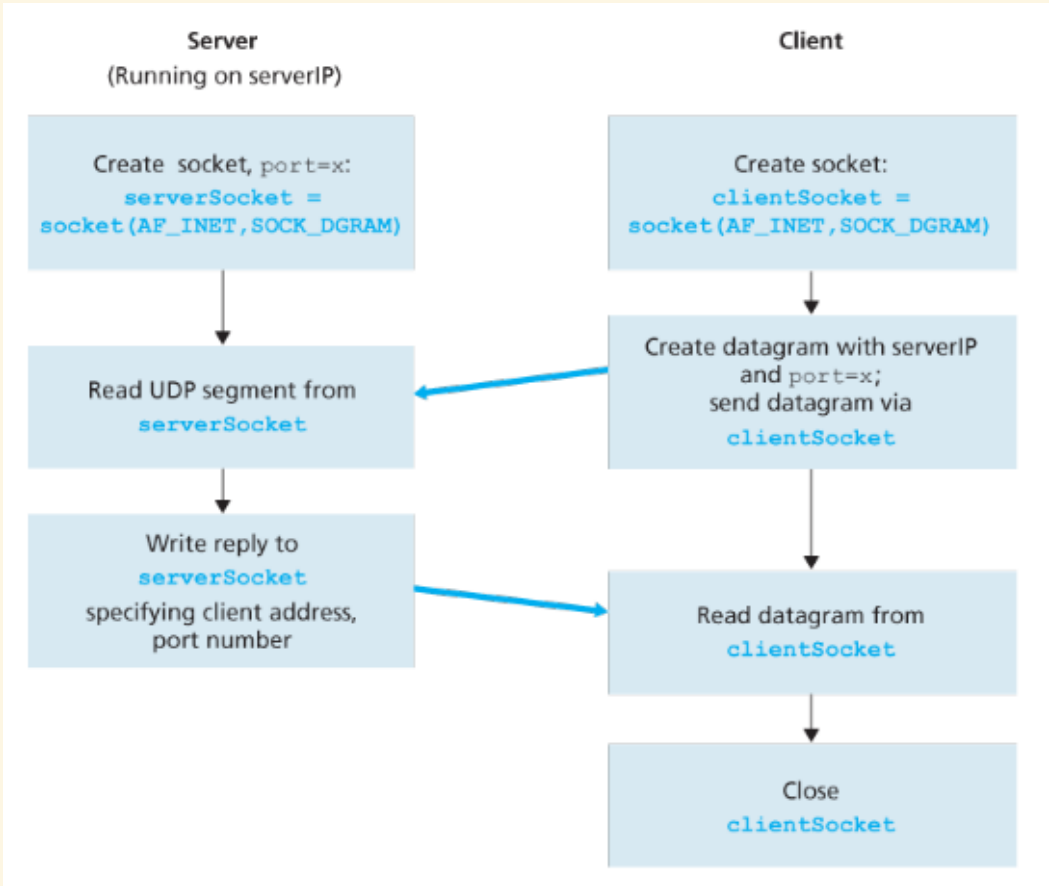
Application programmers concerned only with general idea of sockets, not implementation details

From application programmer's perspective, socket simply sends and/or receives data


Host or server
Host or server
Controlled by application developer
Process
Process
Controlled by application developer
Socket
Socket
Controlled by operating system
TCP with buffers, variables
TCP with buffers, variables
Controlled by operating system
Internet

Example application is server that accepts lowercase sentence and returns uppercase sentence

Examples in both UDP and TCP are shown


Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP
and port=x;
send datagram via
clientSocket
Read UDP segment from
serverSocket
Write reply to
serverSocket
specifying client address,
port number
Read datagram from
clientSocket
Close
clientSocket

UDP client

```
import socket


server_name = 'localhost'
server_port = 12000
buffer_size = 2048

client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
message = input('Input lowercase sentence:')

client_socket.sendto(message.encode('utf-8'), (server_name, server_port))
modified_message, server_address = client_socket.recvfrom(buffer_size)
print(modified_message.decode('utf-8'))
client_socket.close()
```

UDP server

```
import socket



server_ip = 'localhost'
server_port = 12000
buffer_size = 2048

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((server_ip, server_port))

print('The server is ready to receive')

while True:
    message, client_address = server_socket.recvfrom(buffer_size)
    modified_message = message.decode('utf-8').upper()
    server_socket.sendto(modified_message.encode('utf-8'), client_address)
```

Note about encodings

Client and server programs can be:

- running on machines with different architectures
- running on systems with different default languages
- written in different programming languages

There is no such thing as "a string"

Clients and servers are sending *bytes* back and forth, and they need to agree how to interpret them

Note about encodings

Same could be said for integers and other data types, and there are functions to convert to these to machine-independent encodings as well

TCP is slightly more complicated because of connections

Each TCP connection represented by an individual socket

One TCP socket per 4-tuple

```
(server IP, server port, client IP, client port)
```

This means there can be many sockets attached to one server port if there are many clients connected

(Don't confuse the ideas of "port" and "socket" -- they are related, but not the same)

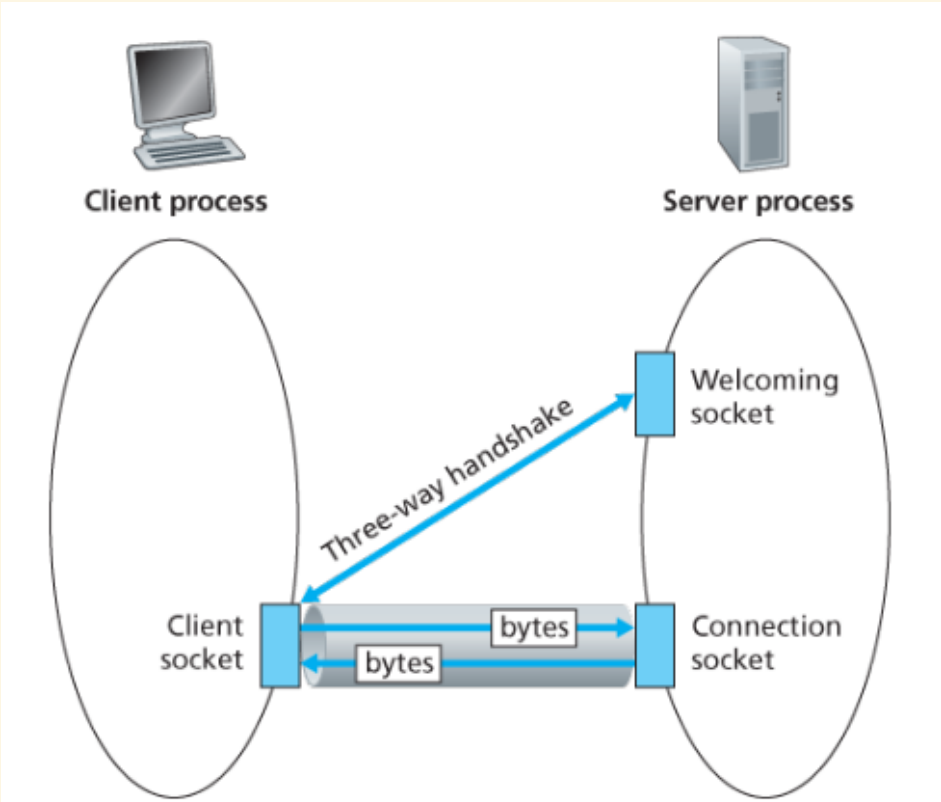
There is one socket that is "pre-connection"

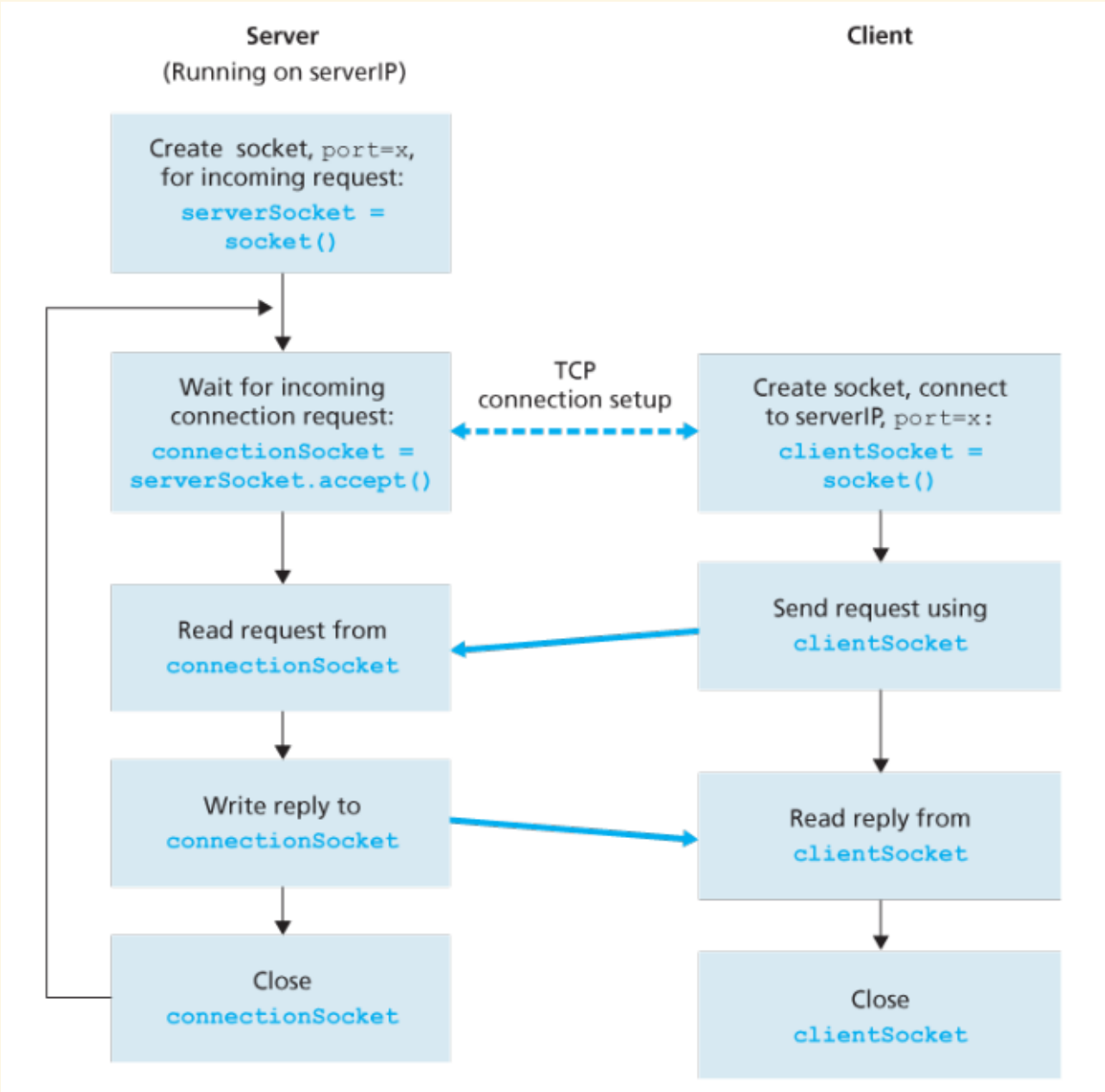
This *welcoming socket* listens for incoming connections, then creates a new *connection socket* per connection

Welcoming socket exists to perform TCP three-way handshake (discussed in future topic)

Welcoming socket never exchanges application-layer information with client

All application-layer information flows between client socket and connection socket


Client process
Server process
Welcoming socket
Three-way handshake
Client socket
bytes
bytes
Connection socket


Server
(Running on serverIP)
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 serverIP, port=x:
clientSocket = socket()
Read request from connectionSocket
Send request using clientSocket
Write reply to connectionSocket
Read reply from clientSocket
Close connectionSocket
Close clientSocket

TCP client

```
import socket


server_name = 'localhost'
server_port = 12000
buffer_size = 1024

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_name, server_port))

sentence = input('Input lowercase sentence:')
client_socket.send(sentence.encode('utf-8'))

modified_sentence = client_socket.recv(buffer_size)
print('From Server: ', modified_sentence.decode('utf-8'))
client_socket.close()
```

TCP server

```
import socket


server_ip = 'localhost'
server_port = 12000
buffer_size = 1024

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port))

server_socket.listen()

print('The server is ready to receive')

while True:
    connection_socket, addr = server_socket.accept()
    sentence = connection_socket.recv(buffer_size).decode('utf-8')
    modified_sentence = sentence.upper()
    connection_socket.send(modified_sentence.encode('utf-8'))
    connection_socket.close()
```

To be able to listen for incoming connections while working, server must be multi-threaded

You learn more about threads in OS, but it essentially means the program can do two or more things at once

Moving to a threaded server does not need to be difficult

Multithreaded TCP server

```
import socket
import threading


def upper_case(connection_socket):
    sentence = connection_socket.recv(buffer_size).decode('utf-8')
    modified_sentence = sentence.upper()
    connection_socket.send(modified_sentence.encode('utf-8'))
    connection_socket.close()


server_ip = 'localhost'
server_port = 12000
buffer_size = 1024

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port))

server_socket.listen()

print('The server is ready to receive')

while True:
    connection_socket, addr = server_socket.accept()
    threading.Thread(target=upper_case, args=(connection_socket,)).start()
```