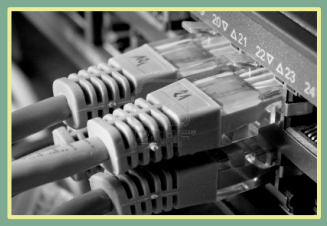
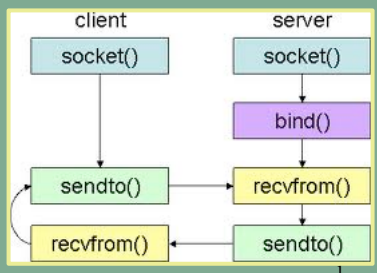


The Application Layer: Sockets Wrap-Up

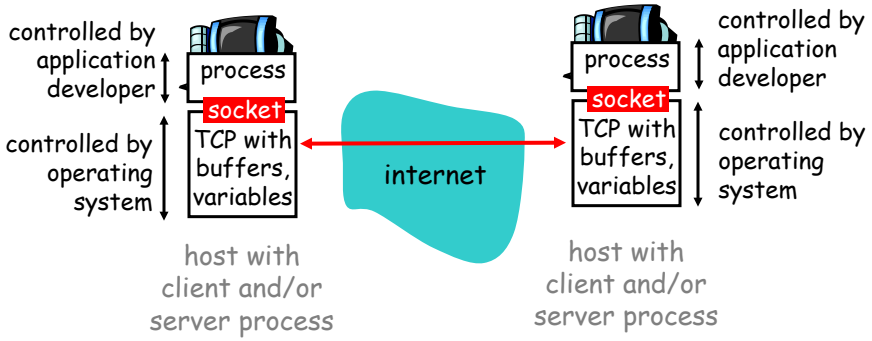
CSC 249
February 13, 2018



Overview

- Review the Socket API
  - Defined for UNIX
  - Used by most operating systems
- Review TCP and UDP examples and flow charts
- Methods for socket programming
- Outline an SMTP server

Socket programming



3

Socket Programming

- Sockets are used to send data from one host to another
  - Sockets provide an interface between the application and the Internet
- Socket programming is analogous to simple file I/O
- In UNIX, all devices are file abstractions
  - Open, close, read, write
  - Sockets are simply one more file abstraction

4

Sockets

- The API is used for communicating between a Client and a Server
- Client
  - Active participant in communication
  - Initiates conversations and sends data
- Server
  - Passively listens and waits for data
- Socket
  - Protocol to use?
  - Identifier of the other machine (IP + port)?
  - Client or server?

5

Connection-Oriented → TCP

- The message is only transferred after a connection has been made
  - Connection creates a virtual pipe between the client and the server such that each knows the other's IP address and protocol port number
- Both parties know that a message will be communicated
- No need to tell destination (IP address and port number) in subsequent messages
  - Because there is a connection!

6

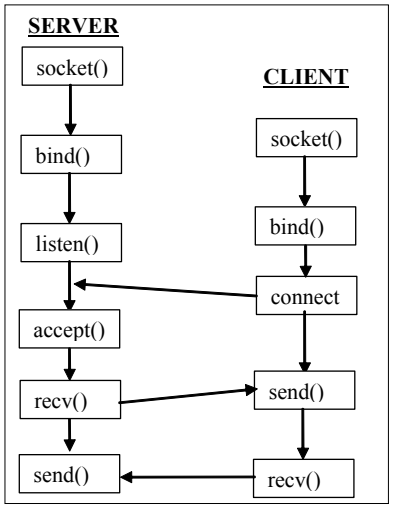
Connectionless → UDP

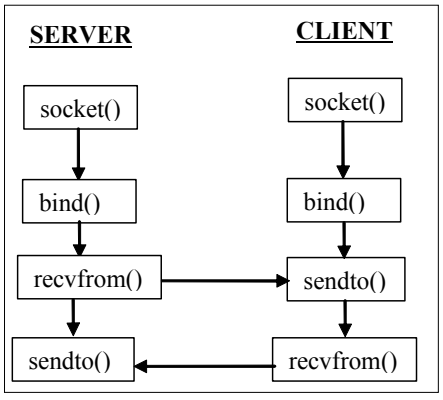
- Send Individual Messages
  - as opposed to a continuous byte stream
- Socket has to be told where to send the message every time
  - Destination IP address and Port number
- Overhead data flow can get excessive if a large number of messages need to be sent between the same hosts

7

Socket Flowcharts

- TCP vs. UDP





8

Review Server steps

- All servers begin by making a function call to "socket()" to create a socket and "bind()" to specify a protocol port number
- UDP: the server is now ready to accept messages
- TCP: additional steps to become ready are
  - Server calls listen() to place the socket in passive mode
  - Server calls accept() to accept a connection request if it comes in

9

Socket programming *with TCP*

The order of steps for using sockets with TCP

- Server process must be running *first*
- Then the client can create a socket, which causes…
  1) DNS lookup for server IP address
  2) TCP to establish connection between the client and server
  3) Which causes the server process to create a new, dedicated socket for this specific client process

10

Socket programming *with TCP*

- Client
  1) Creates message – as a byte stream
  2) Sends the message into its socket
- TCP takes over and delivers the message
  - Guarantees delivery
  - With bytes delivered in the original order
- Server process performs its application duties and sends a response message through its socket…

11

```
# Example to connect to google
from socket import *

print ("Creating Socket...")
s = socket(AF_INET, SOCK_STREAM)
print ("done.")

print ("Looking up port number...")
port = getservbyname('http', 'tcp')
print ("done.")

print ("Connect to remote host on port %d" %port),
s.connect (("www.google.com", port))
print ("done.")

print "Connected from", s.getsockname()
print "Connected to", s.getpeername()
```

12

```
# Client example 2: client2.py
# Run the client after the server is running

from socket import * # Import socket module

s = socket()         # Create a socket object
host = gethostname() # Get local machine name
port = 12345         # Assign a port

print ("Client host is ", host)

s.connect((host, port))
print (s.recv(1024))

s.close          # Close the socket when done
```

13

```
# Example 2: Server2.py
from socket import *

s = socket()         # Create a socket object
host = gethostname() # Get local machine name
port = 12345         # Assign a port number

s.bind((host, port)) # Bind to the port
print ("Server host is ", host)
s.listen(1)          # Wait for client conx

while True:
     c, addr = s.accept() # conx to client
     print ('Got connection from', addr)
     c.send('Thank you for connecting')
     c.close()       # Close the connection
```

14

```
# Example 3: client3.py
from socket import *

HOST = 'localhost'
PORT = 29876
ADDR = (HOST,PORT)
BUFSIZE = 4096

cli = socket(AF_INET,SOCK_STREAM)
cli.connect((ADDR))

data = cli.recv(BUFSIZE)
print (data)

cli.close()
```

15

```
# Example 3: server3.py
from socket import *

HOST = ''          # Use the local host
PORT = 29876       # Assign a port number
ADDR = (HOST,PORT) # define a tuple for the address
BUFSIZE = 4096     # Define buffer for data

# Create a new socket object (serv)
serv = socket( AF_INET,SOCK_STREAM)

# Bind our socket to the address
serv.bind((ADDR))    # Define an address 'tuple'
serv.listen(5)       # Allow 5 connections
print ('listening...')

conn,addr = serv.accept()
print ('...connected!')
conn.send('TEST')
conn.close()
```

16

HW: Web Server

- Develop a web server that handles one HTTP request at a time.
  - Accept and parse the HTTP request message,
  - Get the requested file from the server's file system
  - Create an HTTP response message consisting of the requested file and the appropriate header lines
  - Send the response directly to the client.
  - Use any web browser for the client

17

HW: Web Server Due Dates

- Feb 15
  - The HTML code that your web server will serve up to your requesting web browser
  - A first draft of your Python server code
- Feb 22
  - Python (or other) working server code
    - Beautifully commented with meaningful variable and object names
  - Screen shots of output

18