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

Discussion: Socket Programming

- Sockets are simply a file abstraction in UNIX - what does this mean?
- For the client-server architecture, one is passive and one is active.
  - Which is which?
  - What are their different tasks (methods)?
- Where and when are IP addresses and port numbers used in TCP vs. UDP sockets?
Socket API Overview

- Sockets are used to send data from one host to another
- In UNIX, all devices are file abstractions
  - Open, close, read, write
- Sockets are simply one more file abstraction

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?

Connection-Oriented \(\rightarrow\) 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!

Connectionless \(\rightarrow\) 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
Class Example: SMTP Client (5 minutes now, to come back to toward end of class)

- Develop a simple mail client that sends email to any recipient → a first attempt
  1) Recall the telnet practice with SMTP
  2) Connect to a mail server, dialogue with the mail server using the SMTP protocol
  3) Send an email message to the mail server.
     - Python provides smtplib, with built in methods, but this hides the details of SMTP and socket programming → so do not use this

- To limit spam, mail servers do not accept TCP connections from arbitrary sources.

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

Socket Flowcharts

- TCP vs. UDP

Socket API Overview - details in appendix to these slides

- Socket Programming Procedures
  - Socket()
  - Bind()
  - Listen()
  - Accept()
  - Connect()
  - Along with send and receive procedures
  - Close()

- And for DNS...
  - getHostByName
  - getServByName
  - getProtoByName
# 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()

# 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

# 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

# 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()
# 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()  # Endpoint to which we are listening
print ('...connected!')
conn.send('TEST')
conn.close()

---

## Mail message format

* Example of the actual message - NOT part of the SMTP handshaking process

- header lines, e.g.,
  - To:
  - From:
  - Subject: different from SMTP commands!

- body
  - the "message", ASCII characters only

---

### Class Example: SMTP Client

- Develop a simple mail client that sends email to any recipient  
  a first attempt
  1. Recall the telnet practice with SMTP
  2. Connect to a mail server, dialogue with the mail server using the SMTP protocol,
  3. Send an email message to the mail server.

- To limit spam, mail servers do not accept TCP connection from arbitrary sources.
  - You could try connecting both to both the Smith mail server and to a popular Webmail server, such as an AOL mail server, gmail...

```bash
fcapmaster:~ jcardell$ telnet smtp.smith.edu 25
Trying 131.229.64.236...
Connected to baton.smith.edu.
Escape character is '^]'.
220 baton.smith.edu ESMTP Sendmail ...
C: HELO jbc.edu
S: 250 baton.smith.edu Hello [131.229.102.128], pleased to meet you
C: MAIL FROM: <judy@jbc.edu>
S: 250 2.1.0 <judy@jbc.edu>... Sender ok
C: RCPT TO: <jcardell@smith.edu>
S: 250 2.1.5 <jcardell@smith.edu>... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: to: Easter@Bunny.hop
C: from: Tooth@Fairy.fly
C: subject: How’s Business?
C: 
C: Checking in to see how you’re doing.
C: 
S: 250 2.0.0 s6GFb0Q4007216 Message accepted for delivery
C: QUIT
S: 221 2.0.0 baton.smith.edu closing connection
```
#Sample SMTP client program -> server refuses contact from socket import *

# Messages to send
msg = '\r\nHello World!'   endmsg = '\r\n\r\n'

# Choose a mail server and call it mailserver
mailserver = 'smtp.smith.edu'

# Create socket, establish a TCP conx with mailserver
clientSocket = socket(AF_INET, SOCK_STREAM)

# Port number may change according to the mail server
clientSocket.connect((mailserver, 25))
recv = clientSocket.recv(1024)
print(recv)
if recv[:3] != '220':
    print ('220 reply not received from server.')
    quit()

# Send HELO command and print server response.
hecloCommand = '\HELO smith.edu\r\n'
clientSocket.send(hecloCommand)
recv1 = clientSocket.recv(1024)
print recv1
if recv1[:3] != '250':
    print ('250 reply not received from server.')
    quit()

# Send DATA command and print server response.
data = 'DATA\r\n'
clientSocket.send(data)
recv4 = clientSocket.recv(1024)

# Message ends with a single period.
clientSocket.send(endmsg)

# Send QUIT command and get server response.
quitcommand = '\QUIT\r\n'
clientSocket.send(quitcommand)

---

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

---

HW: Web Server Due Dates

- **Feb 24**
  - Python (or other) working code in 2 weeks
  - Web server code, beautifully commented with meaningful variable and object names
  - Screen shots of output

- **For Feb 17**
  - The HTML code that your web server will serve up to your requesting web browser
  - (you will use a commercial web browser to contact your own web server)
  - **BE SURE** to be working on your web server this first week as well!
Appendix - Socket functions

Procedures: Socket()

- descriptor = socket(protoFamily, type)
  - Creates a socket and returns an integer descriptor
  - ProtoFamily - refers to Family of protocols that this protocol belongs to, for TCP/IP use PF_INET
  - Type - SOCK_STREAM, SOCK_DGRAM
    - SOCK_STREAM - Connection Oriented (TCP)
    - SOCK_DGRAM - Connectionless (UDP)

Close()

- The socket is no longer going to be used
- Close(sock)
  - Sock - the descriptor
- Note: For a connection oriented socket, connection is terminated before socket is closed

Bind()

- Bind(socket, localAddr, addrLen)
  - Call after socket() has been called to bind the socket to a protocol port number
  - Used to assign the port at which the client/server will be waiting for connections/messages
    - The port number is part of the address structure
    - s.bind(('', 80)) specifies that the socket is reachable by any address the machine happens to have
  - Socket - descriptor
  - localAddr - socket address structure including the port number
  - addrLen - length of the address
Listen() - Server Procedure

- Listen(socket, queue size)
  - Called at server
  - socket - descriptor at server
  - queueSize - buffering of requests

- This procedure tells the server to leave a socket running, in passive mode, at this port.

Accept() - Server Procedure

- Newsock = accept(socket, caddr, caddrLen)
  - Accept() fills the fields of the struct caddr with the address of the client that formed the connection
  - Accept() creates a new socket for this connection and returns the descriptor of this new socket
  - The server’s original “listen()” socket remains unchanged

- A request has come to the server
  - The phone is ringing
- Accept picks up the connections (only TCP)

Connect() - Client Procedure

- Connect(socket, saddr, saddrlen)
  - Arguments ‘socket’ is the descriptor of a socket on the client's computer to use for the connection
  - ‘saddr’ and len specify the server’s info
  - With TCP, this initiates the connection to the specified server

- This is used to make the “phone call”

- Two uses
  - Connection-oriented transport - make the call
  - Possible use - Connectionless - identify the server to send the many, independent messages

Send() and Sendto()

- Used to send packets from one host to another
  - Send(socket, data, length, flags)
    - Socket - descriptor
    - Data - pointer to buffer in memory with the data
    - Length - of data to be sent
    - Flags - for debugging, not general use (typ = 0)

- Sendto() is used with an unconnected socket
  - Sendto(socket, data, length, flags, destAddress, addressLen)
Recv() and Recvfrom()

- Used to receive messages in a connection oriented communication
  - **Recv**(socket, buffer, length, flags)
    - Buffer - memory location/structure to store the data
    - Length - the length of buffer

- Recvfrom() is used in connectionless communication
  - **Recvfrom**(socket, buffer, flags, sndraddr, saddrlen)
    - Sndraddr - sender’s address
    - Saddrlen - length of sender’s address