Lab 4

Networking

Prof. Steven Ludtke
N410.07, sludtke@bcm.edu

Simple Python Webserver

```
# This will serve files from the current directory
# we use port 8080 because port 80 is restricted

from BaseHTTPServer import *
from SimpleHTTPServer import *
httpd=HTTPServer(("",8080),SimpleHTTPRequestHandler)
httpd.serve_forever()
```

Socket Module

- gethostname() - name of the current machine
- gethostbyname() - given a name, returns an IP address
- gethostbyaddr() - given an address, returns names and other info
- socket() - create a new socket
  - listen(n) - waits for incoming connections on a socket n>1
  - accept() - accepts an incoming connection, returns tuple with socket
  - connect((host,port)) - connects to a remote socket
  - send(), recv() - send and receive data
  - sendall() - send until success or error
  - makefile() - make a file-like object for the socket
- bind() - bind a socket to an address
- select() - from select module, lets you wait for activity on set of sockets

UDP

```
#receiver
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
s.bind(("",40000))
print s.recv(1000) # up to 1000 bytes

#sender
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
s.bind((“”,40000))
s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
s.sendto(“Hello there”,(“<broadcast>”,40000))
```

Making Connections

```
# Receiver/server
import socket
sock=socket.socket()      # default is to make a normal internet
socket
sock.bind((“”,40000))   # Nothing magic about 40000
sock.listen(1)             # Wait for 'connect' requests
sock2=sock.accept()     # accept the connection (new socket)
print sock2[0].recv(256)   # receive 256 bytes of data


# Sender
import socket
sock=socket.socket()            # default socket
sock.connect((target,40000))  # connect to someone listen()ing
sock.send(“Hello there”)       # send a string
```

Socket vs File Obj.

- Socket:
  - send - will transmit data immediately
  - recv - specifies maximum amount to receive. May not get all sent data in one call. Call multiple times until you have what you expected
- File:
  - write - buffers. Need to call flush() for immediate transmission.
  - read/readline - Reads the full amount expected. Multiple calls not required.

Chat Program

Write a multiuser chat program, where anything typed by one user is displayed on all other users's displays

client <-> server

Chat Program

- Server
  - Make a socket, bind to port
  - Loop forever, listening for connections
    - Add the new socket from the connection to a list
  - Loop forever to all of the sockets in the list
    - When text comes in on a socket, send it back out to others
- Client
  - Make a socket, connect to server
  - Loop forever
    - Read input from user
    - Send to server on socket
    - Quit if correct string is typed
  - Loop forever
    - Read from server socket
    - print received string
    - Quit if correct string is received

Threads

How to do more than one thing at a time:

```
from threading import Thread
def func():
  for i in range(30):
    time.sleep(1)
    print i
thread=Thread(target=func,args=(1,2,3...))   # create a new thread
thread.start()
```

Lab 4

- Download the 3 sample programs from the class website, and the PDF of the lecture (for reference)
- Connect to the "IP15" wireless network. You will not have general network access, this will only allow you to communicate with other computers in the room.
- run udp_chat.py. You can type messages which will appear on your own screen as well as your neighbors. When you're done, type 'exit'
- Look over udp_chat.py to get a feel for how it works.
- There are 2 programs to modify today: tcp_send.py and tcp_receive.py. Pick the most experienced programmer in your group to do tcp_receive. The others can work on tcp_send.

Lab 4

- Test tcp_send and tcp_receive. Have one person run tcp_receive. This will print a network address on the screen. The others run tcp_send, and enter this address, then a message. The messages should appear on the receiver's machine. When someone sends the receiver an 'exit' message, the receiver will quit.

- Modify tcp_send and tcp_receive to act as a simple file transfer system. It should only transfer small (few thousand characters) files.

  - tcp_receive should accept the name of a file to write to (feel free to limit this in appropriate ways for security if you like), followed by the file itself. It can exit after each file, or continue.

  - tcp_send should send one transmission with the filename, then a second transmission containing the contents of the file. The file may be a real file on the sending computer, or just a string with a made up filename. (remember sendall)

  - Note - this will be trickier than it seems at first. You will need to do something about denoting the end of the filename and the start of the file.