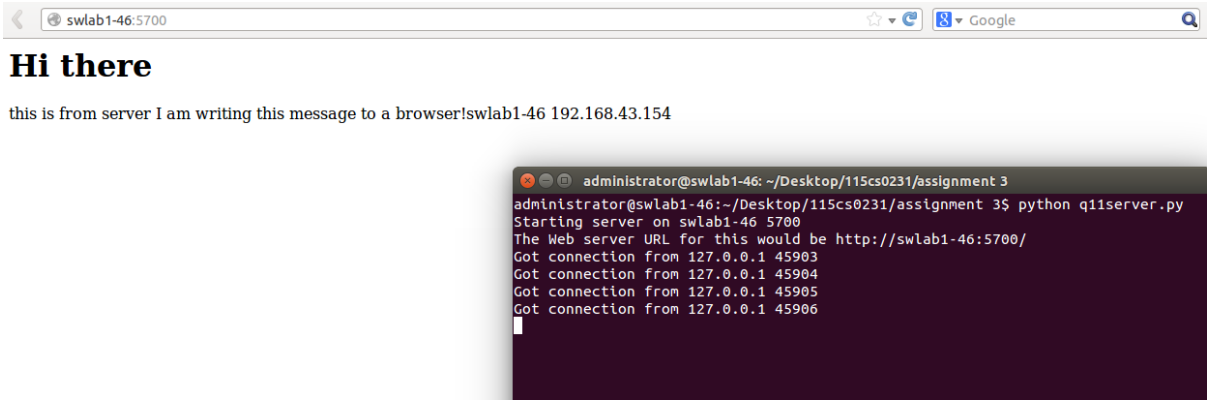
ASSIGNMENT 2

11. Write a simple web server that can return a single line/multiple line of text to any connected web browser.

```
import socket
import fcntl
import struct

def get_interface_ip(ifname):
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        return socket.inet_ntoa(fcntl.ioctl(s.fileno(), 0x8915,
struct.pack('256s',ifname[:15]))[20:24])
host = socket.gethostname()
port = 5700

s = socket.socket()  # get instance
# look closely. The bind() function takes tuple as argument
s.bind((host, port))  # bind host address and port together

# configure how many client the server can listen simultaneously
s.listen(4)
print 'Starting server on', host, port
print 'The Web server URL for this would be http://%s:%d/' % (host, port)

while True:
   c, (client_host, client_port) = s.accept()

   c.recv(1000)
   c.send('HTTP/1.0 200 OK\n')
   c.send('Content-Type: text/html\n')
   c.send('\n')
   c.send("""
   <html>
   <body>
   <h1>Hi there</h1> this is from server I am writing this message to a
browser!"""+host+" "+ get_interface_ip('eth0')+"  "+"""
   </body>
   </html>
   """)
   c.close()
   print 'Got connection from', client_host, client_port
```



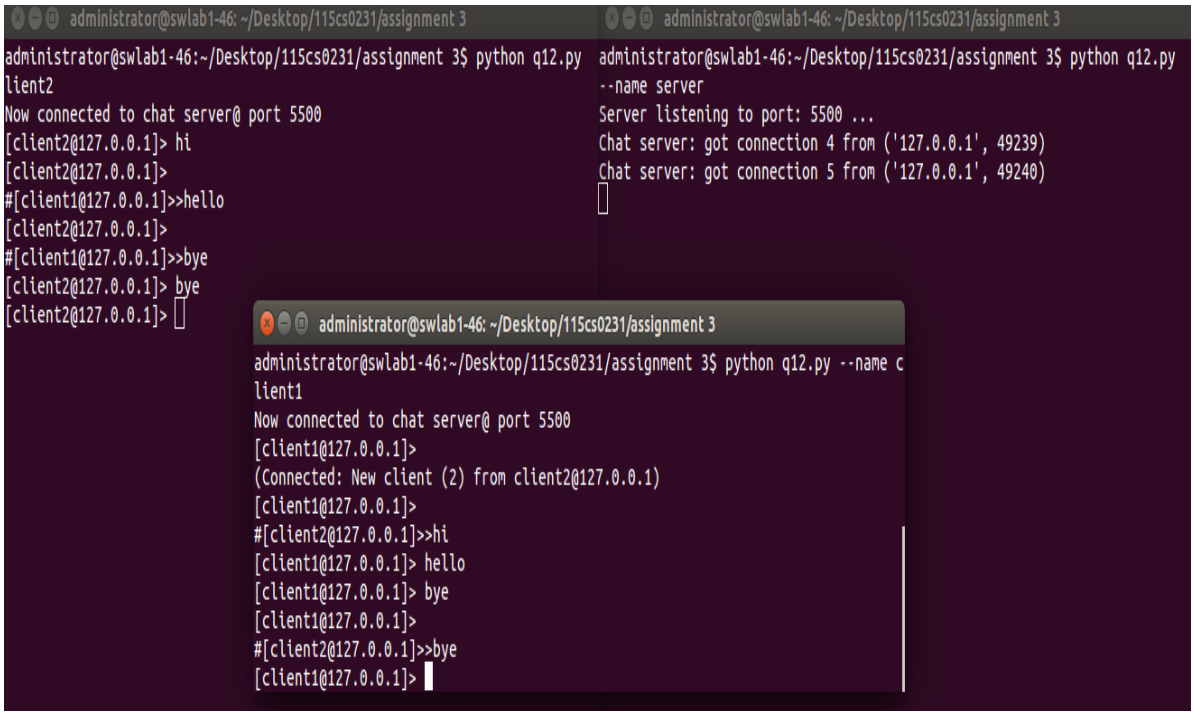
12.Write an efficient chat server that can handle several hundred or a large number of client connections. The chat server initializes with a few data attributes. It stores the count of clients, map of each client, and output sockets. The chat client initializes with a name argument and sends this name to the chat server.

```
import select
import socket
import sys
import signal
import pickle
import struct
import argparse

SERVER_HOST = 'localhost'
CHAT_SERVER_NAME = 'server'

# Some utilities
def send(channel, *args):
    buffer = pickle.dumps(args)
    value = socket.htonl(len(buffer))
    size = struct.pack("L",value)
    channel.send(size)
    channel.send(buffer)

def receive(channel):
    size = struct.calcsize("L")
    size = channel.recv(size)
    try:
        size = socket.ntohl(struct.unpack("L", size)[0])
    except struct.error as e:
        return ''
    buf = ""
    while len(buf) < size:
        buf = channel.recv(size - len(buf))
    return pickle.loads(buf)[0]


class ChatServer(object):
    """ An example chat server using select """
    def __init__(self, port, backlog=5):
        self.clients = 0
        self.clientmap = {}
        self.outputs = [] # list output sockets
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server.bind((SERVER_HOST, port))
        print ('Server listening to port: %s ...' %port)
        self.server.listen(backlog)
```

```
        # Catch keyboard interrupts
        signal.signal(signal.SIGINT, self.sighandler)

    def sighandler(self, signum, frame):
        """ Clean up client outputs"""
        # Close the server
        print ('Shutting down server...')
        # Close existing client sockets
        for output in self.outputs:
            output.close()
        self.server.close()

    def get_client_name(self, client):
        """ Return the name of the client """
        info = self.clientmap[client]
        host, name = info[0][0], info[1]
        return '@'.join((name, host))

    def run(self):
        inputs = [self.server, sys.stdin]
        self.outputs = []
        running = True
        while running:
            try:
                readable, writeable, exceptional = select.select(inputs, self.outputs,
[])
            except select.error as e:
                break

            for sock in readable:
                if sock == self.server:
                    # handle the server socket
                    client, address = self.server.accept()
                    print ("Chat server: got connection %d from %s" % (client.fileno(),
address))
                    # Read the login name
                    cname = receive(client).split('NAME: ')[1]

                    # Compute client name and send back
                    self.clients += 1
                    send(client, 'CLIENT: ' + str(address[0]))
                    inputs.append(client)
                    self.clientmap[client] = (address, cname)
                    # Send joining information to other clients
                    msg = "\n(Connected: New client (%d) from %s)" % (self.clients,
self.get_client_name(client))
                    for output in self.outputs:
                        send(output, msg)
                    self.outputs.append(client)

                elif sock == sys.stdin:
                    # handle standard input
                    junk = sys.stdin.readline()
                    running = False
                else:
                    # handle all other sockets
                    try:
                        data = receive(sock)
                        if data:
```

```
                    # Send as new client's message...
                    msg = '\n#[' + self.get_client_name(sock) + ']>>' + data
                    # Send data to all except ourself
                    for output in self.outputs:
                        if output != sock:
                            send(output, msg)
                else:
                    print ("Chat server: %d hung up" % sock.fileno())
                    self.clients -= 1
                    sock.close()
                    inputs.remove(sock)
                    self.outputs.remove(sock)

                    # Sending client leaving information to others
                    msg = "\n(Now hung up: Client from %s)" %
self.get_client_name(sock)
                    for output in self.outputs:
                        send(output, msg)
            except socket.error as e:
                # Remove
                inputs.remove(sock)
                self.outputs.remove(sock)
    self.server.close()


class ChatClient(object):
    """ A command line chat client using select """

    def __init__(self, name, port, host=SERVER_HOST):
        self.name = name
        self.connected = False
        self.host = host
        self.port = port
        # Initial prompt
        self.prompt='[' + '@'.join((name, socket.gethostname().split('.')[0])) + ']> '
        # Connect to server at port
        try:
            self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.sock.connect((host, self.port))
            print ("Now connected to chat server@ port %d" % self.port)
            self.connected = True
            # Send my name...
            send(self.sock,'NAME: ' + self.name)
            data = receive(self.sock)
            # Contains client address, set it
            addr = data.split('CLIENT: ')[1]
            self.prompt = '[' + '@'.join((self.name, addr)) + ']> '
        except socket.error as e:
            print ("Failed to connect to chat server @ port %d" % self.port)
            sys.exit(1)

    def run(self):
        """ Chat client main loop """
        while self.connected:
            try:
                sys.stdout.write(self.prompt)
                sys.stdout.flush()
                # Wait for input from stdin and socket
                readable, writeable,exceptional = select.select([0, self.sock], [],[])
```

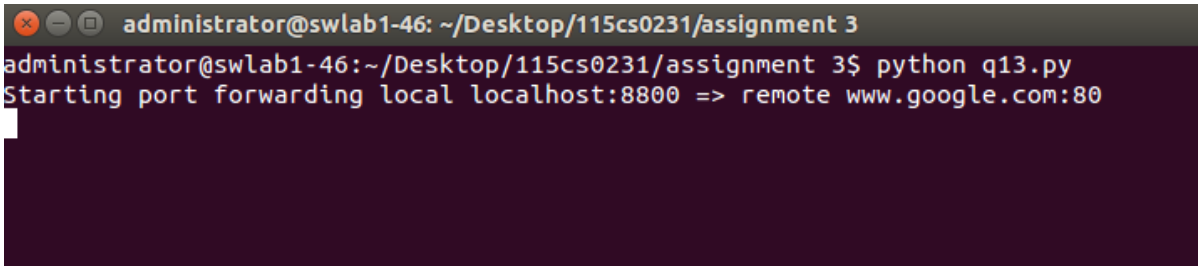
```
            for sock in readable:
                if sock == 0:
                    data = sys.stdin.readline().strip()
                    if data: send(self.sock, data)
                elif sock == self.sock:
                    data = receive(self.sock)
                    if not data:
                        print ('Client shutting down.')
                        self.connected = False
                        break
                    else:
                        sys.stdout.write(data + '\n')
                        sys.stdout.flush()

        except KeyboardInterrupt:
            print (" Client interrupted. """)
            self.sock.close()
            break


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Chat Server')
    parser.add_argument('--name', action="store", dest="name", required=True)

    given_args = parser.parse_args()
    port = 5500
    name = given_args.name
    if name == CHAT_SERVER_NAME:
        server = ChatServer(port)
        server.run()
    else:
        client = ChatClient(name=name, port=port)
        client.run()
```



13.Write program for local port forwarder, that will redirect all traffic from a local port to a particular remote host?

```
BUFSIZE = 4096

import asyncore
import socket
class PortForwarder(asyncore.dispatcher):
	def __init__(self, ip, port, remoteip,remoteport,backlog=5):
		asyncore.dispatcher.__init__(self)
		self.remoteip=remoteip
		self.remoteport=remoteport
		self.create_socket(socket.AF_INET,socket.SOCK_STREAM)
		self.set_reuse_addr()
		self.bind((ip,port))
		self.listen(backlog)
	def handle_accept(self):
		conn, addr = self.accept()
		print "Connected to:",addr
		Sender(Receiver(conn),self.remoteip,self.remoteport)
class Receiver(asyncore.dispatcher):
	def __init__(self,conn):
		asyncore.dispatcher.__init__(self,conn)
		self.from_remote_buffer=''
		self.to_remote_buffer=''
		self.sender=None
	def handle_connect(self):
		pass
	def handle_read(self):
		read = self.recv(BUFSIZE)
		self.from_remote_buffer += read
	def writable(self):
		return (len(self.to_remote_buffer) > 0)
	def handle_write(self):
		sent = self.send(self.to_remote_buffer)
		self.to_remote_buffer = self.to_remote_buffer[sent:]
	def handle_close(self):
		self.close()
		if self.sender:
			self.sender.close()
class Sender(asyncore.dispatcher):
	def __init__(self, receiver, remoteaddr,remoteport):
		asyncore.dispatcher.__init__(self)
		self.receiver=receiver
		receiver.sender=self
		self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
		self.connect((remoteaddr, remoteport))
	def handle_connect(self):
		pass
	def handle_read(self):
		read = self.recv(BUFSIZE)
		self.receiver.to_remote_buffer += read
```

```
        def writable(self):
                return (len(self.receiver.from_remote_buffer) > 0)
        def handle_write(self):
                sent = self.send(self.receiver.from_remote_buffer)
                self.receiver.from_remote_buffer = self.receiver.from_remote_
                buffer[sent:]
        def handle_close(self):
                self.close()
                self.receiver.close()

if __name__ == "__main__":
        local_host = 'localhost'
        remote_host = 'www.google.com'
        local_port=8800
        remote_port=80

        print "Starting port forwarding local %s:%s => remote %s:%s" %
(local_host, local_port, remote_host, remote_port)


        PortForwarder(local_host, local_port, remote_host, remote_port)
        asyncore.loop()
```



14.Write a client that will wait for a particular network service forever or for a time out?

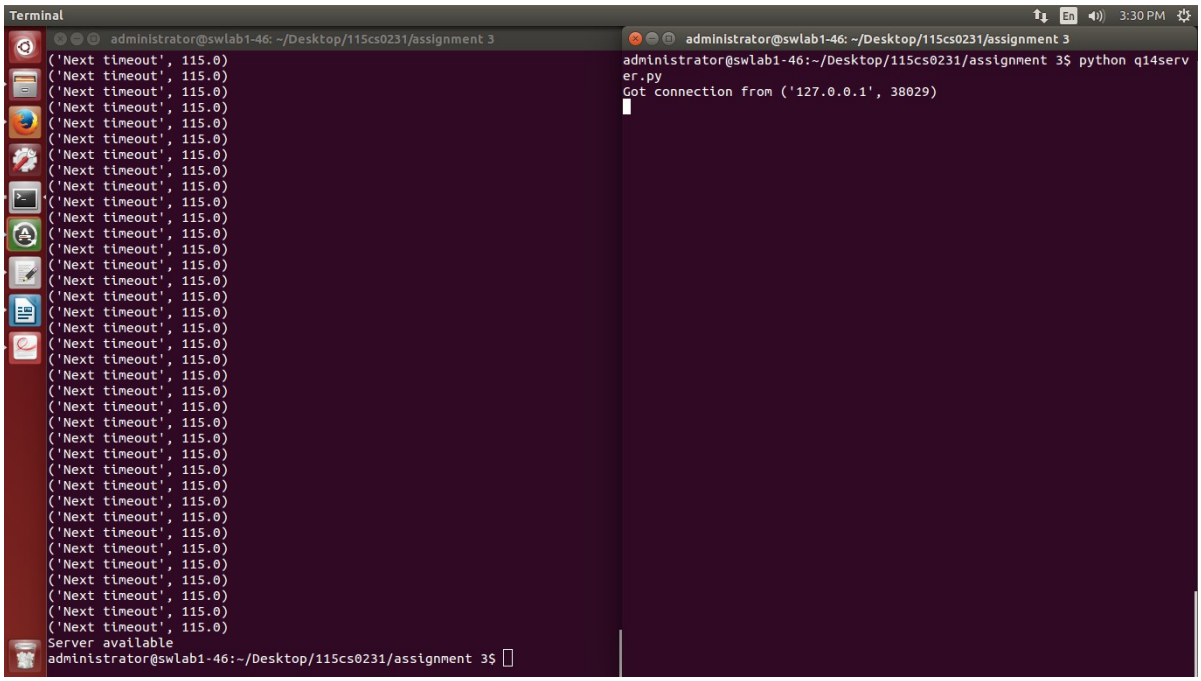
Client Side

```
import select
import socket
from time import time as now
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host='127.0.0.1'
port = 12345
timeout=120
#s.connect(('127.0.0.1', port))
s.settimeout(120.0)
print ("Waiting for ",host, port)
if(timeout):
    end_time=timeout+now()
c=1
while(c==1):
    try:
        if(timeout):
            next_timeout=end_time-now()
```

```
            if next_timeout<0:
                exit()
            else:
                print ("Next timeout" , round(next_timeout))
                s.settimeout(next_timeout)
        s.connect((host, port))

    except socket.timeout, err:
        if timeout:
            exit()
    except socket.error, err:
        c=1#print "Exception"
    else:
        c=0
        print "Server available"
        s.close()
```

Server Side

```
import socket
s = socket.socket()
port = 12345
s.bind(('127.0.0.1', port))
while True:
  c, addr = s.accept()
  print 'Got connection from', addr
```



15.Write a program to list the network interfaces present in your machine?

```
import os
print( os.listdir('/sys/class/net'))
```