CMPSC 381
Data Communications and Networks
Fall 2012
Bob Roos

Lab 4
20 September 2012
Due Thursday, 27 September, 1:30 pm,
in your Sakai dropbox

I strongly suggest that you create a new subdirectory called "lab4" or something similar and place all your files from this lab in that directory. At the end of the assignment I'll ask you to create a "zip" file to upload to the Sakai drop box.

Part 1: File I/O in Python

1. Download the Python programs "`filedemo.py`," "`filedemo2.py`," and "`filedemo3.py`" and the sample text file "`alternative-404.txt`" (any text file will do, of course).

   Run all the Python files to see what they do. Read the code to see how they do it.

   Now write short Python program that does the following (see hints below):

   - Asks the user for a file name
   - Asks the user for a line number in the file (lines are numbered starting at 0)
   - Asks the user for a word number on that line (words are numbered starting at 0)
   - Prints out the requested word on the requested line.
   - If no such file is found, your program should catch an `IOError` exception and print "`File not found`".
   - If the line number or word number are illegal, your program should catch an `IndexError` exception and print "`Illegal index`".

   EXAMPLE:

```
aldenv27:lab4 rroos$ python lab4sol.py
Filename? 404.txt
Which line? 0
Which word in line 0? 0
Alternatives

aldenv27:lab4 rroos$ python lab4sol.py
Filename? 404.txt
Which line? 3
Which word in line 3? 0
See
```

```
aldenv27:lab4 rroos$ python lab4sol.py
Filename? 404.txt
Which line? 3
Which word in line 3? 4
Index out of bounds

aldenv27:lab4 rroos$ python lab4sol.py
Filename? 505.txt
Can't open file
```

Hint: A "`try`" can have multiple "`except`"s, e.g.,

```
try:
  .
  .
  .
except IOError:
  .
  .
  .
except IndexError:
  .
  .
  .
```

Hint: Using "`lines = data.readlines()`" to access a file gives you a list of strings; "`lines[i]`" contains the i-th line. To get individual words, you must use "`split()`" to convert the line into a list of words.

0.1 Web Server

Refer to the (modified) assignment 1 from the book (attached to this). Many of the "`#Fill in`" sections are easily determined from the earlier "`TCPServer.py` example that we looked at in class.

2. Complete the web server code as given in the book's assignment; please make the following changes:

   - Use a port number of 12345, not "6789" as in the book
   - In your server code, right after receiving a message from the client, display the message in the server terminal window. This is the "GET" request, plus any header fields, that came from the browser. You'll need this information to answer one of the questions below.
   - Where the original assignment says "`Send one HTTP header line`," I want you to send several. In particular, I want you to send the header lines at the bottom of page 105 and

the top of page 106 `except` for the "`Content-Length`" header. These can be sent using a sequence of "`send`" commands. Each line should have an explicit "`\n`" character. The last header line should have *two* `\n` characters (in other words, there should be a blank line after the headers.)

- "Personalize" your "`404 not found`" message so that it's clear the message appearing in the browser window is from your server.

3. In your server directory, download the sample HTML files and the image file from the Sakai site. In your browser, view the files "`mypage.html`" and try clicking on the three files. (If you just see the raw HTML then you did something wrong with the headers.) Take a screen shot showing me the navigation bar with your `http://...` request and the result of loading `mypage.html`. Take another screen shot showing me the (personalize) error message from clicking on the `page3` link.

4. Why does the Web server set the filename to "`message.split()[1]`" (refer to the message received from the client, which I asked you to display)?

5. Why does the "`open`" command use "`filename[1:]`"? (again, refer to the message that the client sent to your server)?

6. Is the extra "`\n`" after the last header line important? Why or why not? (You might refer to RFC 822, section 3.1, a 1982 document that specifies the standard for text messages over the Internet.)

Hand In

Make sure your Python program from part 1, your web server, your screen shots, and your answers to the questions are all in your lab4 directory. Move up one level from your `lab4` directory (e.g., "`cd ..`") and type "`zip -r lastnamelab4.zip lab4`" (if you named it something else, use the something else). Now upload this to your drop box.

Be sure your name appears in every document so that when I print it out I'll know who owns that document! (Screen shots are a possible exception.)

Socket Programming Assignment 1: Web Server

In this lab, you will learn the basics of socket programming for TCP connections in Python: how to create a socket, bind it to a specific address and port, as well as send and receive a HTTP packet. You will also learn some basics of HTTP header format.

You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "404 Not Found" message back to the client.

Code

Below you will find the skeleton code for the Web server. You are to complete the skeleton code. The places where you need to fill in code are marked with `#Fill in start` and `#Fill in end`. Each place may require one or more lines of code.

I have created several more HTML files for you to try, plus an image file. All should go in the same directory as the server.

Running the Server

Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example:

Use your server's IP address; use 12345 instead of 6789

http://128.238.251.26:6789/HelloWorld.html

'HelloWorld.html' is the name of the file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number ~~6789~~. The browser should then display the contents of HelloWorld.html. If you omit ":~~6789~~", the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80.

12345

Then try to get a file that is not present at the server. You should get a "404 Not Found" message.

~~What to Hand in~~

~~You will hand in the complete server code along with the screen shots of your client browser, verifying that you actually receive the contents of the HTML file from the server.~~

See lab handout.

Skeleton Python Code for the Web Server

```
#import socket module

from socket import *

serverSocket = socket(AF_INET, SOCK_STREAM)

#Prepare a sever socket

#Fill in start

#Fill in end

while True:

    #Establish the connection

    print 'Ready to serve...'

    connectionSocket, addr =   #Fill in start              #Fill in end

    try:

        message =   #Fill in start          #Fill in end

        filename = message.split()[1]

        f = open(filename[1:])

        outputdata = #Fill in start       #Fill in end

        #Send one HTTP header lines into socket

        #Fill in start

        #Fill in end

        #Send the content of the requested file to the client

         for i in range(0, len(outputdata)):

             connectionSocket.send(outputdata[i])

         connectionSocket.close()

    except IOError:

        #Send response message for file not found

        #Fill in start

        #Fill in end

        #Close client socket

        #Fill in start

        #Fill in end

serverSocket.close
```

Print out the message in order to see what the browser sent to the server!

several