CAMOSUN COLLEGE

COMPUTER SCIENCE DEPARTMENT

COMP 272 – INTRO TO DATA COMMUNICATIONS

LAB 5 – TCP SOCKET PROGRAMMING: WEB SERVER

DEMO Your TCP Web Server during lab time Nov. 10 or Nov. 12
Answers to Questions in Part 3 DUE Nov. 12 by 11:59PM in D2L
Submission Instructions are on the last page.
You may work individually or in pairs for this lab.

Theory:

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 the HTTP header format.

Procedure:

PART 1:

You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request (parsing has already been implemented), 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 a proper HTTP "404 Not Found" message back to the client.

Specifically, your server should:

(1) Bind to a specified port and wait to listen for a client request
(2) Do a "3-way handshake" and create a connectionSocket
(3) Parse the request from the browser (already implemented)
(4) If the file exists on the server, send a HTTP response message. The response message should consist of a proper header and the data. Specifically, the header should contain:
    - the status line (protocol, version number, status code and phrase)
    - a Content-Type: text/html header field (you don't need the charset part).
    - a Connection: close header field (*optional*. You may need this if you find your server crashes. Some setups immediately try to get the *flavicon.ico* icon which will crash your server).
    - no other header fields are necessary.
    - each header line must end with a carriage return and line feed ('\r\n')
    - the entire header must also end in a carriage return and line feed ('\r\n')
(5) After the header line(s), the server should respond with the file contents (already implemented)

(6) If the file does not exist on the server, respond with the proper status code and phrase for this situation and send a basic '404 Not Found' HTML page back to the client (see the Node.js example in Part 2).

Code:

A Python source file, Skeleton_TCPWebServer.py, is provided to you in D2L for this lab. The code is incomplete. Even so, make sure to read through and understand the code that is there. You are to complete this code.

The places you need to fill in the code are marked with **#--TODO--**. Each place may require one or more lines of code.

Running the Server:

A basic HTML file (HelloWorld.html) is provided for you with the skeleton server code in D2L. Put this HTML file in the same directory that the server is in. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). From another host (or during development, you can use localhost), open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HelloWorld.html

Ensure Windows Firewall is turned off (or the port you are using is open) on the host running the TCP Server

'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, the port number is 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.

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

Hints:

- Refer to the TCP example from the Transport Layer Socket Programming slides. **DO NOT COPY AND PASTE CODE FROM THE SLIDES** (use the posted example code instead).
- Refer to the slides from Chapt. 2 Part 1 to assist you in this lab. Specifically, the format of an HTTP response message.
- Remember that the Python Standard Library documentation is at http://docs.python.org/2/library/

PART 2 (Optional – Not graded):

Install Node.js on a machine. Go to nodejs.org and click on the Install or Download button. Leave all options as default. Node.js is a JavaScript based framework that runs atop the V8 JavaScript engine. What's interesting about Node.js is that there is no separation between web server software and server-side code (for example, Apache + PHP). Node.js can do both!

In a plain text file enter the following code:

```
var http = require('http');
var fs = require('fs');
http.createServer(function (request, response) {
    var f = 'HelloWorld.html';
    if (request.url === '/' + f) {
        fs.readFile(f, function (err, data) {
            response.writeHead(200, {'Content-Type': 'text/html'});
            response.end(data);
        });
        return;
    }
    response.writeHead(404);
    response.end('<html><head></head><body><h1>404 Not
Found</h1></body></html>');
}).listen(8080);
```

Name this file **server.js**. This web server should behave in exactly the same way as the TCP web server written in Python (the port number in the code above is 8080 but you can change it to whatever you like). To run this server, in a command prompt, type the following:

```
node server.js
```

PART 3:

- Ensure your browser cache is cleared and fire up Wireshark.
- Run your server code (either the Python one or if you did Part 2, the Node.js one) on some remote machine.
- Start packet capture in Wireshark.
- Request the HelloWorld HTML page through your browser.
- Stop Wireshark capture.

Answer the following questions:

1. Use the Wireshark filter box so that the packet listing window only shows TCP packets and only those TCP packets exchanged with the IP Address of the host your server is running on. What filter statement did you use?

2. What source port number is your client browser using? What is the destination port number? Is this what you expected? Why?

3. Wireshark uses relative sequence numbers. This means that the TCP SYN segment from the client will appear with a sequence number of 0. If you expand Transmission Control Protocol in the Packet Header Details Window and then click on sequence number, you

will see the hexadecimal value of the actual sequence number. You can also expand the flags in the Packet Header Details Window. What flag is being set that denotes this as being a SYN segment from the client?

4. What is the acknowledgment number of the SYNACK segment sent by your server to the client computer in reply to the SYN? Is this what you expected? Why? What flags do you see set in this segment that identifies it as a SYNACK segment?

5. After the 3-way handshake, your client computer will send its GET request and should receive an 'OK' HTTP reply from the server. What is the Acknowledgement number of the OK reply? Why did it jump up by so much? (Hint: ACK#'s are the SEQ# expected from the other side and SEQ#'s are the number of the first byte in the segment's payload. If you look at TCP in the Packet Details Window for the GET request, a Len attribute denotes the number of bytes in the payload of this segment).

6. Expand Transmission Control Protocol in the Packet Header Details Window for whatever packet sends the HTML page to the client. Notice the field that says next sequence number. If the server were to send another packet to the client, this is the sequence number it would use. What is this number?

7. After the 'OK' reply from the server, the client will send an ACK. What is the Acknowledgment number in this packet? Why? (Hint: how big is the HTML file sent to the client)?

Submission:

DEMO your Python server running. From a different client machine, DEMO your browser fetching the HelloWorld.html page. Also demo when a file is not found.

HAND IN the answers to questions 1 to 7 above in either a plain text file or PDF in the lab 5 dropbox in D2L. Only one partner needs to hand in the file. Make sure both partners' names appear in the file. DO NOT compress the file.