CSP334: Computer Networks
Lab Assignment No 7
Socket Programming #2

First Review of your implementation : 1st November, Second Review of your implementation : 8th November 2018
Date of Final submission : 18:00 hrs, 15th November 2018

Autumn Semester 2018

Instructions:

1. *Total marks of the assignment: 240*
2. *Please use only the C as the language of programming.*
3. *Targets in the First Review : Completed programs in problems 1 to 6, Total Max Marks = 120; Targets in the Second Review : Completed problem no 11, Max marks = 40*
4. *Please submit/upload on LMS, the following files for each of your programs: (1) the client and the server source files each (2) the client and the serve executable files each (3) a brief Readme file that shows the usage of the program.*
5. *Please appropriately comment your program and name all the identifiers suitable, to enable enhanced readability of the code.*
6. *Neat coding is necessary as illustrated in the class, that will carry specific marks for each program that you write.*
7. *In the first review (during the lab hours) on the stipulated date, it is expected that a few (if not all) of the assignments in this list are compledted and the execution shown and explained to the teacher in the lab.*

Problems

1. Study the structure viz. struct addrinfo defined as follows. Write a program that takes as a command line argument the name of the host (e.g. www.yahoo.com) and uses the call `int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);` to populate the addrinfo structure variable with the required fields and then prints the IP address of the host. Note that you may have to use the function `inet_ntop()` also to print the IP address in the string format on the display.

```
struct addrinfo {
int              ai_flags;
int              ai_family;
int              ai_socktype;
int              ai_protocol;
socklen_t        ai_addrlen;
struct sockaddr *ai_addr;
char            *ai_canonname;
struct addrinfo *ai_next;
};
```

2. Write a Client program and an iterative Server program both of which use the TCP protocol, to implement the following scenario of messages exchanges: The client sends a mesage "Hello Server" to the server. The server responds to this message by echoing the message received from the client and suffixing its own "Hello Client" message to the client message. The server then sends appropriately formatted, time of the day also to the client; which the client outputs on the standard output, along with the echoed message it received from the server.

3. Write a simple stream server which merely sends out a `"Hello, World"` on the stream connection to the client. How would you test this server even before you have written the client ? Use port number 3993 - hardcode it in the program itself. Write the client code to work with servername as the first argument and the port number and the service helloservice as the second argument. Implement the server as an iterative server and make provision in it to ensure that it cleans up the child processes (that it might create) decently. Observe what happens if you run the client before you run the server ?

4. Modify the Client and the Server programs in problem 3 to now use UDP instead of TCP. Is this a concurrent server or an iterative server, now ?

5. Write a TCP echo server and a client. Whenever the client sends a message to the server and the server must echo back the same message to the client and wait further for any message to come from any client next. The interaction terminates when Client terminates the connection.
   While testing your code, first start the server and then before starting the client, use `netstat a` command with appropriate filters, to display ONLY TCP state of all the sockets created by your programs. Use appropriate command to find the state of the client, server processes. What are the states of the processes ?
   Next, start the client and when the connection is established, before giving any inputs on the client side, again display the TCP state of the server and client sockets. USe appropriate command to find the state of the client, server processes. What are the states of the processes ?
   Next, type in some input on the client standard input to be communicted to the server. Again, display the TCP state of the server and client sockets. USe appropriate command to find the state of the client, server processes. What are the states of the processes ?
   Next, terminate the client and immediately give the netstat -a command on the server to catch the TCP state of the server socket at that instant. Give the same command after some time and display again the state of the server socket. What are the states of the processes on both client and the server ? Is there any processes in zombie state ? Which process is in zombie state and reason why is it in such state.

6. Create a simple instant messaging system in which the client and server send a message of maximum 80 characters one at a time in strict alteration as follows:

   - Server is waiting for connection

   - Client is connected to Server

   - Client sends message

   - Server receives message

   - Server sends message

   - Client receives message Above sequence from steps iterates till the client sends an explicit Goodbye message, whereupon the server terminates the connection.

7. Write Client socket program that implements the *arp* command. That is, when executed with the IP address of any client in the network as argument, it uses the *ioctl* system call to obtain the corresponding hardware address and displays the hardware address. Lookup man pages of *ioctl* to find out how to use the call.

8. Write an ftp client that sends a request for downloading a file to your own ftp server. The server responds with the contents of the file that is stored on the client local directory. The server checks the size of the file before responding and if the file size is greater than 1024 bytes, instead of sending the file, the responds with the message "File size greater than 1 KB" and waits further for another request from the client. The connection is terminated only when client terminates the same.

9. Now, instead of using ftp, you would use develop a Web server that also responds with the file contents upon being contacted by the client. Your simple Web server would be capable of processing only one request. Specifically, your Web server will (i) create a connection socket when contacted by a client (a web browser); (ii) receive the HTTP request from this connection (read the http command sequences, discussed in class); (iii) parse the request to determine the specific file being requested; (iv) get the requested file from the servers file system, restricted to local directory; (v) create an HTTP response message consisting of the requested file preceded by header lines; and (vi) send the response over the TCP connection to the requesting browser. If a browser requests a file that is not present in your server, your server should return a 404 Not Found error message. You may use the skeleton code for your server available on textbook's companion Web site. Your job is to complete the code, run your server, and then test your server by sending requests from browsers. Use the port no other than 80 if you run your server on a host that already has a Web server running on it.

10. In this programming assignment, you will write a client ping program. Your client will send a simple ping message to a server, receive a corresponding pong message back from the server, and determine

the delay between when the client sent the ping message and received the pong message. This delay is called the Round Trip Time (RTT). The functionality provided by the client and server is similar to the functionality provided by standard ping program available in modern operating systems. However, standard ping programs use the Internet Control Message Protocol (ICMP), whereas your client would be a simple UDP-based ping program.
Your ping program is to send 10 ping messages to the target server over UDP. For each message, your client is to determine and print the RTT when the corresponding pong message is returned. Because UDP is an unreliable protocol, a packet sent by the client or server may be lost. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should have the client wait up to one second for a reply from the server; if no reply is received, the client should assume that the packet was lost and print a message accordingly. For this assignment, the complete code for the server is available in the companion Web site of the textbook. Your job is to write the client code, which will be very similar to the server code. It is recommended that you first study carefully the server code. You can then write your client code, liberally cutting and pasting lines from the server code.

11. In this problem, you would be implementing an HTTP client. Your client should take command line arguments specifying a server name or IP address, the port on which to contact the server, the method you use, and the path of the requested object on the server. As discussed in the classes on http, HTTP uses the GET and the PUT methods. Hence, your socket program must implement the GET and PUT methods of HTTP as per the format specified below:

    - GET : The format of the command line is *myclient host port_number GET filename*
    - The basic client action should proceed as follows:
        a. Connect to the server via a connection-orieted socket.
        b. Submit a valid HTTP/1.0 GET request for the supplied URL.
        c. GET /index.html HTTP/1.x (end with extra CR/LF)
        d. Read (from the socket) the server‘s response and display it as program output.
        e. Once you have this part of the client working, you should demonstrate it with the following test case viz. Use it to get a file of your choosing from a “real ”web server on the internet. For example, myclient www.cnn.com 80 GET index.html