CMPT 471: ASSIGNMENT 3

YOU MAY WORK IN GROUPS OF 1-3 STUDENTS FOR THIS ASSIGNMENT

PROBLEM 1: 75 points

Submit the code for your client and your server to the course management system. You may use C or C++ to develop the server and clients, but you MUST use the C socket library (not a C++ socket library). Your code must run in the CSIL LINUX environment.

- 40 points for a dual stack file transfer server (sequential operation)
- 35 points for the points for a dual stack TCP echo client (sequential operation)
- 60 points given for correctly functioning server (32 points) and client (28 points).
    - It is much better to submit a consistently working code that satisfies some requirements than a partially working (unstable) code that sometimes satisfies more requirements. If your code crashes or hangs during one of the tests you will receive a substantial deduction from your grade (~5-10 points).
- 15 points given for explanation and/or detailed comments explaining how the client (7 points) or server (8 points) functions
- If you submit a sequential server instead of an instance server you can receive a deduction of 15 points
- If you submit an IPv4 only client and server you can receive a deduction of 25 points.

Write a dual stack TCP file transfer client and a dual stack TCP file transfer server. You may begin with the sample code provided if you wish. The sample code is an echo client and server (not optimally implemented). The sample server is an instance server that created a process that is a copy of the original server to service each request. Specifications of each client and the server are given below.

For smoother and easier development you may wish to write a client and sequential server for IPv4 only first, then add the support for IPv6, then convert to an instance server.

GENERAL SPECIFICATIONS

1. Your code must be written in C or C++ and must run in the CSIL LINUX environment
2. You may assume that files to be transferred are in the same folder as the TCP file transfer server.
3. IPv4 requests and IPv6 requests can arrive at the server in any order, you MAY NOT assume that they alternate.
4. The instance server will create a new copy of itself to process each file transfer. You may create the copy as a new process OR as a new thread, the sample code produces a new process.
5. If a client request is made using the IPv6 address of the server then the request and the file will be transferred using IPv6.
6. If a client request is made using the IPv4 address of the server then the request and the file will be transferred using IPv4.
7. Your file transfer server should be able to respond to both IPv4 and IPv6 requests. It should respond to requests from IPv4 and IPv6 clients sequentially (in any order) or, for the instance server, concurrently (at the same time) without restarting the server.

8. Your TCP client and server should use an internal buffer with a size that is specified as a command line argument. The size of the buffer will be the maximum size of the block of data that can be carried in a single TCP segment. The size of the buffer must be the same in the clients and the server. This buffer will be referred to as the send/receive buffer. This is NOT the sliding window buffer.
9. Your client should take as command line arguments the server IP address and port number, the client IP address, the filename of the requested file and the size of the send/receive buffer in that order as command line arguments,
10. Your servers should take as command line arguments the server IPv4 port number, the sever IPv6 port number and the size of send/receive buffer. Arguments should be in the order mentioned in the previous sentence.

TCP file transfer client specifications:

- Constructs a message to send to the file transfer server that will indicate the file that should be transferred.
- If the command line arguments for the IP address of the server is an IPv4 address creates an IPv4 socket, otherwise creates an IPv6 socket.
- Sets the MTU of the new socket (from buffer size)
- Connects to the server
- Sends the constructed message to the file transfer server.
- If the packet contains the error message "COULD NOT OPEN REQUESTED FILE"
    - Clean up and terminate the client
- Otherwise the packet will contain the message "FILE SIZE IS xxxxx bytes" where xxxx is the size of the file in bytes.
    - Extract and save the number of bytes
    - Open a file with the same name as the file requested on the client host (in the same folder as the client is running)
    - If the file cannot be opened print the error message "COULD NOT OPEN OUTPUT FILE", clean up and terminate the client.
- Until all data in the file has been received.
    - A packet of data is received
    - The data is removed from the packet and written to the file on the client.
    - The socket waits to read the next packet that arrives.
- Duplicate data should not be added to the file
- After execution of the client this file on the client host should be identical to the file on the server host.

File transfer server specifications:

1. The server should create an IPv4 TCP socket and an IPv6 TCP socket.
2. The server should use the values in the command line arguments to bind the IPv4 and IPv6 sockets to the correct port and IP address.
3. The server should convert the IPv4 and IPv6 sockets to listening sockets.
4. The instance server should spawn a copy of itself to manage each TCP connection request.
5. Each copy should be a separate process or a separate thread.
6. Each copy of the TCP echo server should
    a. Accept a connection request.
        i. If the connection request is for the IPv4 port connect to the IPv4 socket

        ii. If the connection request is for the IPv6 port connect to the IPv6 socket

b. Close the listening sockets
c. Receive the request for the file to transfer.
d. Extract the filename from the request and attempt to open the file
e. Should send a message to the client
    1. If the file could not be found or not be opened the message should be
       "COULD NOT OPEN REQUESTED FILE"
    2. If the file is open and ready to send the message should be
       "FILE SIZE IS xxxxx bytes".
f. If the file cannot be found or cannot be opened then clean up and terminate the copy of the server.
g. Send the contents of the file one segment at a time. Wait approximately 1 second between sending successive segments.
h. Close the file
i. Terminate the copy of the server

Default values:

- For both the client and the servers the IPv4 port number should default to 33455 if no port number is given. And the IPv6 port number should default to 33446
- For both the client and the server the IP address should default to the localhost address
- For both the client and the servers the send receive buffer size should default to 1440 for IPv4 and 1280 for IPv6 if no size is given.
- The default filename will be fileToTransfer
- If no arguments are given your clients and server should print a message indicating the command line inputs/options for running the program.

Test your clients and server to assure that they function correctly both when running on the same machine (loopback) and when running on different machines. Use the dual stack network (lab1-06 to lab1-010) for testing.

PROBLEM 2: (15 points) Problem 9.5 from Comer
Consider an Ethernet that has one conventional host, H, and 12 routers connected to it. Find a single (slightly illegal) frame carrying an IP packet that, when sent by host H, causes H to receive exactly 24 packets.

PROBLEM 3: (10 points)
Consider computation of a TCP checksum. Assume that although the checksum field in the segment has not been set to zero, the result of computing the checksum is zero. What can you conclude?