Programming Assignment #4

CSCE 3530 – Introduction to Computer Networks

Fall 2020

100 Points
Due: 11/10/2020, 11:55 PM

Instructions: Compile the C programs in one of the CSE (cse01 – cse06) servers and make sure it's working. Comment your code. Create a Makefile to compile the source code and to clean the binaries. Create a readme file that describes how to compile, execute, and test the code. Create an assignment folder with folder name as *euid_PA4* (example: xyz0202_PA4) and add all the source files, Makefile, and the readme file to the folder. Please create a zip archive of your assignment folder and upload the zip file to Canvas. **Not following the above instructions could result up to 50% deduction from your assignment score. Late submissions are not allowed.**

Objective:

Simulate TCP 3-way handshake and closing a TCP connection in the application layer using a client-server architecture.

Requirements:

1. Create a C based client-server architecture using TCP sockets
2. The server should run on cse03.cse.unt.edu machine and the client should run on cse02.cse.unt.edu machine
3. The client should be able to communicate with the server using TCP segments
4. The client should be able to demonstrate a 3-way TCP handshake with the server
5. The client should be able to demonstrate closing a TCP connection.

Procedure:

1. Create a C-based server that can runs on cse03.cse.unt.edu and accepts single client's request on using sockets
2. Create a C-based client that runs on cse02.cse.unt.edu and connects to the server
3. On the client side, create a TCP segment as shown in Figure 1 with C structure that has the following fields
   a. 16-bit source and destination port [Type: `unsigned short int`]
   b. 32-bit sequence number [Type: `unsigned int`]
   c. 32-bit acknowledgement number [Type: `unsigned int`]

   d. 4-bit data offset or header length (in multiples of 32-bit) [Type: `unsigned short int` for combined fields of d, e, and f]
   e. 6-bit reserved section
   f. 6-bit flags
   g. 16-bit receive window for flow control [Type: `unsigned short int`, set to zero]
   h. 16-bit checksum [Type: `unsigned short int`, computed after the header and data is populated]
   i. 16-bit urgent data pointer [Type: `unsigned short int,` set to zero]
   j. 32-bit Options [Type: `unsigned int`, set to zero]
   k. 1024-byte data [Type: `char,` set to nulls]
4. The variable length payload or the data field does not have any data for TCP 3-way handshake and closing a TCP connection demonstration. The data field is 1024 bytes for data which is set to nulls as we are going to use the structure for demonstrating opening and closing of the connection
5. Find the source port (using an OS function call), destination port, and header length and populate the corresponding fields of all the client and server TCP segments created below
6. Demonstrate TCP handshaking by following the below steps. Refer to Figure 2.
   a. Create a connection request TCP segment as follows:
      i. Assign a random value for the initial client sequence number and assign zero for the client acknowledgement number
      ii. Set the SYN bit to 1
      iii. Compute the 16-bit checksum for the entire TCP segment and populate the checksum field
      iv. Print on the console and write to client.out file the values of source port, destination port, sequence number, acknowledgement number, header length, flags (ACK, SYN, FIN), and the checksum of the segment that is created
      v. Send the created connection request segment to the server.
   b. The server should receive the TCP segment from the client and compute the checksum of the received TCP segment. If the segment does not have any errors, then it will check the flags. If the client is requesting a connection, then the server will respond to the request by creating a connection granted TCP segment. Create a connection granted response as follows:
      i. Assign a random value for initial server sequence number and assign the received sequence number from the client + 1 as the server acknowledgement number
      ii. Set both the SYN bit and ACK bit to 1

iii. Compute the 16-bit checksum of the entire TCP segment and populate the checksum field
iv. Print on the console and write to server.out file the values of source port, destination port, sequence number, acknowledgement number, header length, flags (ACK, SYN, FIN), and the checksum of the segment that is created
v. Send the created connection granted (SYNACK) segment to the client.

c. The client should receive the TCP segment from the server and compute the checksum of the received TCP segment. If the segment does not have any errors, then it will check the flags. If the server has granted connection, then the client will respond back with an acknowledgement in the TCP segment. Create the TCP segment as follows:
   i. Assign the received acknowledgment number from the server as the client sequence number and assign the received sequence number from the server + 1 as the client acknowledgement number
   ii. Set the ACK bit to 1
   iii. Compute the 16-bit checksum of the entire TCP segment and populate the checksum field
   iv. Print on the console and write to client.out file the values of source port, destination port, sequence number, acknowledgement number, header length, flags (ACK, SYN, FIN), and the checksum of the segment that is created
   v. Send the created client ACK segment to the server.
d. The server should receive the TCP segment from the client and compute the checksum of the received TCP segment. If the segment does not have any errors, then it will check the flags. If the client has acknowledged for the SYNACK segment, then print on the console and write to server.out file that the connection has been established.

7. Demonstrate closing of a TCP connection by following the below steps. Refer to Figure 3.
   a. Create a close request TCP segment as follows
      i. Assign 1024 as client sequence number and 512 as acknowledgement number
      ii. Set the FIN bit to 1
      iii. Compute the 16-bit checksum of the entire TCP segment and populate the checksum field
      iv. Print on the console and write to client.out file the values of source port, destination port, sequence number, acknowledgement number, header length, flags (ACK, SYN, FIN), and the checksum of the segment that is created

v. Send the created close request segment to the server

b. The server should receive the TCP segment from the client and compute the checksum of the received TCP segment. If the segment does not have any errors, then it will check the flags. If the client is requesting to close the connection, the server will respond back with an acknowledgment TCP segment. Create an acknowledgement TCP segment as follows
   i. Assign 512 as server sequence number and acknowledgement number equal to client sequence number + 1
   ii. Set the ACK bit to 1
   iii. Compute the 16-bit checksum of the entire TCP segment and populate the checksum field
   iv. Print on the console and write to server.out file the values of source port, destination port, sequence number, acknowledgement number, header length, flags (ACK, SYN, FIN), and the checksum of the segment that is created
   v. Send the created acknowledgment segment to the client.
c. The client should receive the TCP segment from the server and compute the checksum of the received TCP segment. If the segment does not have any errors, then it will check the flags. If the server has sent an acknowledgement for the close request, then it will wait for the next segment. The server again sends another close acknowledgement TCP segment with FIN bit set. Create the TCP segment as follows
   i. Assign again 512 as server sequence number and acknowledgement number equal to client sequence number + 1
   ii. Set the FIN bit to 1
   iii. Compute the 16-bit checksum of the entire TCP segment and populate the checksum field
   iv. Print on the console and write to server.out file the values of source port, destination port, sequence number, acknowledgement number, header length, flags (ACK, SYN, FIN), and the checksum of the segment that is created
   v. Send the created close acknowledgment segment to the client
d. The client should receive the TCP segment from the server and compute the checksum of the received TCP segment. If the segment does not have any errors, then it will check the flags. If the client has received a segment with the FIN bit set, then the client responds back with an acknowledgement TCP segment. Create an acknowledgement TCP segment as follows
   i. Assign a sequence number as client sequence number + 1 and acknowledgement number equal to server sequence number + 1
   ii. Set the ACK bit to 1

iii. Compute the 16-bit checksum of the entire TCP segment and populate the checksum field
iv. Print on the console and write to client.out file the values of source port, destination port, sequence number, acknowledgement number, header length, flags (ACK, SYN, FIN), and the checksum of the segment that is created
v. Send the created acknowledgement segment to the server and exit the client program after a sleep of 2 seconds.

e. The server should receive the TCP segment from the client and compute the checksum of the received TCP segment. If the segment does not have any errors, then it will check the flags. If the client has acknowledged for the final FIN segment, then print on the console and write to server.out file that the connection is closed and exit the server program.

8. Make sure the server port number is sourced as an argument to the client as shown below

```
./tclient <port_number>
```

9. Run the server as

```
./tserver <port_number>
```

10. Assume client and server has no timeout constraints
11. An example code (cksum.c) to compute the checksum is given on Canvas.

Deliverables:

1. Commented server and client C code
2. A Makefile to compile (*make*) all the source code and to clean (*make clean*) all the executables
3. A readme file that describes how to compile, execute, and test the code.

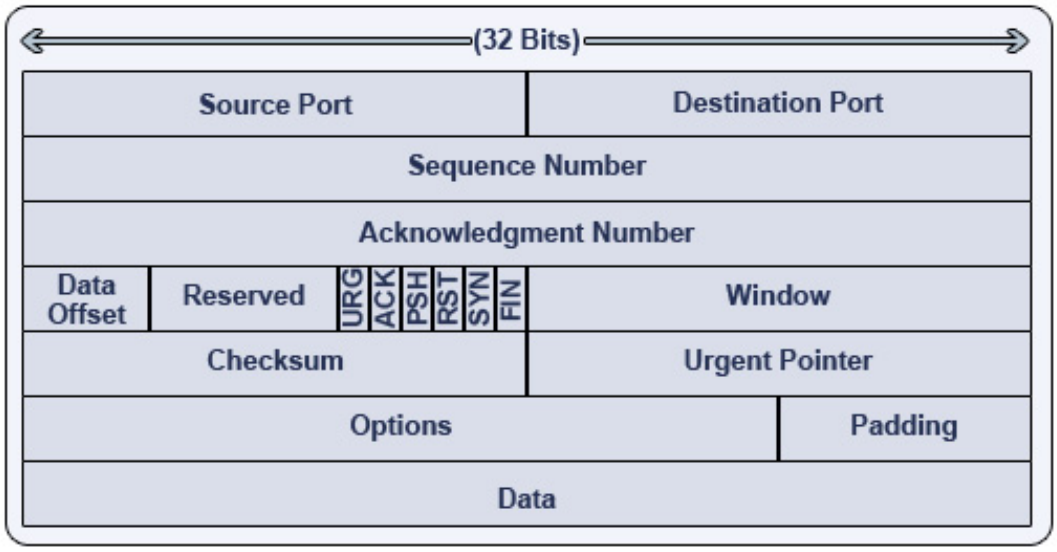

Figure 1. TCP Segment Structure
[Source: http://cse1.net/recaps/11-tcpip.html]

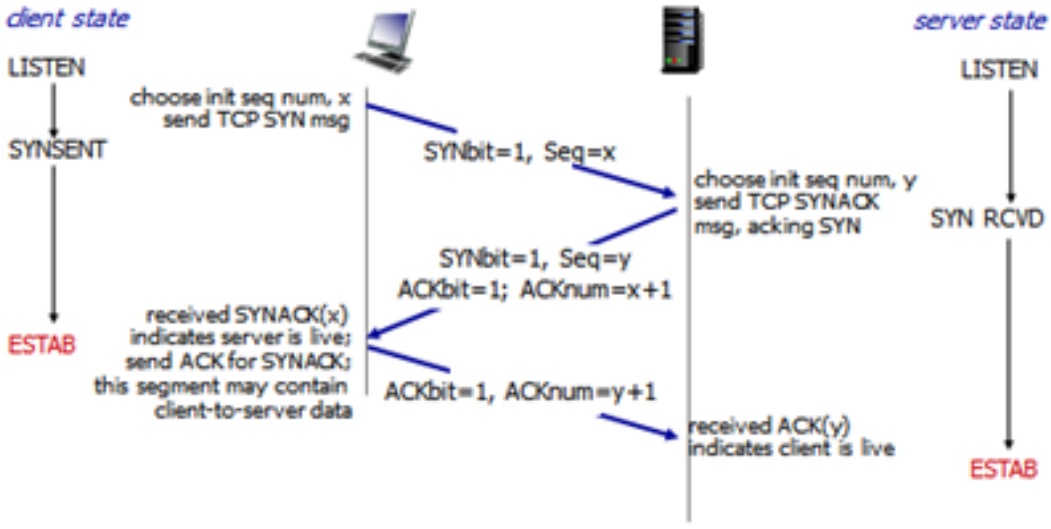

Figure 2. TCP 3-Way Handshake
[Source: Lecture notes]

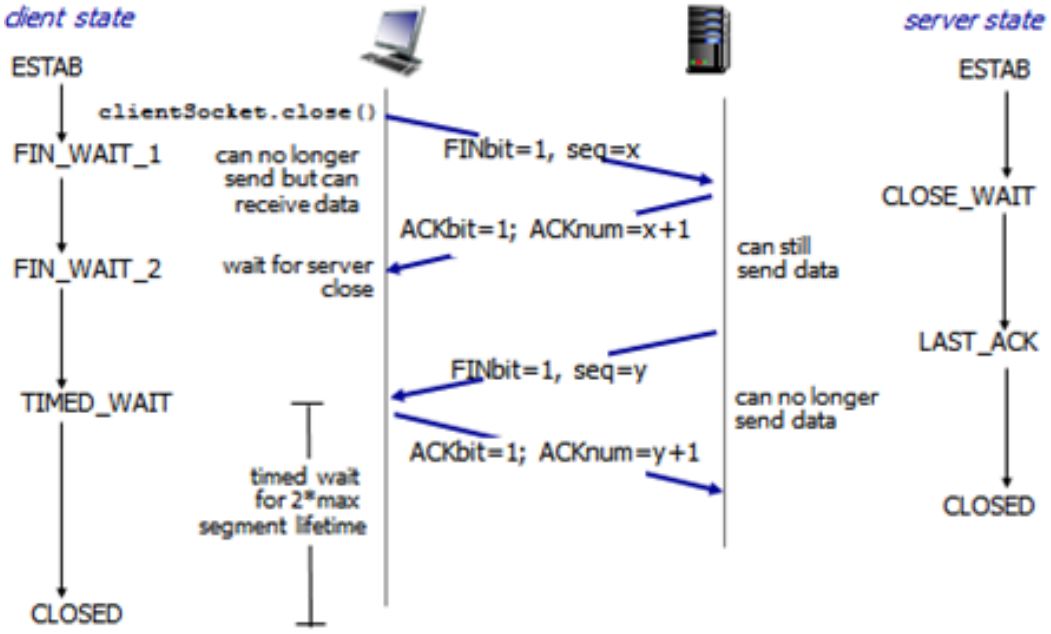

Figure 3. Closing a TCP connection

[Source: Lecture notes]