**Module 3.1 – TCP Sockets.**

**Module 3 required reading material:**

- [1] Brian "Beej Jorgensen" Hall, "Beej's Guide to Network Programming, v3.1.11". April 2023. https://beej.us/guide/bgnet/html/split/
    - Read chapter 6.

- [2] Lewis Van Winkle, "Hands-On Network Programming with C". Packt Publishing. May 2019. ISBN: 9781789349863. https://learning.oreilly.com/library/view/hands-on-network-programming/9781789349863/
    - Read chapter 3.

- [3] Jon Erickson, "Hacking the Art of Exploitation 2nd ed". No Starch Press.  February 2008. ISBN: 978-1593271442. https://learning.oreilly.com/library/view/hacking-the-art/9781593271442/
    - Read chapter 0x04, sections 0x425 – 0x427.

- [4] W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, "The Sockets Networking API: UNIX® Network Programming Volume 1, Third Edition". Addison Wesley. November 2003. ISBN: 0-13-141155-1. https://learning.oreilly.com/library/view/the-sockets-networking/0131411551/

    - This book provides a more in-depth/technical explanation for the topics covered in this module.
    - Read Chapter 4 – Elementary TCP sockets. Sections 4.1 – 4.6, 4.9 - 4.10
    - Read Section 2.6 for an explanation of the Three-way handshake and connection termination.

In this module we'll cover TCP sockets in-depth. We'll write TCP client and server applications, cover blocking I/O, and multiplexing I/O.

Now that we've covered socket functions and relevant structures, we'll take a different approach for the rest of the modules. We're mainly going to be writing network applications and we'll write our notes in the code itself. If there is any topic that requires a more in-depth explanation, we'll add it to the word document and PDF as needed. You are expected to read the textbooks and resources and work through the examples in the book.

This module mainly focuses on explaining the code repository layout, how to build our binaries, and how to use the header file(s) where we'll be implementing our code.

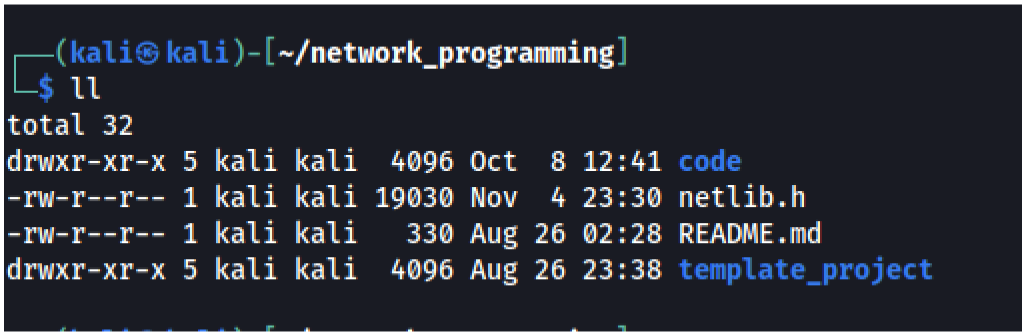
As you'll inevitably notice, once you've written a few network applications using the socket API, the code gets repetitive since we use the same set of functions. For most of the code we write, we'll use a few software engineering practices such as using header files to organize and group our functions and we'll include the header file in our main application. We'll also be using the Linux program "make" to automate the building process.

**Navigating the repo**

You're encouraged to work through the problems and compare notes with the implementation provided, after all the only way to learn programming is programming.

It is assumed that you have set up your Kali VM, installed the required tools and binaries. If not, go back to Module 1.2 and perform the steps necessary to set up your development environment.

Folder Layout



```
┌──(kali㉿kali)-[~/network_programming]
└─$ ll
total 32
drwxr-xr-x 5 kali kali  4096 Oct  8 12:41 code
-rw-r--r-- 1 kali kali 19030 Nov  4 23:30 netlib.h
-rw-r--r-- 1 kali kali   330 Aug 26 02:28 README.md
drwxr-xr-x 5 kali kali  4096 Aug 26 23:38 template_project
```

- template_project
  - This directory serves as a template directory that you can use to build your applications. The code directory was based off template_project.
- Netlib.h
  - This is the header file containing all the functionality related to socket programming. I grouped the code here to avoid duplication of code and focus on the specific topics.
- README.md
  - This is the readme file for the repo, do read it.
- Code
  - This is the main directory for all the applications written as I went through the material.
  - It contains a Make file that you should read and use to build your own Make file to build your applications. You'll need to research how to use Make but we only need the basics for our purposes.
  - It contains a bin directory where your ELF binaries are placed once compiled and linked.
  - It contains a directory for each topic/application i.e.: http, https, tcp, udp, etc.

You don't have to set up the source directory the same way I did, nor do you have to use header files to group your code, this is how I chose to do it for simplicity. It is assumed you have the required technical knowledge to build, debug, and write small to medium software. "Hands-On Network Programming with C" shows how to build and link the applications in each chapter, I suggest doing the same if you want to keep it simple.

I used VS Code but you're free to use any IDE of your choosing.

**Writing a simple TCP server**

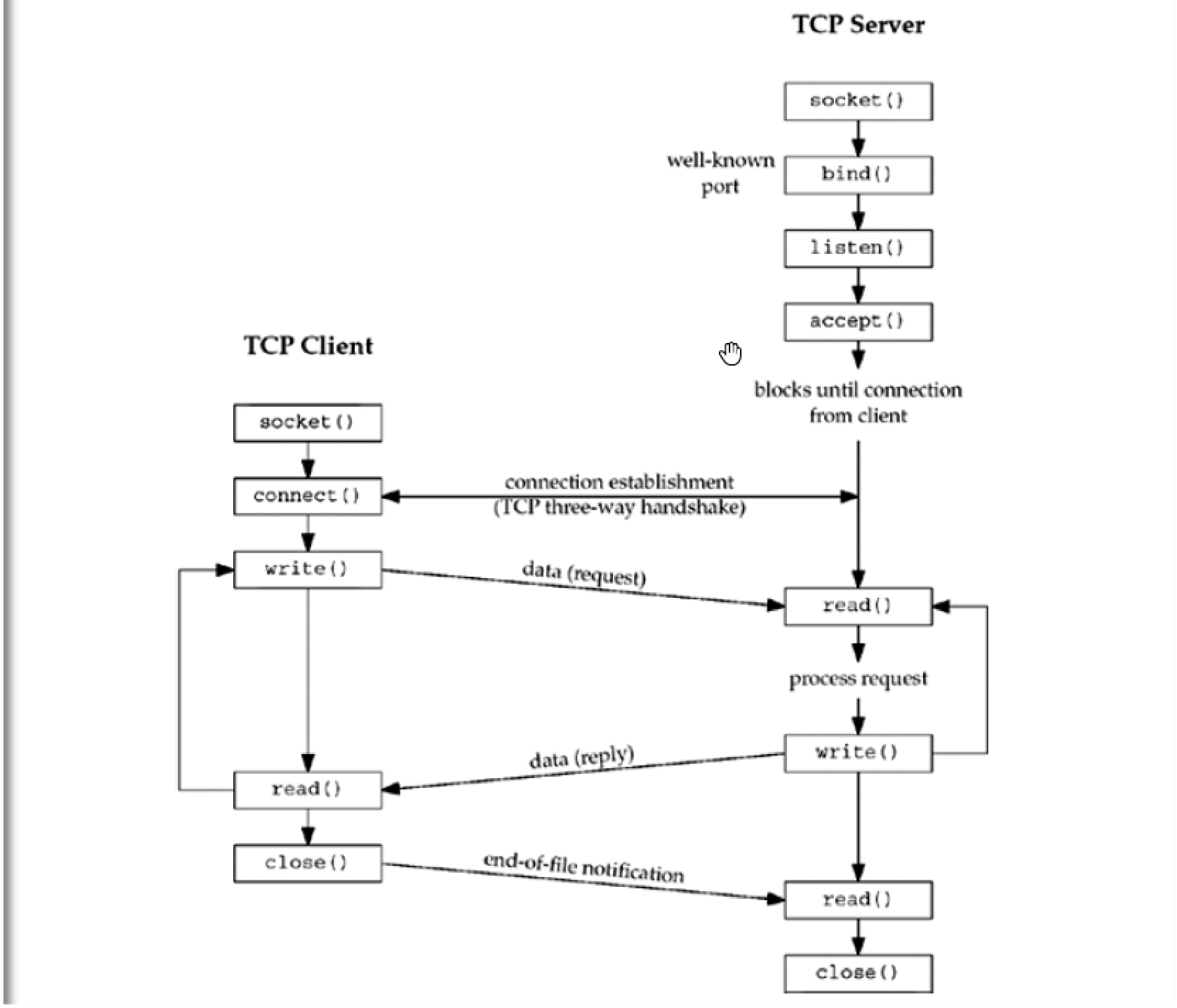
Our first program is server.c  found in the Beej's Guide to Network Programming book, chapter 6. You'll get the most out of this course by re-writing each program and fiddling with the code to get a better understanding of how each of the functions work.

Once you're done writing your own version, you can browse to: "network_programming/code/src/tcp_server and open server.c" and read through the comments if you need help. Or you can read through the code and implement your version after.

From "Hands-On Network Programming with C", read the section on the time_server.c program in chapter 2.

I found it useful to have this diagram opened whenever I wrote a networking application, recommend doing the same.

**Figure 4.1. Socket functions for elementary TCP client/server.**



**TCP Connection Establishment and Termination**

Recall, that TCP is a connection-oriented protocol with reliability, among other things. UDP on the other hand is a connectionless protocol and not as "reliable" with no guarantee of packet delivery. Let's cover two aspects of how TCP accomplishes its connection-oriented capabilities. Read Section 2.6 for a more thorough explanation of the Three-way handshake and connection termination.

<u>TCP Connection Establishment and the "Three-way handshake"</u>

One of the common network programming paradigms is the Client-Server paradigm , which is the main focus of this course.  In this model and using the TCP protocol, the client initiates a TCP connection with the server through a process called the "Three-way handshake" before actual data can be sent and/or received between the two systems.

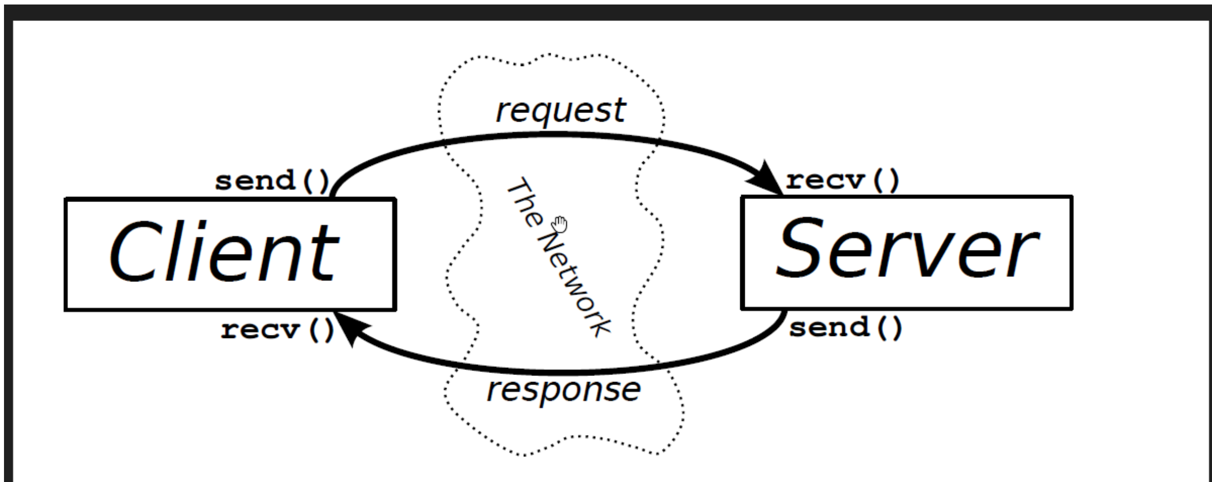


Figure 6.1: Client-Server Interaction.

Let's go through The three-way handshake and how each of the socket functions map to each step of the process.

To aid in our understanding of the connect, accept, and close functions and to help us debug TCP applications using the netstat program, we must understand how TCP connections are established and terminated, and TCP's state transition diagram.

The following scenario occurs when a TCP connection is established:

1. The server must be prepared to accept an incoming connection. This is normally done by calling socket, bind, and listen and is called a passive open.

2. The client issues an active open by calling connect. This causes the client TCP to send a ''synchronize'' (SYN) segment, which tells the server the client's initial  sequence number for the data that the client will send on the connection. Normally, there is no data sent with the SYN; it

just contains an IP header, a TCP header, and possible TCP options (which we will talk about shortly).

3. The server must acknowledge (ACK) the client's SYN and the server must also send its own SYN containing the initial sequence number for the data that the server will send on the connection. The server sends its SYN and the ACK of the client's SYN in a single segment.

4. The client must acknowledge the server's SYN.

The minimum number of packets required for this exchange is three; hence, this is called TCP's *three-way handshake*. We show the three segments in Figure 2.2.

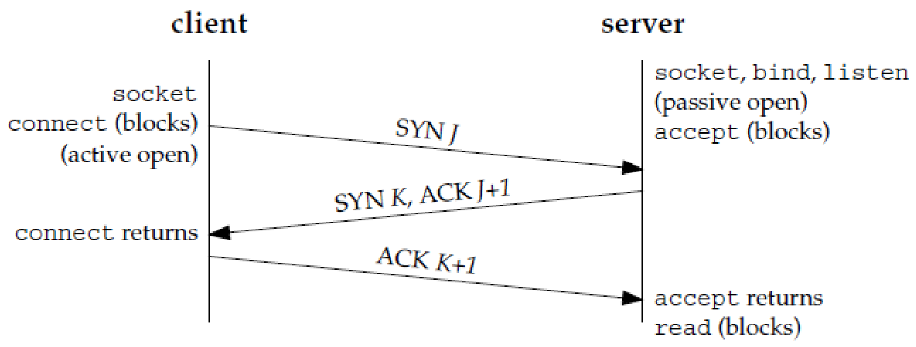


**Figure 2.2** TCP three-way handshake.

TCP Connection Termination

On the reverse side, we have the connection tear down process. As part of its reliability, TCP needs to make sure the connection is closed in an orderly manner and that there is no data transmission pending when the connection is terminated under normal circumstances.

While it takes three segments to establish a connection, it takes four to terminate a connection.

1. One application calls close first, and we say that this end performs the active close. This end's TCP sends a FIN segment, which means it is finished sending data.

2. The other end that receives the FIN performs the passive close. The received FIN is acknowledged by TCP. The receipt of the FIN is also passed to the application as an end-of-file (after any data that may have already been queued for the application to receive), since the receipt of the FIN means the application will not receive any additional data on the connection.

3. Sometime later, the application that received the end-of-file will close its socket. This causes its TCP to send a FIN.

4. The TCP on the system that receives this final FIN (the end that did the active close) acknowledges the FIN.

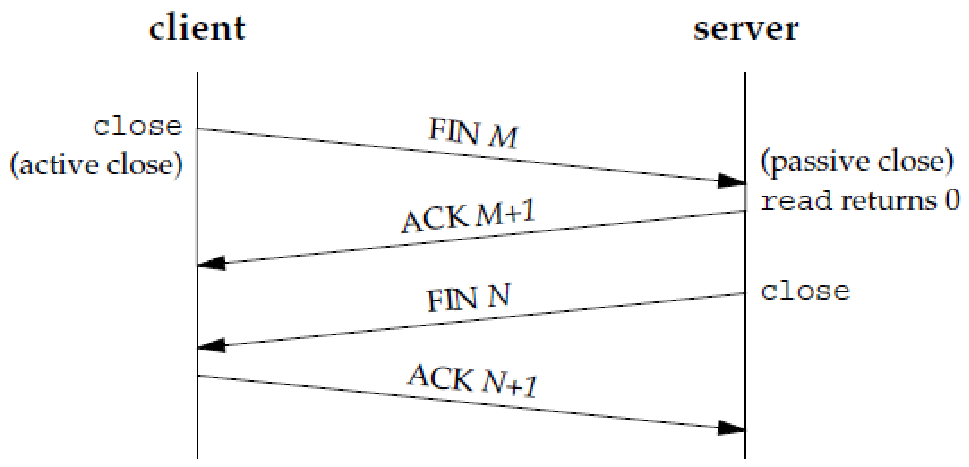


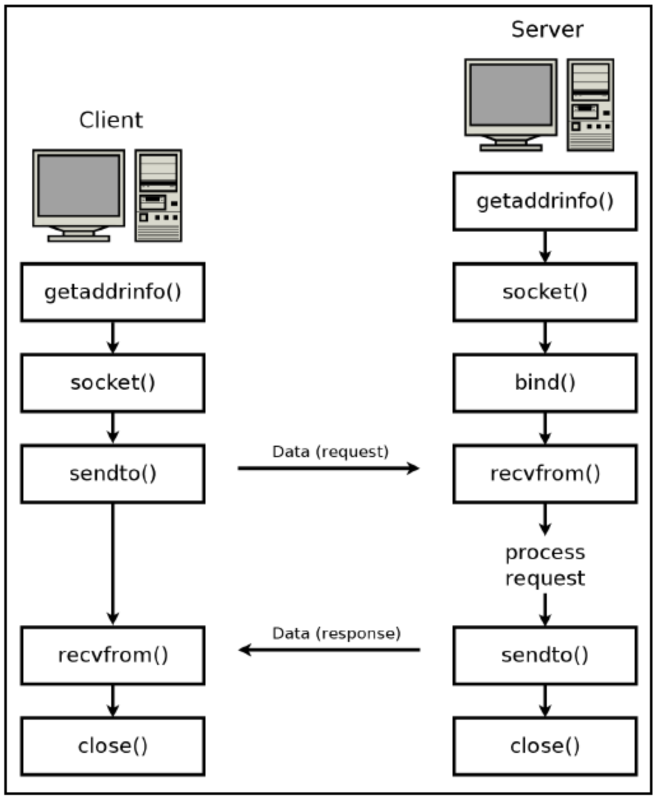
**Figure 2.3** Packets exchanged when a TCP connection is closed.

*UDP has no concept of a connection setup or teardown, and you will not see either of the handshake processes take place in a packet monitor when monitoring UDP connections. UDP will be covered in the next module.

**Assignments/Homework**

There are no required assignments, but you're expected to work through the material and write code, that was at least my approach.

I worked through each of the programs found in both textbooks, researched the different functions, and tweaked things until I felt I had a good understanding of the material. As I progressed through each chapter, I relied more on the following graphics to write my applications and less so on the code provided by the book. I recommend you do the same.



The other factor that helped me learned the concepts was taking the code from “Hands-On Network Programming with C" and writing it for Linux only instead of copying and pasting the code.

**Assignments**

Work through the following and add comments to each of the socket functions and structures with your understanding of what they’re doing.

1. “Beej's Guide to Network Programming – chapter 6
    a. Re-write the server.c program
2. “Hands-On Network Programming with C" – chapter 2
    a. Re-write the time_server.c program to run on Linux only.
3. Run Wireshark as you build your TCP client and observe how each function call maps to the three-way handshake and connection termination process.
    a. Call socket() and monitor for new inodes
    b. Call connect() and monitor for a SYN packet in Wireshark, and so on.