**Module 2.2 – Introduction to the Socket API part 1.**

**Goals:**

* Cover the Socket API.
* Socket API structures.
* Write a simple socket application.

**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/>
  + Chapter 3.
* [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 2.
* [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, section 0x420 -Sockets.
* [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 3 – Sockets Introduction. <https://learning.oreilly.com/library/view/the-sockets-networking/0131411551/ch03.html>

In this module we’ll learn about the socket API and will work on our first socket application. Our goal is to learn the “anatomy” of a socket program, and the general flow of a socket server or client program. Armed with this knowledge, we'll be able to explore each of the individual functions in greater detail. A lot of this information has already been covered in the referenced material (we will not be reinventing the wheel), I recommend you read those resources. We’ll do our best to summarize the relevant information, but this module is by no means a substitute for reading the textbooks.

Modules 2 will cover a lot of background information required for socket programming, our focus will be mainly on the technical details and less so on programming. As we progress through the course, our focus will shift to writing more applications and only covering the relevant technical details for new functions and concepts introduced in those modules. It is imperative that we master the concepts in module 2. You will soon learn that all networking applications we write will use the concepts covered here, that is every single application will call almost the same number of APIs and structures.

**Anatomy of a Socket Program**

There are different network programming paradigms, we'll start our journey into socket programming by exploring the Client-Server model and we'll cover the common socket APIs for both TCP and UDP.

"Network programming is usually done using a client-server paradigm. In this paradigm, a server listens for new connections at a published address. The client, knowing the server's address, is the one to establish the connection initially. Once the connection is established, the client and the server can both send and receive data. This can continue until either the client or the server terminates the connection. A traditional client-server model usually implies different behaviors for the client and server. The way web browsing works, for example, is that the server resides at a known address, waiting for connections. A client (web browser) establishes a connection and sends a request that includes which web page or resource it wants to download. The server then checks that it knows what to do with this request and responds appropriately (by sending the web page)." [2, pg. 47]

Server:

* Passively waits for and responds to client connections.

Client:

* Initiates communication.
* Must know the address and port of the server.

**The socket API**

There is a lot of background information to cover before we can start writing networking applications using the Socket API. Even a small application which only job is to retrieve the network interfaces on your system will require knowledge of multiple socket functions and different and complicated C-structures.

* The set of networking system calls (socket API or functions) allow a programmer to access the network functionality provided by a system, in this case a Unix system or any \*system supporting the sockets API.
* These system calls are implemented at the kernel level. That is, the real functionality is implemented in the kernel. Whenever you call one of these functions, the kernel takes over and does all the work.
* Older textbooks and legacy code will use most of the APIs covered here but there have been new functions added to the socket API which make programming a lot easier. We will cover modern network programming practices as much as possible; make sure to keep that fact in mind whenever you read textbooks and legacy code.

Let’s use a simple TCP client (see screenshot below) that connects to a server and sends a message to start learning about the different socket functions.

A screen shot of a computer program

Description automatically generated

You’ll quickly notice that even a simple TCP client with very basic error handling makes use of multiple functions and structures, each with multiple arguments. The client makes use of the following: socket functions, functions to handle network vs host byte order, and socket related structures.

We’ll cover each in depth in the following modules. You can man each of these functions on a Linux system to learn more by running man <function\_name> (man socket) on a terminal.

* socket() creates and initializes a new socket.
* bind() associates a socket with a particular local IP address and port number.
* listen() is used on the server to cause a TCP socket to listen for new connections.
* connect() is used on the client to set the remote address and port. In the case of TCP, it also establishes a connection.
* accept() is used on the server to create a new socket for an incoming TCP connection.
* send() and recv() are used to send and receive data with a socket.
* sendto() and recvfrom() are used to send and receive data from sockets without a bound remote address.
* close() (Berkeley sockets) and closesocket() (Winsock sockets) are used to close a socket. In the case of TCP, this also terminates the connection.
* shutdown() is used to close one side of a TCP connection. It is useful to ensure an orderly connection teardown.
* select() is used to wait for an event on one or more sockets.
* getnameinfo() and getaddrinfo() provide a protocol-independent manner of working with hostnames and addresses.
* setsockopt() is used to change some socket options.
* fcntl() (Berkeley sockets) and ioctlsocket() (Winsock sockets) are also used to get and set some socket options.

Network Byte Order and related functions: These Socket APIs are used to transmit data across networks using different byte orderings. (list is not exhaustive):

* htonl(long value) Host-to-Network Long
  + Converts a 32-bit integer from the host’s byte order to network byte order.
* htons(short value) Host-to-Network Short:
  + Converts a 16-bit integer from the host’s byte order to network byte order.
* ntohl(long value) Network-to-Host Long:
  + Converts a 32-bit integer from network byte order to the host’s byte order.
* ntohs(long value) Network-to-Host Short:
  + Converts a 16-bit integer from network byte order to the host’s byte order.
* Inet\_pton() Presentation to numeric
  + Converts an Internet Protocol (IP) address from its presentation format (string) to its numeric format (binary).

And finally, we have the C-structures used by socket functions (list is not exhaustive):

* struct addrinfo
  + This structure is a more recent invention and is used to prep the socket address structures for subsequent use. It’s also used in host name lookups, and service name lookups: getaddrinfo() and getnameinfo().
* struct sockaddr{ ... }
  + This structure describes a generic socket address and holds socket address information for many types of sockets.
* struct sockaddr\_in{ ... } & struct sockaddr\_in6:
  + Structure describing an Internet socket address. This structure makes it easy to reference elements of the socket address.
* struct sockaddr\_storage
  + Designed to be large enough to hold both IPv4 and IPv6 structures. Used when you don't know in advance if the API call will fill your structure with an IPv4 or IPv6 address.

This module packs a lot of information, I recommend going through the required reading to get a more detailed explanation of the material. We will cover socket functions and structures in more detail in the next modules.