A Beginners Guide to Socket Programming in C

Prerequisites

- A decent C programming experience.
- Basic understanding of networking concepts.

Socket Programming in C

What are Sockets?

Sockets allow communication between two different processes on the same or different machines.

What is socket programming?

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

What is Client-Server Communication?

Client/Server communication involves two components, namely a client and a server. There are usually multiple clients in communication with a single server. The clients send requests to the server and the server responds to the client requests.

Client is sometimes on and initiates requests to the server whenever interested. It needs to know the address of the server.

Server is always on and services requests from many clients. It doesn’t initiate contact with any clients.

How to use sockets?

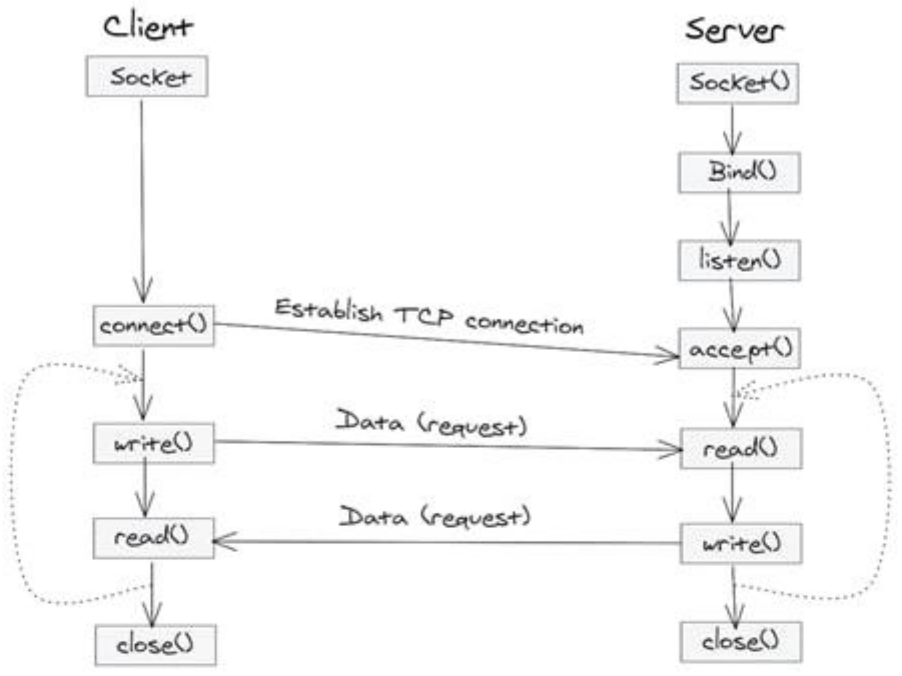
1. Set up a socket.
2. Send and Receive the packets.
3. Close the socket.

Typical Client Program Using TCP

1. Set up a Socket (Prepare to communicate)
    1. Create a socket
    2. Determine server IP address and port number
    3. Initiate the connection to the server
2. Send and receive packets (Exchange data with the server)
    1. Write data (i.e., request) to the socket
    2. Read data (i.e., response) from the socket
    3. Do stuff with the data (e.g., display a Web page)
3. Close the socket.

Typical Server Program Using TCP

1. Set up a Socket (Prepare to communicate)
    1. Create a socket s_listen (i.e., the listening socket)
    2. Associate server's IP address and port no. with the socket
2. Wait to hear from a client
    1. Indicate how many connections can be pending on the socket
    2. Accept an incoming connection from a client, create a new socket s_new for the client.
3. Send and receive packets (Exchange data with the client over the new socket s_new)
    1. Read data (i.e., client request) from the socket
    2. Handle the request
    3. Write data (i.e., server response) to the socket
    4. Close the socket s_new
4. Repeat 2.2-3.4 with the next connection request



Let us see how each step is done.

1. Socket creation

Both client and server need to setup the socket
int socket (domain, type, protocol);
Returns a socket descriptor on success, -1 on failure.

- **domain**
    - AF_INET for Ipv4
    - AF_INET6 for Ipv6
- **type**
    - SOCK_STREAM for TCP
    - SOCK_DGRAM for UDP
- **protocol**
    - 0

Example:
int sockfd = socket (AF_INET, SOCK_STREAM, 0);

2. Bind

Only the server needs to bind.
int bind (int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);

- sockfd is a file descriptor socket() that is returned.
- my_addr
  struck sockaddr_in for IPv4
- struct sockaddr_in {
- short sin_family; // e.g. AF_INET
- unsigned short sin_port; // e.g. htons(3490)
- struct in_addr sin_addr; // see struct in_addr below
- char sin_zero[8]; // zero this if you want to
- };
  struct in_addr
  struct in_addr {
  unsigned long s_addr; // load with inet_aton()
  };
- addrlen is the size of the address structure

Example for bind()

```
struct sockaddr_in my_addr;
int sockfd;
if ((sockfd = socket (AF_INET, SOCK_STREAM, 0) < 0) {
    printf (“Error while creating the socket\n”);
    exit(1);
}
bzero (&my_addr, sizeof(my_addr)); // zero structure out
my_addr.sin_family = AF_INET; // match the socket() call
my_addr.sin_port = htons(5100); // specify port to listen on
my_addr.sin_addr.s_addr = htonl(INADDR_ANY); //allow the server to accept a client connection on any
interface
if((bind(sockfd, (struct sockaddr *) &my_addr, sizeof(saddr)) < 0 {
    printf(“Error in binding\n”);
     exit(1);
}
```

3. Listen

Only the server needs to listen
int listen (int sockfd, int backlog)

- **backlog** specifies the maximum number of pending connections the kernel should queue for the socket. Listen returns 0 if OK, -1 on error

4. Accept

Only the server can accept the incoming client connections.
int accept (int sockfd, struct sockaddr *fromaddr, socklen_t *addrlen)

- **fromaddr** is a pointer to store the client address
- **addrlen** is a pointer to store the returned size of addr. accept() takes the first connection off the queue for sockfd and create a new socket (the return value) for communicating with the client. accept() return a new socket descriptor if OK, -1 on error

5. Connect

The client need not bind, listen or accept. All client needs to do is to just connect to the server.
int connect (int sockfd, struct sockaddr *toaddr, socklen_t addrlen)

- **toaddr** contains the IP address and port number of the serve
- **addrlen** is length of the socket address structure connect() returns 0 if connection is successful and -1 on error

6. Sending and receiving the data

- ssize_t read(int sockfd, void *buffer, size_t len)
    - Read up to n bytes from sockfd into buffer
    - Returns the number of bytes read on success (0 indicates end of file), -1 on error.

Example read (sockfd, buffer, sizeof(buffer));

- ssize_t write(int sockfd, const void *buffer, size_t n)
    - Write up to n bytes from buffer to sockfd
    - Returns the number of bytes written on success, -1 on error.

Example write (sockfd, “Hello”, strlen(“Hello”));

7. Close

Don't forget to close the socket descriptor after all the effort we've put.
int close(int sockfd)