Bamshad's Network Programming Steps

Abstract:

When I was a telecommunication student one of our assignments was about socket programming in Linux. Hopefully, there were a lot of sources to learn it. However, during this period of time I have faced with some questions.

For example, in a case where the server should handle multiple clients I found a source that they used select function in both server and client sides. I could understand that we need to use select() function in the server side (because it should deal with multiple clients) but, why we have the select() in the client's code as well? The client is going to communicate only with one server!

To find the answers of my questions, I made some small steps for each of them. In each step we only face with a small question and therefore it is easy understand it.

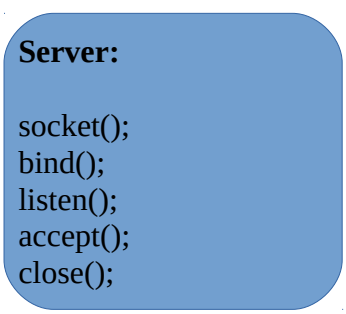
Table of Contents

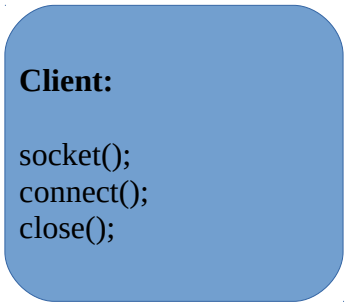
Step 1: Connect

Description

It is the most basic step in socket programming. A server wait for a client and as soon as it got a connection, it accepts it and then both sides close the sockets.

Code structure





Questions And Answers

Question: How in the server side, the code wait for a client in the accept() line?

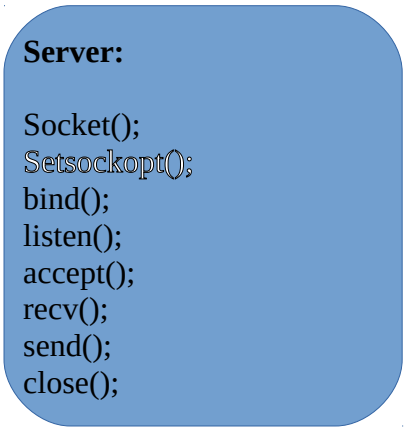
Answer: Accept() is a ***blocking*** function and therefore, it wait as long as a client arrives. It is just like fgets() that wait for the user input.

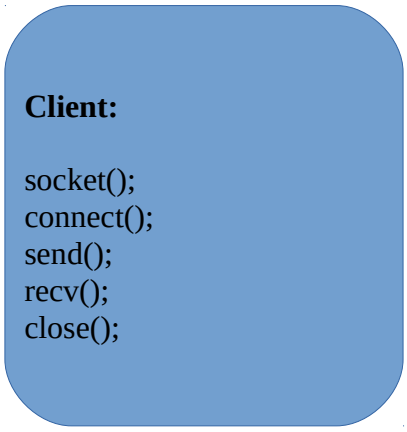
Step 2: Send And Receive

Description

After making connection between server and client, now we want to transfer data. So, in the server side between accept() and close() and in the client side between connect() and close() we are going to use send() and recv().

Code structure

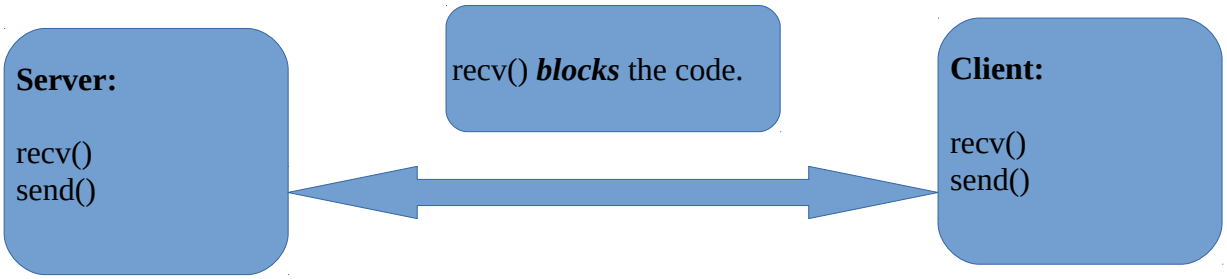




Questions And Answers

Question 1. Does it matter if we change the order of send and receive?

Answer: recv() is a ***blocking*** function. Therefore, it is not possible to first place recv() in both sides.



Question 2. What is the role of setsockopt()?

Answer: “sometimes, you might notice, you try to rerun a server and bind() fails, claiming Address already in use.
What does that mean?

Well, a little bit of a socket that was connected is still hanging around in the kernel, and it's hogging the port. You can either wait for it to clear (a minute or so), or add code to your program allowing it to reuse the port, like this:"

```
int yes=1;

//lose the pesky "Address already in use" error message

if (setsockopt(listener,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
}
```

So the sequence should be
socket()
setsockopt()
bind()

” [1]

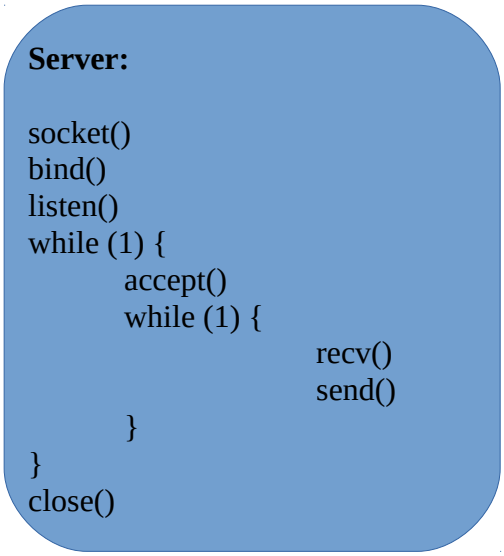
Step 3: Echo (while)

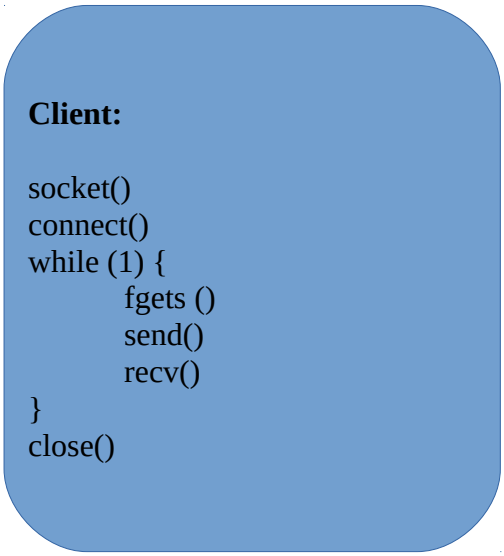
Description

So far we could establish a connection and transfer data between a server and a client. However, in the previous steps everything happened just one time. This step is proposed to learn about continues connection and data transfer. To make it clear, take s look on the following criteria:

1. Server wait for a client.
2. When a client arrived, they connect and send and receive data. The chat continues as long as client is connected. If client disconnect, the server should keep continue and look for another client. So, the server is always running till it is stopped by the user (press control + C).
3. Server just echo whatever it receive from the client.

Code structure





Questions And Answers

Question 1: Why we need the first while(1) loop in the server code?

Answer: As it mentioned in the description of this step, we want server to continue waiting for another client if the current client disconnects. Therefore, the accept() is placed in the while loop too always monitor for a client.

Question 2: In the server side, why recv() and send() are not putted in the same loop where accept() is placed?

Answer: Both accept() and recv() are blocking functions. So, if we put them in a same loop, the server must receive a new client to pass accept() and reach the recv(). Therefore, we made the second loop to continue the chat as long as the peer client is alive.

Question 3: Why do we need a while(1) loop in the client side?

Answer: Because, we want it to communicate with the server as long as we close the connection manually (that should be done by pressing control + C).

Step 4: select

Description

In the last step, our server is limited to one client. So, here the main goal is to handle multiple clients. Similar to the previous part, the server echo whatever it receive form a client but, it send it to all the connected peers. So, all the clients can see each other messages.

Server:

```
socket()
bind()
Listen()
FD_ZERO(&readFds)
FD_SET(serverFd, &readFds)
fdMax = serverFd
While (1) {
        tempReadFds = readFds
        select()

        if(FD_ISSET(serverFd, &tempReadFds)) {
                accept()
                FD_SET(clientFd, &readFds)
                if (fdMax < clientFd) {
                        fdMax = clientFd;
                }
        }

        for (int i=serverFd+1; i<=fdMax; i++) {
                if(FD_ISSET(i, &tempReadFds)) {
                        if (recv(i) <= 0) {
                                close(i)
                                FD_CLR(i, &readFds)
                        }
                        else {
                                for (int j=serverFd+1; j<=fdMax; j++) {
                                        if (FD_ISSET(j, &readFds)) {
                                                send(j)
                                        }
                                }
                        }
                }
        }
}
close()
```

If something happen in the server's fd = A new client arrives

If something happens in one of the client's fd = A message from a current client

Client:

```
socket()
connect()
while (1) {
        select()
        if (FD_ISSET(0,&fds)) {
                fgets()
                send()
        }
        if (FD_ISSET(clientFd,&fds)) {
                recv()
        }
}
close()
```

If something happen in the stdin's fd (0) = An input message from user

If something happens in the client's fd = A message from server

Question 1. Why in the server side, we need to use the select function?

Answer:

```
while(1) {

        accept();

        while (1) {

                recv();
                send();

        }

}
```

A new client is arrived but, it can not go out of the loop because recv() ***blocks*** it.

```
While (1) {
        tempReadFds = readFds
        select()

        if(FD_ISSET(serverFd, &tempReadFds)) {
                accept()
                FD_SET(clientFd, &readFds)
                if (fdMax < clientFd) {
                        fdMax = clientFd;
                }
        }

        for (int i=serverFd+1; i<=fdMax; i++) {
                if(FD_ISSET(i, &tempReadFds)) {
                        if (recv(i) <= 0) {
                                close(i)
                                FD_CLR(i, &readFds)
                        }
                        else {
                                for (int j=serverFd+1; j<=fdMax; j++) {
                                        if (FD_ISSET(j, &readFds)) {
                                                send(j)
                                        }
                                }
                        }
                }
        }
}
```

If something happen in the server's fd
=
A new client arrives

If something happens in one of the client's fd
=
A message from a current client

Question 2. Why in the client side, we need to use select function?

```
while(1) {
        fgets();
        recv();
}
```

The server sends a message (which is one of the clients' message) but, fgets() wait (*blocks* the code) for the client input.

```
while(1) {
        select()
        if (FD_ISSET(0,&fds)) {
                fgets();
        }
        if (FD_ISSET(clientFd,&fds)) {
                recv();
        }
}
```

If there is a user input.

If there is a message from server.

Question 3. Why in the server side we have a for loop before if(FD_ISSET(i, &tempReadFds))?

It is explained in the step's description that we are going to handle multiple clients. Therefore, we should repeat monitoring process of the clients activities as many time as we reach the last one. If we had only one client then, we could simply remove the for loop and replace i variable with client's fd like:

if(FD_ISSET(clientFd, &tempReadFds))

Question 4. Why we need two set of file descriptors?

"On exit, each of the file descriptor sets is modified in place to indicate which file descriptors actually changed status. (Thus, if using select() within a loop, the sets must be reinitialized before each call.)" [2]

"Notice I have two file descriptor sets in the code: master and read_fds (readFds and tempReadFds in our code). The first, master (readFds), holds all the socket descriptors that are currently connected, as well as the socket descriptor that is listening for new connections". [3]

"The reason I have the master (readFds) set is that select() actually changes the set you pass into it to reflect which sockets are ready to read. Since I have to keep track of the connections from one call of

select() to the next, I must store these safely away somewhere. At the last minute, I copy the master into the read_fds, and then call select()". [3]

"But doesn't this mean that every time I get a new connection, I have to add it to the master (readFds) set? Yup!" [3]

"And every time a connection closes, I have to remove it from the master (readFds) set? Yes, it does." [3]

"Notice I check to see when the listener socket is ready to read. When it is, it means I have a new connection pending, and I accept() it and add it to the master (readFds) set". [3]

"Similarly, when a client connection is ready to read, and recv() returns 0, I know the client has closed the connection, and I must remove it from the master (readFds) set". [3]

"If the client recv() returns non-zero, though, I know some data has been received. So I get it, and then go through the master (readFds) list and send that data to all the rest of the connected clients". [3]

References

1: , UBUNTU forums, 2006, https://ubuntuforums.org/showthread.php?t=704600
2: Michael Kerrisk, Linux select man page, 2018, http://man7.org/linux/man-pages/man2/select.2.html
3: Beej Jorgensen, Beej's Guide to Network ProgrammingUsing Internet Sockets, 2016, https://beej.us/guide/bgnet/