





How to implement TCP sockets in C

 Edpresso Team 

TCP sockets are used for communication between a server and a client process. The server's code runs first, which opens a port and listens for incoming connection requests from clients. Once a client connects to the same (server) port, the client or server may send a message. Once the message is sent, whoever receives it (server or client) will process it accordingly.



Server-side

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <sys/socket.h>
4 #include <arpa/inet.h>
5
6 int main(void)
7 {
8     int socket_desc, client_sock, client_size;
9     struct sockaddr_in server_addr, client_addr;
10    char server_message[2000], client_message[2000];
11
12    // Clean buffers:
13    memset(server_message, '\0', sizeof(server_message));
14    memset(client_message, '\0', sizeof(client_message));
15
16    // Create socket:
17    socket_desc = socket(AF_INET, SOCK_STREAM, 0);
18
19    if(socket_desc < 0){
20        printf("Error while creating socket\n");
21        return -1;
22    }
23    printf("Socket created successfully\n");
24
25    // Set port and IP:
26    server_addr.sin_family = AF_INET;
27    server_addr.sin_port = htons(2000);
28    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
29
30    // Bind to the set port and IP:
31    if(bind(socket_desc, (struct sockaddr*)&server_addr, sizeof(serv
```



Server-side

Explanation

Include the header files `sys/socket.h` and `arpa/inet.h`:

```
#include <sys/socket.h>
#include <arpa/inet.h>
```

Create a socket that returns a socket descriptor; this will be used to refer to the socket later on in the code:

```
int socket_desc = socket(AF_INET, SOCK_STREAM, 0);
```

- The server-side code keeps the address information of both the server and the client in a variable of type `sockaddr_in`, which is a `struct`.

Initialize the server address by the port and IP:

```
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(2000);
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

Bind the socket descriptor to the server address:

```
bind(socket_desc, (struct sockaddr*)&server_addr, sizeof(server_addr));
```

Turn on the socket to listen for incoming connections:

```
listen(socket_desc, 1);
```

Store the client's address and socket descriptor by accepting an incoming connection:

```
struct sockaddr client_addr;
int client_size = sizeof(client_addr);
int client_sock = accept(socket_desc, (struct sockaddr*)&client_addr, &client_size);
```

- The server-side code stops and waits at `accept()` until a client calls `connect()`.

Communicate with the client using `send()` and `recv()`:

```
recv(client_sock, client_message, sizeof(client_message), 0);
send(client_sock, server_message, strlen(server_message), 0);
```

- When `recv()` is called, the code stops and waits for a message from the client.

Close the server and client socket to end communication:

```
close(client_sock);
close(socket_desc);
```

Client-side

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <sys/socket.h>
4  #include <arpa/inet.h>
5
6  int main(void)
7  {
8      int socket_desc;
9      struct sockaddr_in server_addr;
10     char server_message[2000], client_message[2000];
11
12     // Clean buffers:
13     memset(server_message, '\0', sizeof(server_message));
14     memset(client_message, '\0', sizeof(client_message));
15
16     // Create socket:
17     socket_desc = socket(AF_INET, SOCK_STREAM, 0);
18
19     if(socket_desc < 0){
20         printf("Unable to create socket\n");
21         return -1;
22     }
23
24     printf("Socket created successfully\n");
25
26     // Set port and IP the same as server-side:
27     server_addr.sin_family = AF_INET;
28     server_addr.sin_port = htons(2000);
29     server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
30
31     // Send connection request to server:

```


[Courses](#)
[Log In](#)
[Join for free](#)

Client-side

Explanation

Include header files, create a socket, and initialize the server's address information in a variable of type `sockaddr_in`, similar to how it was done at the server-side:

```

#include <sys/socket.h>
#include <arpa/inet.h>

int socket_desc = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(2000);
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

```

Send a connection request to the server, which is waiting at `accept()`:

```

connect(socket_desc, (struct sockaddr*)&server_addr, sizeof(server_a
ddr));

```

Communicate with the server using `send()` and `recv()`:

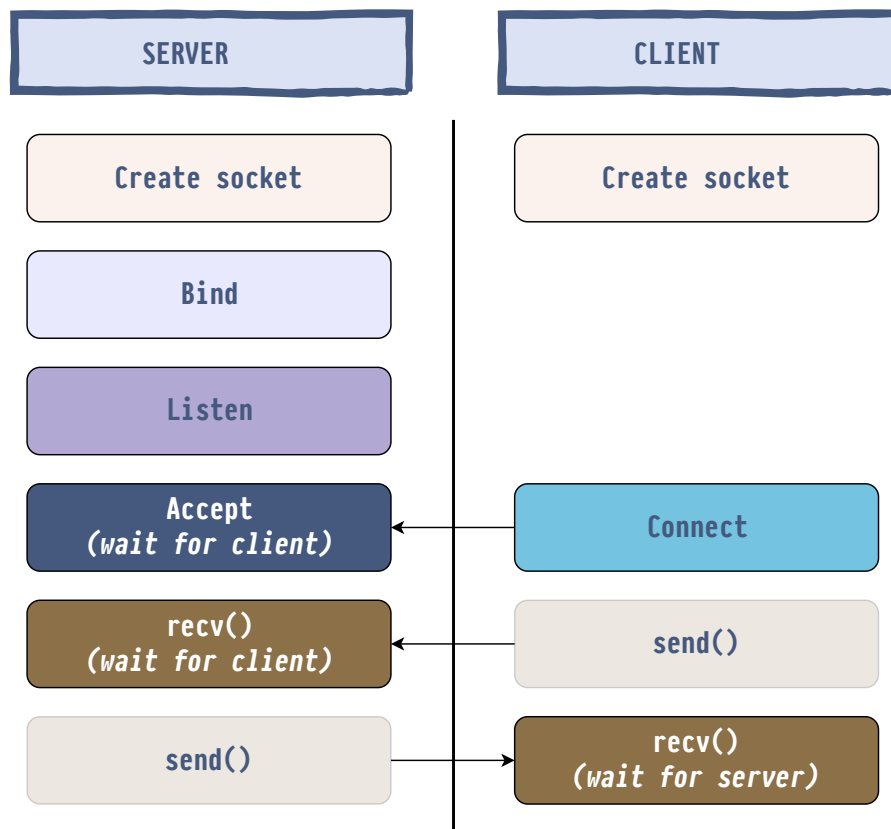
```
send(socket_desc, client_message, strlen(client_message), 0);
recv(socket_desc, server_message, sizeof(server_message), 0);
```

- The client waits for the server to send a message when `recv()` is called.

Close the socket:

```
close(socket_desc);
```

A deadlock will occur if both the client and the server are waiting for each other's message at `recv()`.



Control-flow of a client-server program

Client-server app

The following client-server application enables a client to connect to a server and send *only* one message. The server replies with ***"This is the server's message"*** and the communication terminates.

Instructions

1. Click the **Run** button in the widget below and execute the command for the *Server*. If the socket is created successfully, the message ***Listening for incoming connections...*** will be displayed.
2. Press the + button to open another terminal tab and execute the *Client's* command.
3. Enter a message in the *Client* tab which is sent to the *Server*.
4. The *Server's* response will be shown in the *Client's* tab.

Server	Client
usercode/server	usercode/client

```

#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>

int main(void)
{
    int socket_desc, client_sock, client_size;
    struct sockaddr_in server_addr, client_addr;
    char server_message[100], client_message[100];

    // Clean buffers:
    memset(server_message, '\0', sizeof(server_message));
    memset(client_message, '\0', sizeof(client_message));

    // Create socket:
    socket_desc = socket(AF_INET, SOCK_STREAM, 0);

    if(socket_desc < 0){
        printf("Error while creating socket\n");
        return -1;
    }
    printf("Socket created successfully\n");

    // Set port and IP:
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Bind to the set port and IP:
    if(bind(socket_desc, (struct sockaddr*)&server_addr, sizeof(server_addr))<0){
        printf("Couldn't bind to the port\n");
        return -1;
    }
    printf("Done with binding\n");

    // Listen for clients:
    if(listen(socket_desc, 1) < 0){
        printf("Error while listening\n");
        return -1;
    }
    printf("\nListening for incoming connections.....\n");

    // Accept an incoming connection:
    client_size = sizeof(client_addr);
    client_sock = accept(socket_desc, (struct sockaddr*)&client_addr, &client_size);

    if (client_sock < 0){
        printf("Can't accept\n");
        return -1;
    }
    printf("Client connected at IP: %s and port: %i\n", inet_ntoa(client_addr.sin_addr), client_addr.sin_port);

    // Receive client's message:
    if (recv(client_sock, client_message, sizeof(client_message), 0) < 0){
        printf("Couldn't receive\n");
        return -1;
    }
    printf("Msg from client: %s\n", client_message);

    // Respond to client:
    strcpy(server_message, "This is the server's message.");

    if (send(client_sock, server_message, strlen(server_message), 0) < 0){
        printf("Can't send\n");
        return -1;
    }

    // Closing the socket:
    close(client_sock);
    close(socket_desc);

    return 0;
}

```

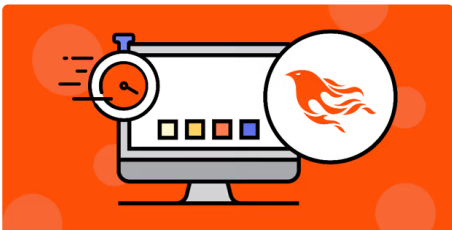
RELATED TAGS

- sockets
- networking
- tcp
- c language

License: Creative Commons -Attribution -
ShareAlike 4.0 (CC-BY-SA 4.0)
(<https://creativecommons.org/licenses/by-sa/4.0/>)



Related Courses



The Pragmatic Programmers

Building Real-time
Applications with Phoenix &
Elixir



Intermediate

Preview →

(/courses/building-real-time-applications-phoenix-elixir)

Keep Exploring

- Socket programming in Python
- How to perform socket programming in Java
- Chromebrew



Learn in-demand tech skills in half the time

SOLUTIONS

Educative for
Enterprise

(/enterprise)
Educative for
Individuals

(/individual-learner)
Educative for HR/
recruiting

(//try.educative.io/recruiting)
Educative for
Bootcamps

(/bootcamps)

RESOURCES

Educative Blog

(/blog)
Edpresso

(/edpresso)

CONTRIBUTE

Become an Author

(https://learn.educative.io/become-an-educative-author)
Become an
Affiliate

(/affiliate)
Become a
Contributor

(/m/write-on-edpresso)

ABOUT US

Our Team

(/team)
Careers Hiring

(/careers)

PRODUCTS

Educative Learning

(/individual-learner)
Educative
Onboarding

(/onboarding)
Educative Skill
Assessments

(/assessments)

PRICING

For Individuals

(/unlimited)
For Enterprise

(/enterprise-pricing)

LEGAL

Privacy Policy

(/privacy)
Terms of Service

(/terms)
Business Terms of
Service

(/enterprise-terms)

MORE

Course Catalog

(/explore)
Early Access
Courses

(/explore/early-access)
Free Trials

(/b2c-trial)
Earn Referral

Credits

(/refer-a-friend)
CodingInterview.com

(/codinginterview.com)
Press

(/press)
Contact Us

(/contactUs)



educativeinc)(//linkedin.com/company/educative-inc/)



(//twitter.com/educativeinc)



(//www.youtube.com/channel/UCT_8FqzTir2Q1B0tvX_DPPw/?sub_confirmation=1)



(//educativese

Copyright ©2022 Educative, Inc. All rights reserved.