Department of Computer Science and Engineering,
College of Engineering, Anna University, Guindy, Chennai.
CS6111 Computer Networks Laboratory

Year/Sem/Batch: III /V / P **Date:** 18-Aug-2022

Exercise: 1 (Study of Socket Programming)

What is Socket programming?

Socket programming is a way of connecting two nodes on a network to communicate with each other. The server socket listens on a particular port at an IP, while the client socket requests the server to establish a connection.

Socket are generally employed in client server applications. The server creates a socket, attaches it to a network port addresses then waits for the client to contact it. The client creates a socket and then attempts to connect to the server socket. When the connection is established, transfer of data takes place.

Types of Sockets:

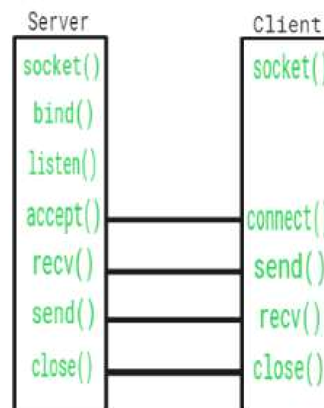
There are two types of Sockets: the datagram socket and the stream socket.

Datagram Socket:

This is a type of network which has connection less point for sending and receiving packets. It is similar to mailbox. The letters (data) posted into the box are collected and delivered (transmitted) to a letterbox (receiving socket).

Stream Socket

In Computer operating system, a stream socket is type of interprocess communications socket or network socket which provides a connection-oriented, sequenced, and unique flow of data without record boundaries with well defined mechanisms for creating and destroying connections and for detecting errors. It is similar to phone. A connection is established between the phones (two ends) and a conversation (transfer of data) takes place.



Function Call	Description
Create()	To create a socket
Bind()	It's a socket identification like a telephone number to contact
Listen()	Ready to receive a connection
Connect()	Ready to act as a sender
Accept()	Confirmation, it is like accepting to receive a call from a sender
Write()	To send data
Read()	To receive data
Close()	To close a connection

Most of the Net Applications use the Client-Server architecture, which refers to two processes or two applications that communicate with each other to exchange some information. One of the two processes acts as a client process, and another process acts as a server.

Client Process

This is the process, which typically makes a request for information. After getting the response, this process may terminate or may do some other processing.

Example, Internet Browser works as a client application, which sends a request to the Web Server to get one HTML webpage.

Server Process

This is the process which takes a request from the clients. After getting a request from the client, this process will perform the required processing, gather the requested information, and send it to the requestor client. Once done, it becomes ready to serve another client. Server processes are always alert and ready to serve incoming requests.

Example – Web Server keeps waiting for requests from Internet Browsers and as soon as it gets any request from a browser, it picks up a requested HTML page and sends it back to that Browser.

Note that the client needs to know the address of the server, but the server does not need to know the address or even the existence of the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

Types of Server

There are two types of servers you can have –

Iterative Server – This is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile, another client keeps waiting.

Concurrent Servers – This type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to fork a child process to handle each client separately.

How to Make Client

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. Both the processes establish their own sockets.

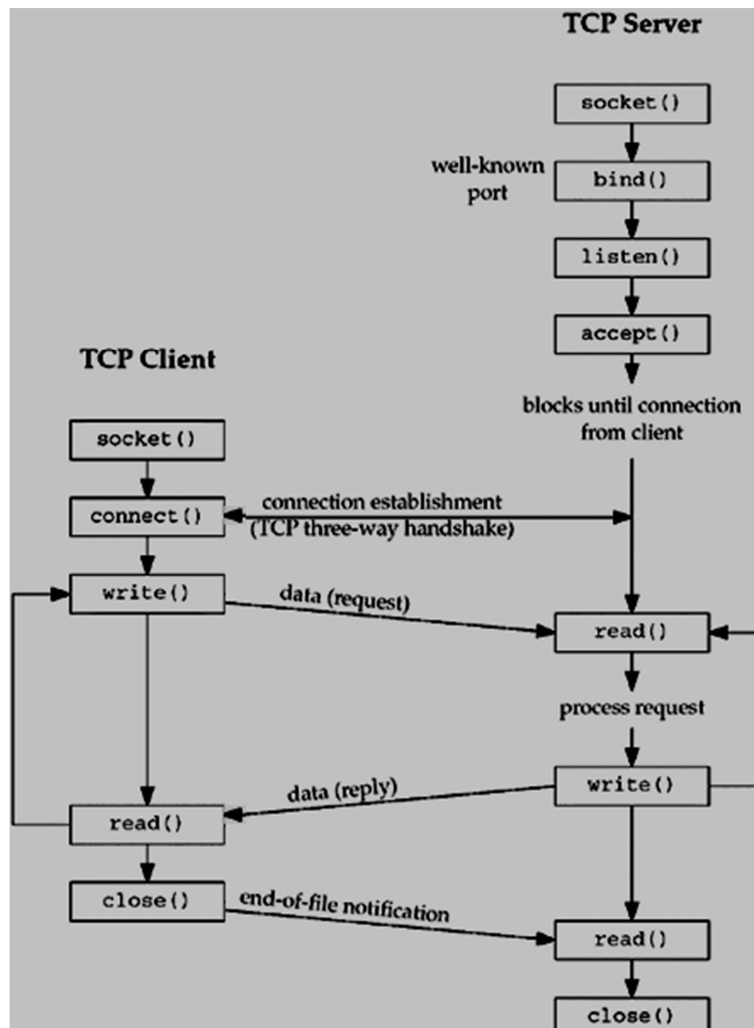
The steps involved in establishing a socket on the client side are as follows –

Step 1. Create a socket with the socket () system call.
Step 2. Connect the socket to the address of the server using the connect () system call.
Step 3. Send and receive data. There are a number of ways to do this, but the simplest way is to use the read() and write() system calls.

How to make a Server?

The steps involved in establishing a socket on the server side are as follows –

Step 1. Create a socket with the socket() system call.
Step 2. Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
Step 3. Listen for connections with the listen() system call.
Step 4. Accept a connection with the accept() system call. This call typically blocks the connection until a client connects with the server.
Step 5. Send and receive data using the read() and write() system calls.



The first step common for both client and the server is creating a socket.

Socket creation:

int sockfd = socket(domain, type, protocol)

sockfd:socket descriptor[return value]

domain:Communication domain e.g., AF_INET

type:communication type

SOCK_STREAM: TCP, SOCK_DGRAM: UDP

protocol:Protocol value for IP->0.

Next steps for server:

Bind:

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

After creation of the socket, bind function binds the socket to the address and port number specified in addr. Returns -1 upon failure.

Listen:

int listen(int sockfd, int maxlen);

Waits for the client to approach the server to make a connection.Maxlenis the maximum lengthto which the queue of pending connections may grow(max number of clients). Returns -1 upon failure.

Accept:

int newsocket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket, creates a new connected socket, and returns a new file descriptor referring to that socket. Connection is established between client and servernow, and they are ready to transfer data. Returns -1 upon failure.

Next step for Client:

Connect:

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.Returns -1 upon failure.

To send and receive data:

int send(int socket_descriptor,char *buffer,int buffer_length,int flags)

-> Returns number ofbytes transmitted, -1 if error.

int recv(int socket_descriptor, char *buffer, int buffer_length, int flags)

-> Returns number ofbytes received, -1 if error.

To close a socket:

status = close(sockid);

sockid: the file descriptor (socket being closed)

status: 0 if successful, -1 if error

This closes a connection and frees up the port used by the socket.

Exercise

1. Create a TCP chat between server and a single client.

tcp_server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#define N 100
#define PORT 3542
int main(){
int serv_sockfd = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv_addr;
```

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
serv_addr.sin_addr.s_addr = INADDR_ANY;
if (bind(serv_sockfd, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) < 0) return 0;
if (listen(serv_sockfd, 1) < 0) return 0;
printf("Server listening on port %d.\n", PORT);
int client_sockfd = accept(serv_sockfd, NULL, NULL);
char buffer_rec[N];
char buffer_send[N];
while(1){
printf("\nSERVER: ");
gets(buffer_send);
if(strcmp(buffer_send,"exit") == 0){
send(client_sockfd, buffer_send, sizeof(buffer_send), 0);
break;
}
send(client_sockfd, buffer_send, sizeof(buffer_send), 0);
recv(client_sockfd, buffer_rec, sizeof(buffer_send), 0);
printf("\nCLIENT: %s\n",buffer_rec);
if(strcmp(buffer_rec,"exit") == 0){
break;
}
}
close(serv_sockfd);
return 0;
}
```

tcp_client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#define N 100
#define PORT 3542
int main(){
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv_addr;
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
serv_addr.sin_addr.s_addr = INADDR_ANY;
int status = connect(sockfd, (struct sockaddr*) &serv_addr, sizeof(serv_addr));
if (status == -1){
printf("Error in making the connection\n");
return 0;
}
printf("Connection established!\n");
```

```
char buffer_rec[N];
char buffer_send[N];
while(1){
recv(sockfd, buffer_rec, sizeof(buffer_send), 0);
printf("\nSERVER: %s\n",buffer_rec);
if(strcmp(buffer_rec,"exit") == 0){
break;
}
printf("\nCLIENT: ");
gets(buffer_send);
if(strcmp(buffer_send,"exit") == 0){
send(sockfd, buffer_send, sizeof(buffer_send), 0);
break;
}
send(sockfd, buffer_send, sizeof(buffer_send), 0);
}
close(sockfd);
return 0;
}
```