

BÁO CÁO LINUX SYSTEM PROGRAMING

VŨ NGỌC CƯỜNG

Mã nguồn: <https://github.com/cuongvungoc/Network-Programing.git>

MỤC LỤC

DANH SÁCH HÌNH VẼ	3
DANH SÁCH BẢNG BIỂU.....	4
CHƯƠNG 1. SOCKET INTRODUCTION.....	5
1.1 Socket overview.....	5
1.2 Communication domain	5
1.3 Socket types	6
1.4 Socket address structure	6
1.5 Network byte order	7
1.6 Address conversion functions.....	8
1.6.1 The inet_aton() and inet_ntoa() functions	8
1.6.2 The inet_pton() and inet_ntop() functions	9
1.6.3 The inet_addr function	9
1.6.4 The sock_ntop function	10
1.7 Socket creation	10
1.8 Binding a socket to an address: bind()	10
1.9 Connection termination: close() and shutdown()	11
1.10 Read and write function	11
1.11 Socket-specific I/O system call: recv() and send()	12
1.12 The sendfile() system call	13
1.13 Select and poll function	14
1.13.1 Select() system call.....	14
1.13.2 Poll() system call.....	17
1.13.3 Select() vs poll()	17
CHƯƠNG 2. TCP SOCKET	19
2.1 Listening for Incoming connections: listen().....	19
2.2 Accepting a connection: accept().....	20
2.3 Connecting to a peer socket: connect()	20
CHƯƠNG 3. UDP SOCKET.....	25
3.1 Exchange datagrams: recvfrom() and sendto()	25
TÀI LIỆU THAM KHẢO	36

DANH SÁCH HÌNH VẼ

Hình 1-1 Socket in OSI model	5
Hình 1-2 Big-endian and little-endian byte order for 2-byte and 4 -byte integers.....	7
Hình 1-4 Sendfile() system call	14
Hình 1-5 Ví dụ chương trình về select() system call.....	17
Hình 1-6 Ví dụ về chương trình poll() system call.....	18
Hình 2-1 Luồng hoạt động của Stream Socket.....	19
Hình 2-2 I/O on stream socket.....	20
Hình 2-3 Stream socket server side	22
Hình 2-4 Stream socket client side	23
Hình 2-5 Kết quả chạy chương trình stream socket	24
Hình 3-1 Luồng hoạt động datagram socket	25
Hình 3-2 Code server side datagram socket.....	27
Hình 3-3 Code client side datagram socket.....	28
Hình 3-4 Kết quả chạy chương trình datagram socket.....	29
Hình 3-5 Code ví dụ server side bài thực hành	32
Hình 3-6 Code client side bài thực hành	34
Hình 3-7 Kết quả chạy bài thực hành	35

DANH SÁCH BẢNG BIỂU

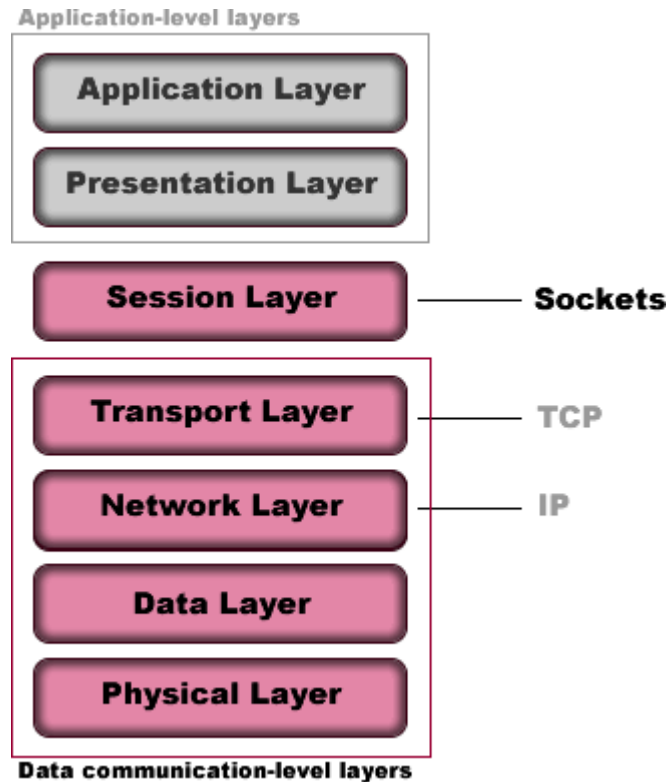
Bảng 1-1 Communication domain.....	5
Bảng 1-2 Socket type.....	6
Bảng 1-3 How in shutdown() system call	11
Bảng 1-4 Flags for recv() system call.....	12
Bảng 1-5 Flags for send() system call	13

CHƯƠNG 1. SOCKET INTRODUCTION

1.1 Socket overview

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 address, while the other socket reaches out to the other from a connection.



Hình 1-1 Socket in OSI model

1.2 Communication domain

Bảng 1-1 Communication domain

Domain	Communication performed	Communication between applications	Address format	Address structure
AF_UNIX	Within kernel	On same host	pathname	Sockaddr_un
AF_INET	Via IPv4	On host connected via an IPv4 network	32 bit IPv4 + 16 bit port number	Sockaddr_in

AF_INET6	Via IPv6	On host connected via an IPv4 network	128 bit IPv4 + 16 bit port number	Sockaddr_in6
----------	----------	---------------------------------------	-----------------------------------	--------------

1.3 Socket types

Bảng 1-2 Socket type

Property	Socket type	
	Stream	Datagram
Reliable delivery	Y	N
Message boundaries preserved	N	Y
Connection-oriented	Y	N

Stream socket: (SOCK_STREAM)

- Provide a reliable, bidirectional, bytes stream communication channel.
- Connection oriented

Datagram Socket (SOCK_DGRAM)

- Allow data to be exchanged in the form of messages called datagrams.
- Message boundaries are preserved, but data transmission is not reliable.
- It doesn't need to be connected to another socket in order to be used (Connectionless).

1.4 Socket address structure

The sockaddr structure is typically defined as follows:

```
struct sockaddr {
    sa_family_t sa_family; /* Address family (AF_* constant) */
    char sa_data[14]; /* Socket address (size varies according to socket domain) */
};
```

This structure serves as a template for all of the domain-specific address structures.

Each of these address structures begins with a family field corresponding to the `sa_family` field of the `sockaddr` structure. (The `sa_family_t` data type is an integer type specified in SUSv3.)

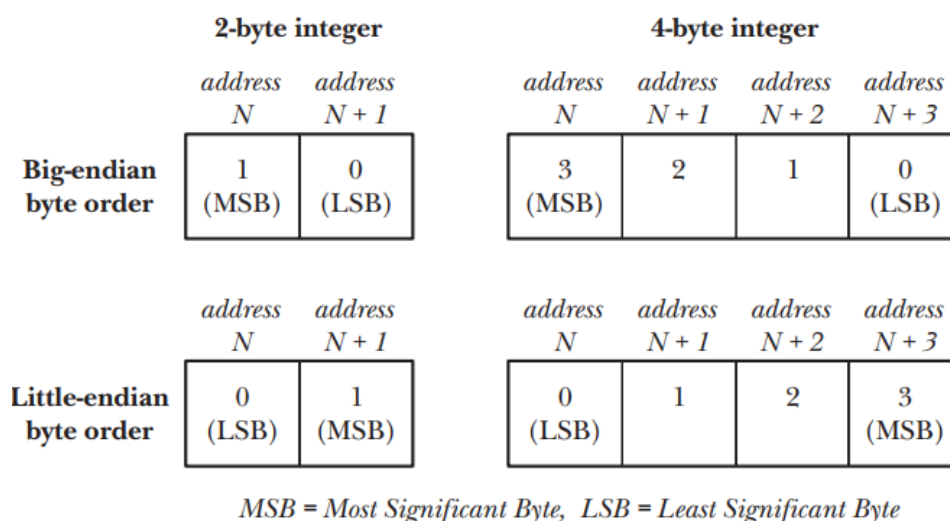
The value in the family field is sufficient to determine the size and format of the address stored in the remainder of the structure.

1.5 Network byte order

- IP addresses and port numbers are integer values.
- One problem we encounter when passing these values across a network is that different hardware architectures store the bytes of a multibyte integer in different orders.

The most notable example of a little-endian architecture is x86.

- Most other architectures are big endian.
- A few hardware architectures are switchable between the two formats.
- The byte ordering used on a particular machine is called the host byte order



Hình 1-2 Big-endian and little-endian byte order for 2-byte and 4 -byte integers

Since port numbers and IP addresses must be transmitted between, and understood by, all hosts on a network, a standard ordering must be used. This ordering is called network byte order and happens to be big endian.

The `htons()`, `htonl()`, `ntohs()`, and `ntohl()` functions are defined (typically as macros) for converting integers in either direction between host and network byte order.

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

```
uint16_t htons(uint16_t host_uint16);
```

Returns host_uint16 converted to network byte order

```
uint32_t htonl(uint32_t host_uint32);
```

Returns host_uint32 converted to network byte order

```
uint16_t ntohs(uint16_t net_uint16);
```

Returns net_uint16 converted to host byte order

```
uint32_t ntohl(uint32_t net_uint32);
```

Returns net_uint32 converted to host byte order

Htons: host to network short

Htonl: host to network long

Short integer: 16 bits

Long integer: 32 bits

1.6 Address conversion functions

1.6.1 The *inet_aton()* and *inet_ntoa()* functions

The *inet_aton()* and *inet_ntoa()* functions convert IPv4 addresses between dotted-decimal notation and binary form (in network byte order). These functions are nowadays made obsolete by *inet_pton()* and *inet_ntop()*.

Inet_aton(ASCII to network): convert the dotted-decimal string pointed to by *str* into an IPv4 address in network byte order, which is returned in the *in_addr* structure pointed to by *addr*.

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

```
int inet_aton(const char *str, struct in_addr *addr);
```


Returns 1 (true) if str is a valid dotted-decimal address, or 0 (false) on error

The `inet_aton()` function returns 1 if the conversion was successful, or 0 if str was invalid.

The `inet_ntoa()` (“network to ASCII”) function performs the converse of `inet_aton()`.

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

```
char *inet_ntoa(struct in_addr addr);
```

Returns pointer to (statically allocated) dotted-decimal string version of
addr

1.6.2 The `inet_pton()` and `inet_ntop()` functions

The `inet_pton()` and `inet_ntop()` functions allow conversion of both IPv4 and IPv6 addresses between binary form and dotted-decimal or hex-string notation.

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

```
int inet_pton(int domain, const char *src_str, void *addrptr);
```

Returns 1 on successful conversion, 0 if src_str is not in presentation
format, or -1 on error

```
const char *inet_ntop(int domain, const void *addrptr, char *dst_str,  
size_t len);
```

Returns pointer to dst_str on success, or NULL on error

P: presentation

N: network

Presentation form is human-readable string:

- 204.152.189.116 (IPv4 dotted-decimal address);
- ::1 (an IPv6 colon-separated hexadecimal address); or
- ::FFFF:204.152.189.116 (an IPv4-mapped IPv6 address)

1.6.3 The `inet_addr` function

The `inet_addr()` function shall convert the string pointed to by cp, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.

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

```
in_addr_t inet_addr(const char *cp);
```

Return the internet address on success, -1 on error

1.6.4 The sock_ntop function

```
#include "unp.h"
```

```
char *sock_ntop(const struct sockaddr *sockaddr, socklen_t addrlen);
```

Return NON-NULL pointer on success, NULL on error

1.7 Socket creation

The socket() system call creates a new socket

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

```
int socket(int domain, int type, int protocol);
```

Returns file descriptor on success, or -1 on error

Protocol argument always specified as 0 for socket types.

Example: int socket(AF_INET, SOCK_STREAM, 0)

1.8 Binding a socket to an address: bind()

The bind() system call binds a socket to an address.

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

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

Returns 0 on success, or -1 on error

- The sockfd argument is a file descriptor obtained from a previous call to socket().
- The addr argument is a pointer to a structure specifying the address to which this socket is to be bound.
- The addrlen argument specifies the size of the address structure.
- The addr argument is a pointer to a structure specifying the address to which this socket is to be bound.

1.9 Connection termination: close() and shutdown()

The usual way of terminating a stream socket connection is to call `close()`. Calling `close()` on a socket closes both halves of the bidirectional communication channel. Sometimes, it is useful to close one half of the connection, so that data can be transmitted in just one direction through the socket. The `shutdown()` system call provides this functionality.

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

```
int shutdown(int sockfd, int how);
```

Returns 0 on success, or -1 on error

The `shutdown()` system call closes one or both channels of the socket `sockfd`, depending on the value of `how`.

Bảng 1-3 How in shutdown() system call

SHUT_RD	Close the reading half of the connection. Subsequent reads will return end-of-file (0). Data can still be written to the socket.
SHUT_WR	Close the writing half of the connection. Once the peer application has read all outstanding data, it will see end-of-file.
SHUT_RDWR	Close both the read and the write halves of the connection. This is the same as performing a SHUT_RD followed by a SHUT_WR.

Aside from the semantics of the `how` argument, `shutdown()` differs from `close()` in another important respect: it closes the socket channel(s) regardless of whether there are other file descriptors referring to the socket. (In other words, `shutdown()` is performing an operation on the open file description, rather than the file descriptor.)

1.10 Read and write function

```
#include"rdwrn.h"
```

```
ssize_t readn(int fd, void *buffer, size_t count);
```

Returns number of bytes read, 0 on EOF, or -1 on error

```
ssize_t writen(int fd, void *buffer, size_t count);
```

Returns number of bytes written, or -1 on error

The `readn()` and `writen()` functions take the same arguments as `read()` and `write()`. However, they use a loop to restart these system calls, thus ensuring that the requested

number of bytes is always transferred (unless an error occurs or end-of-file is detected on a read()).

```
#include "read_line.h"
```

```
ssize_t readLine(int fd, void *buffer, size_t n);
```

Returns number of bytes copied into buffer (excluding terminating null byte), or 0 on end-of-file, or -1 on error

The readLine() function reads bytes from the file referred to by the file descriptor argument fd until a newline is encountered. The input byte sequence is returned in the location pointed to by buffer, which must point to a region of at least n bytes of memory.

1.11 Socket-specific I/O system call: recv() and send()

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

```
ssize_t recv(int sockfd, void *buffer, size_t length, int flags);
```

Returns number of bytes received, 0 on EOF, or -1 on error

```
ssize_t send(int sockfd, const void *buffer, size_t length, int flags);
```

Returns number of bytes sent, or -1 on error

Flags, is a bit mask that modifies the behavior of the I/O operation.

For recv():

Bảng 1-4 Flags for recv() system call

MSG_DONTWAIT	Perform a nonblocking recv(). If no data is available, then instead of blocking, return immediately with the error EAGAIN
MSG_OOB	Receive out-of-band data on the socket.
MSG_PEEK	Retrieve a copy of the requested bytes from the socket buffer, but don't actually remove them from the buffer. The data can later be reread by another recv() or read() call.

MSG_WAITALL	causes the system call to block until length bytes have been received.
-------------	--

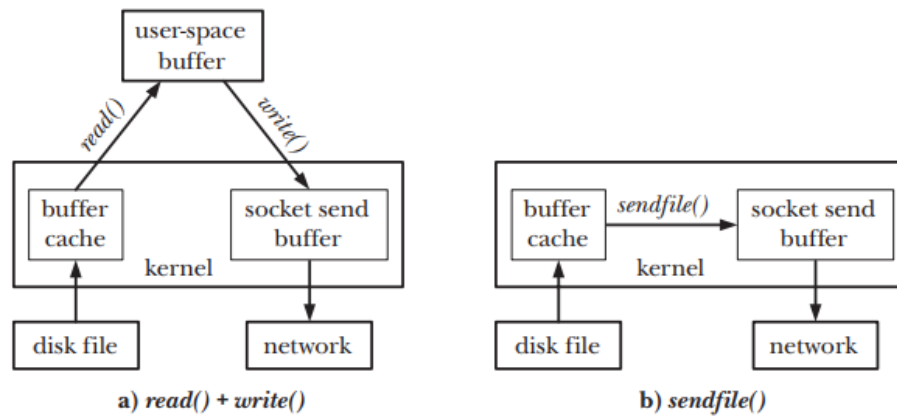
For send()

Bảng 1-5 Flags for send() system call

MSG_DONTWAIT	Perform a nonblocking send(). If the data can't be immediately transferred (because the socket send buffer is full), then, instead of blocking, fail with the error EAGAIN.
MSG_MORE	This flag is used with TCP sockets to achieve the same effect as the TCP_CORK socket option with the difference that it provides corking of data on a per-call basis.
MSG_NOSIGNAL	When sending data on a connected stream socket, don't generate a SIGPIPE signal if the other end of the connection has been closed.
MSG_OOB	Send out-of-band data on a stream socket.

1.12 The sendfile() system call

When an application calls sendfile(), the file contents are transferred directly to the socket, without passing through user space.



Hình 1-3 Sendfile() system call

```
#include <sys/sendfile.h>
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);

Returns number of bytes transferred, or -1 on error
```

The `sendfile()` system call transfers bytes from the file referred to by the descriptor `in_fd` to the file referred to by the descriptor `out_fd`.

Offset:

- If not NULL: point to an `off_t` that specifies the starting file offset from which bytes should be transferred from `in_fd`.
- If offset is NULL: then bytes are transferred from `in_fd` starting at the current file offset, and the file offset is updated to reflect the number of bytes transferred.

Count: specifies the number of bytes to be transferred.

1.13 Select and poll function

1.13.1 *Select()* system call

The `select()` system call blocks until one or more of a set of file descriptors becomes ready.

```
#include <sys/time.h> /* For portability */
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout);
```

Returns number of ready file descriptors, 0 on timeout, or -1 on error

The `nfds`, `readfds`, `writfds`, and `exceptfds` arguments specify the file descriptors that `select()` is to monitor.

The timeout argument can be used to set an upper limit on the time for which `select()` will block

- `readfds` is the set of file descriptors to be tested to see if input is possible;
- `writfds` is the set of file descriptors to be tested to see if output is possible; and
- `exceptfds` is the set of file descriptors to be tested to see if an exceptional condition has occurred.

`Fd_set`: bit mask

```
#include<sys/select.h>
void FD_ZERO(fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
```

Returns true (1) if `fd` is in `fdset`, or false (0) otherwise

- `FD_ZERO()` initializes the set pointed to by `fdset` to be empty.
- `FD_SET()` adds the file descriptor `fd` to the set pointed to by `fdset`.
- `FD_CLR()` removes the file descriptor `fd` from the set pointed to by `fdset`.
- `FD_ISSET()` returns true if the file descriptor `fd` is a member of the set pointed to by `fdset`.

Example:

```
1  #include <stdio.h>
2  #include <sys/time.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5
6  #define TIMEOUT 10      /* Timeout for select()*/
7
8  #define BUF_LEN 1024    /* buffer length */
9
10 int main (void)
11 {
12     struct timeval tv;
13     fd_set readfds;
14     int ret = -1;
15
16     /* Khởi tạo tập hợp readfds và thêm mô tả file stdin vào readfds*/
17     FD_ZERO(&readfds);
18
19     FD_SET(STDIN_FILENO, &readfds);
20
21     /* Thiết lập timeout */
22     tv.tv_sec = TIMEOUT;
23     tv.tv_usec = 0;
24
25     /*Block stdin đến khi stdin sẵn sàng đọc*/
26
27     /*Tập hợp mô tả file writefds và exceptfds truyền vào NULL*/
28     ret = select (STDIN_FILENO + 1, &readfds, NULL, NULL, &tv);
29
30     if (-1 == ret)
31     {
32         perror("Select error.\n" );
33         return 1;
34     }
35     else if (0 == ret)
36     {
37         printf("Timeout after %d seconds.\n" , TIMEOUT);
38         return 0;
39     }
40
41     if (FD_ISSET(STDIN_FILENO, &readfds))
42     {
43         char buf[BUF_LEN+1];
44         int len = -1;
45
46         /* Đọc dữ liệu từ mô tả file của stdin */
47
48         len = read(STDIN_FILENO, buf, BUF_LEN);
49
50         if (-1 == len)
51         {
52             perror("Read fd error.\n" );
53             return 1;
54         }
55
56         if(len)
57         {
58             buf[len] = '\0' ; /*manual vì read() không thêm ký tự null vào cuối string*/
59             printf ("read: %s\n" , buf);
60         }
61         return 0;
62     }
63     return 1;
64 }
```



```

● cuongvn@cuongvn:~/Cuong/Network-Programing/Socket$ ./select
Timeout after 10 seconds.
● cuongvn@cuongvn:~/Cuong/Network-Programing/Socket$ ./select
haha
read: haha
○ cuongvn@cuongvn:~/Cuong/Network-Programing/Socket$ █

```

Hình 1-4 Ví dụ chương trình về select() system call

1.13.2 Poll() system call

Performs a similar task to select().

Select(): three sets, each marked to indicate the fd of interest.

Poll(): Provide a list of fd, each marked with the set of events of interest.

```

#include<poll.h>
int poll(struct pollfd fds[], nfds_t nfds, int timeout);

```

Returns number of ready file descriptors, 0 on timeout, or -1 on error

1.13.3 Select() vs poll()

Within Linux kernel, select() and poll() employ same set of kernel internal poll routine.

Note:

Poll routines are destined from the poll() system call itself

Poll routine is a list of actions corresponding to each file.

Each poll routine execute and return the information of fd see if it ready to read or write.

Poll() has higher performance:

select pass fd-set, nfds

Poll pass fds[]

However, select is more common in practice because it compatible with more system.

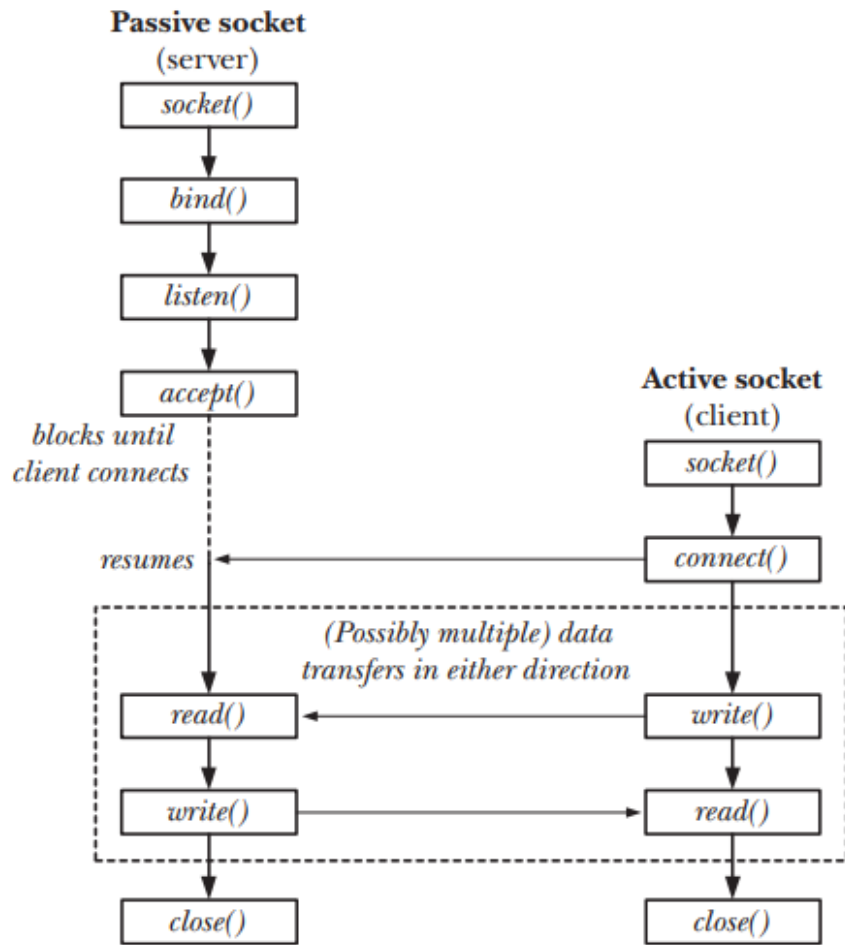
Example:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/poll.h>
4
5  #define TIMEOUT 5 /*timeout 5 seconds*/
6
7  int main(void)
8  {
9      struct pollfd fds[2];
10     int ret = -1;
11
12     /* Kiểm tra stdin sẵn sàng đọc */
13     fds[0].fd = STDIN_FILENO;
14     fds[0].events = POLLIN;
15
16     /* kiểm tra stdout sẵn sàng ghi */
17     fds[1].fd = STDOUT_FILENO;
18     fds[1].events = POLLOUT;
19
20     ret = poll(fds, 2, TIMEOUT * 1000);
21
22     if (-1 == ret)
23     {
24         perror("poll() error!");
25         return -1;
26     }
27     if (0 == ret)
28     {
29         printf("poll() timeout after %d seconds.\n", TIMEOUT);
30         return 0;
31     }
32
33     /*Kiểm tra revents cho stdin va stdout*/
34     if (fds[0].revents & POLLIN)
35     {
36         printf("stdin ready to read\n");
37     }
38     if (fds[1].revents & POLLOUT)
39     {
40         printf("stdout ready to write\n");
41     }
42
43     return 0;
44 }
```

```
● cuongvn@cuongvn:~/Cuong/Network-Programing/Socket$ ./poll
stdout ready to write
● cuongvn@cuongvn:~/Cuong/Network-Programing/Socket$ ./poll < poll.c
stdin ready to read
stdout ready to write
○ cuongvn@cuongvn:~/Cuong/Network-Programing/Socket$
```

Hình 1-5 Ví dụ về chương trình poll() system call

CHƯƠNG 2. TCP SOCKET



Hình 2-1 Luồng hoạt động của Stream Socket

Over view of system calls used with stream socket

2.1 Listening for Incoming connections: listen()

The `listen()` system call marks the stream socket referred to by the file descriptor `sockfd` as passive. The socket will subsequently be used to accept connections from other (active) sockets.

```
#include<sys/socket.h>
int listen(int sockfd, int backlog);
```

Returns 0 on success, or -1 on error

- Backlog: maximum length to which the queue of pending connections for sockfd may grow.

- Connection arrive when queue full - ECONNREFUSED

2.2 Accepting a connection: accept()

The `accept()` system call accepts an incoming connection on the listening stream socket referred to by the file descriptor `sockfd`. If there are no pending connections when `accept()` is called, the call blocks until a connection request arrives.

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

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

Returns file descriptor on success, or `-1` on error

It extracts the first connection request on the queue of pending connections for the listening socket, `sockfd`, creates a new connected socket, and returns a new file descriptor referring to that socket.

2.3 Connecting to a peer socket: connect()

The `connect()` system call connects the active socket referred to by the file descriptor `sockfd` to the listening socket whose address is specified by `addr` and `addrlen`.

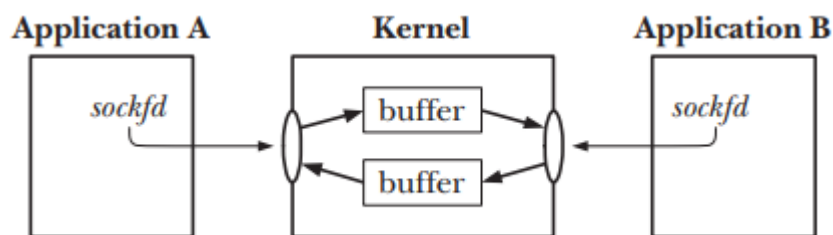
```
#include<sys/socket.h>
```

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

Returns 0 on success, or `-1` on error

The `addr` and `addrlen` arguments same as `bind()`.

I/O on stream socket



Hình 2-2 I/O on stream socket

Stream sockets provide a bidirectional communication channel.

Example:

Server side

```

86 int main(int argc, char const *argv[])
87 {
88     int server_fd, new_socket;
89     int e;
90     struct sockaddr_in address;
91     int opt = 1;
92     int addrlen = sizeof(address);
93
94     FILE *fp;
95     char filename[] = "send.txt";
96
97     // Creating socket file descriptor
98     server_fd = socket(AF_INET, SOCK_STREAM, 0);
99     if (server_fd < 0)
100     {
101         perror("socket failed");
102         exit(EXIT_FAILURE);
103     }
104     printf("Server socket created successfully. \n");
105
106     // Forcefully attaching socket to the port 8080
107     if (setsockopt(server_fd, SOL_SOCKET,
108                   SO_REUSEADDR | SO_REUSEPORT, &opt,
109                   sizeof(opt)))
110     {
111         perror("setsockopt");
112         exit(EXIT_FAILURE);
113     }
114
115     address.sin_family = AF_INET;
116     address.sin_addr.s_addr = INADDR_ANY;
117     address.sin_port = htons(PORT);
118
119     // Forcefully attaching socket to the port 8080
120     e = bind(server_fd, (struct sockaddr *)&address, sizeof(address));
121     if (e < 0)
122     {
123         perror("bind failed");
124         exit(EXIT_FAILURE);
125     }
126     printf("Binding successfull.\n");
127
128     if (listen(server_fd, 3) == 0)
129     {
130         printf("Listening...\n");
131     }
132     else
133     {
134         perror("Error in listening\n");
135         exit(EXIT_FAILURE);
136     }
137
138     new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen);
139     if (new_socket < 0)
140     {
141         perror("accept");
142         exit(EXIT_FAILURE);
143     }
144
145     struct sockaddr_in *pV4Addr = (struct sockaddr_in *)&address;
146     struct in_addr ipAddr = pV4Addr->sin_addr;
147     char str[INET_ADDRSTRLEN];
148     inet_ntop(AF_INET, &ipAddr, str, INET_ADDRSTRLEN);
149     printf("Listening from client with IP: %s, PORT: %d\n", str, ntohs(pV4Addr->sin_port));
150

```

```

19 void send_file(FILE *fp, int sockfd)
20 {
21     printf("Send file called!\n");
22     char data[SIZE] = {0};
23
24     while (fgets(data, SIZE, fp) != NULL)
25     {
26         if (send(sockfd, data, sizeof(data), 0) == -1)
27         {
28             perror("Error in sending file");
29             exit(1);
30         }
31         bzero(data, SIZE);
32     }
33     fseek(fp, 0L, SEEK_END);    // seek to the EOF
34     int size = ftell(fp);
35     rewind(fp);
36
37     printf("Number of bytes sent: %d bytes\n", size);
38     printf("File data sent successfully.\n");
39 }

```

Hình 2-3 Stream socket server side

Client side

```

68     char *ip = "127.0.0.1";
69     int e;
70
71     int sockfd;
72     struct sockaddr_in server_addr;
73
74     sockfd = socket(AF_INET, SOCK_STREAM, 0);
75     if (sockfd < 0)
76     {
77         printf("\n Socket creation error \n");
78         return -1;
79     }
80     printf("Server socket created successfully.\n");
81
82     server_addr.sin_family = AF_INET;
83     server_addr.sin_port = htons(PORT);
84     server_addr.sin_addr.s_addr = inet_addr(ip);
85
86     e = connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
87     if (e == -1)
88     {
89         printf("\nConnection Failed \n");
90         return -1;
91     }
92     printf("Connected to Server.\n");
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Hình 2-4 Stream socket client side

```
cuongvn@cuongvn:~/Cuong/Network-Programing/Socket/StreamSocket$ ./server
Server socket created successfully.
Binding successfull.
Listening...
Listening from client with IP: 127.0.0.1, PORT: 54510
Send file called!
Number of bytes sent: 41 bytes
File data sent successfully.

cuongvn@cuongvn:~/Cuong/Network-Programing/Socket/StreamSocket$ ./client
Server socket created successfully.
Connected to Server.
-----
1. List all file on server
2. Download a file on server
-----
Enter your selection: 2
Data written in the file successfully.
-----

Socket > StreamSocket > ≡ send.txt
1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 | Line 6

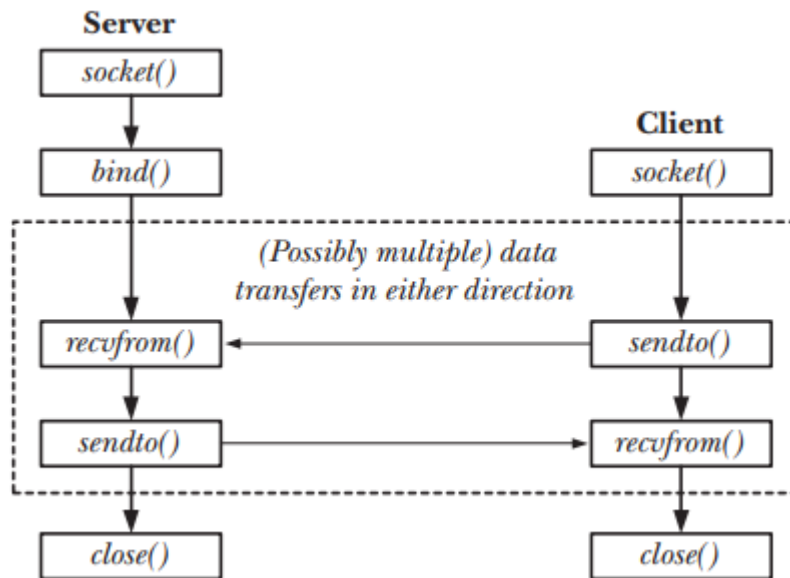
Socket > StreamSocket > ≡ recv.txt
1 Line 1
2 Line 2
3 Line 3
4 Line 4
5 Line 5
6 | Line 6
```

Hình 2-5 Kết quả chạy chương trình stream socket

CHƯƠNG 3. UDP SOCKET

Datagram Socket is also known by another name which is a connectionless Socket. Accordingly, this is a Socket that works over the User Datagram Protocol (UDP).

It can work at its best even without establishing a connection between the two machines.



Hình 3-1 Luồng hoạt động datagram socket

Overview of system calls used with datagram sockets

3.1 Exchange datagrams: recvfrom() and sendto()

The recvfrom() and sendto() system calls receive and send datagrams on a datagram socket.

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

```
ssize_t recvfrom(int sockfd, void *buffer, size_t length, int flags,  
                  struct sockaddr *src_addr, socklen_t *addrlen);
```

Returns number of bytes received, 0 on EOF, or -1 on error

```
ssize_t sendto(int sockfd, const void *buffer, size_t length, int flags,  
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

Returns number of bytes sent, or -1 on error

Flags – a bit mask controlling socket-specific I/O features. Can specify flags as 0 if don't require any of these features.

Example:

Server side

```
int sockfd, nBytes;
struct sockaddr_in addr_con;
int addrlen = sizeof(addr_con);
addr_con.sin_family = AF_INET;
addr_con.sin_port = htons(PORT_NO);
addr_con.sin_addr.s_addr = INADDR_ANY;
char net_buf[NET_BUF_SIZE];
FILE *fp;

// socket()
sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

if (sockfd < 0)
    printf("\nfile descriptor not received!!\n");
else
    printf("\nfile descriptor %d received\n", sockfd);

// bind()
if (bind(sockfd, (struct sockaddr *)&addr_con, sizeof(addr_con)) == 0)
    printf("\nSuccessfully binded!\n");
else
    printf("\nBinding Failed!\n");

while (1)
{
    printf("\nWaiting for file name...\n");

    // receive file name
    clearBuf(net_buf);

    nBytes = recvfrom(sockfd, net_buf,
                      NET_BUF_SIZE, 0,
                      (struct sockaddr *)&addr_con, &addrlen);

    fp = fopen(net_buf, "r");
    printf("\nFile Name Received: %s\n", net_buf);
    if (fp == NULL)
        printf("\nFile open failed!\n");
    else
        printf("\nFile Successfully opened!\n");

    // process
    if (sendFile(fp, net_buf, NET_BUF_SIZE))
    {
        sendto(sockfd, net_buf, NET_BUF_SIZE,
                0,
                (struct sockaddr *)&addr_con, addrlen);
        break;
    }

    // send
    sendto(sockfd, net_buf, NET_BUF_SIZE,
            0,
            (struct sockaddr *)&addr_con, addrlen);
    clearBuf(net_buf);
}
if (fp != NULL)
    fclose(fp);
}
```

```

32 // function sending file
33 int sendFile(FILE *fp, char *buf, int s)
34 {
35     int i, len;
36     if (fp == NULL)
37     {
38         strcpy(buf, nofile);
39         len = strlen(nofile);
40         buf[len] = EOF;
41         for (i = 0; i <= len; i++)
42             buf[i] = Cipher(buf[i]);
43         return 1;
44     }
45
46     char ch, ch2;
47     for (i = 0; i < s; i++)
48     {
49         ch = fgetc(fp);
50         ch2 = Cipher(ch);
51         buf[i] = ch2;
52         if (ch == EOF)
53             return 1;
54     }
55     return 0;
56 }

```

Hình 3-2 Code server side datagram socket

Client side

```

int main()
{
    int sockfd, nBytes;
    struct sockaddr_in addr_con;
    int addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    char net_buf[NET_BUF_SIZE];
    FILE *fp;

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM,
                   IP_PROTOCOL);

    if (sockfd < 0)
        printf("\nfile descriptor not received!!\n");
    else
        printf("\nfile descriptor %d received\n", sockfd);

    while (1)
    {
        printf("\nPlease enter file name to receive:\n");
        scanf("%s", net_buf);
        sendto(sockfd, net_buf, NET_BUF_SIZE,
              sendrecvflag, (struct sockaddr *)&addr_con,
              addrlen);

        printf("\n-----Data Received-----\n");

        while (1)
        {
            // receive
            clearBuf(net_buf);
            nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
                             sendrecvflag, (struct sockaddr *)&addr_con,
                             &addrlen);

            // process
            if (recvFile(net_buf, NET_BUF_SIZE))
            {
                break;
            }
        }
        printf("\n-----\n");
    }
}

```

```

32 // function to receive file
33 int recvFile(char *buf, int s)
34 {
35     int i;
36     char ch;
37     for (i = 0; i < s; i++)
38     {
39         ch = buf[i];
40         ch = Cipher(ch);
41         if (ch == EOF)
42             return 1;
43         else
44             printf("%c", ch);
45     }
46     return 0;
47 }

```

Hình 3-3 Code client side datagram socket

```

cuongvn@cuongvn:~/Cuong/Network-Programing/Socket/UDPSocket$ ./server
file descriptor 3 received
Successfully binded!
Waiting for file name...
File Name Received: send.txt
File Successfully opened!
Waiting for file name...
^

cuongvn@cuongvn:~/Cuong/Network-Programing/Socket/UDPSocket$ ./client
file descriptor 3 received

Please enter file name to receive:
send.txt

-----Data Received-----
haha
hehe
huhu
hihi

```

Hình 3-4 Kết quả chạy chương trình datagram socket

Example:

Server chứa một số danh sách file binary. Client có thể request danh sách các file binary hiện có trên server. Client có thể nhập tên file và tải về từ server. File nhận được ở Client phải giữ nguyên tính toàn vẹn.

Cho phép nhiều client cùng truy cập

Server side

```

#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <dirent.h>

#define PORT 8080
#define SIZE 1024
#define LIST_FILES '1'
#define DOWNLOAD '2'
#define QUIT 'q'

```

```

int server_fd, new_socket;
int e;
struct sockaddr_in address;
int opt = 1;
int addrlen = sizeof(address);

FILE *fp;

// Creating socket file descriptor
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd < 0)
{
    perror("socket failed");
    exit(EXIT_FAILURE);
}
printf("Server socket created successfully. \n");

```

```

// Forcefully attaching socket to the port 8080
if (setsockopt(server_fd, SOL_SOCKET,
               SO_REUSEADDR | SO_REUSEPORT, &opt,
               sizeof(opt)))
{
    perror("setsockopt");
    exit(EXIT_FAILURE);
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

```

```

// Forcefully attaching socket to the port 8080
e = bind(server_fd, (struct sockaddr *)&address, sizeof(address));
if (e < 0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}
printf("Binding successfull.\n");

if (listen(server_fd, 3) == 0)
{
    printf("Listening...\n");
}
else
{
    perror("Error in listening\n");
    exit(EXIT_FAILURE);
}

```

```

new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen);
if (new_socket < 0)
{
    perror("accept");
    exit(EXIT_FAILURE);
}

struct sockaddr_in *pV4Addr = (struct sockaddr_in *)&address;
struct in_addr ipAddr = pV4Addr->sin_addr;
char str[INET_ADDRSTRLEN];
inet_ntop(AF_INET, &ipAddr, str, INET_ADDRSTRLEN);
printf("Listening from client with IP: %s, PORT: %d\n", str, ntohs(pV4Addr->sin_port));

```

```

pthread_t threads[MAX_THREAD];
int n_thread = 0;
// pthread_t thread_id;
while (1)
{
    new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen);
    if (new_socket < 0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    printf("Connection accepted from: %s, PORT: %d\n", inet_ntoa(address.sin_addr), ntohs(address.sin_port));
    pthread_create(&threads[n_thread++], NULL, handle_client, (int *)new_socket);
}

pthread_exit(NULL);

// closing the connected socket
close(new_socket);
// closing the listening socket
shutdown(server_fd, SHUT_RDWR);
return 0;

```

```

void send_file(FILE *fp, int sockfd)
{
    printf("Send file called!\n");
    char data[SIZE] = {0};

    while (fgets(data, SIZE, fp) != NULL)
    {
        if (send(sockfd, data, sizeof(data), 0) == -1)
        {
            perror("Error in sending file");
            exit(1);
        }
        bzero(data, SIZE);
    }
    fseek(fp, 0L, SEEK_END); // seek to the EOF
    int size = ftell(fp);
    rewind(fp);

    printf("Number of bytes sent: %d bytes\n", size);
    printf("File data sent successfully.\n");
}

```

```

void list_file(int sockfd, int new_socket)
{
    struct dirent *dir;
    DIR *d;
    d = opendir(".");
    char quit[100] = "q";
    if (d)
    {
        while ((dir = readdir(d)) != NULL)
        {
            char buffer[SIZE] = {0};
            send(new_socket, dir->d_name, strlen(dir->d_name), 0);
            read(new_socket, buffer, SIZE);
        }
    }
    send(new_socket, quit, strlen(quit), 0);

    free(dir);
    free(d);
}

```

```

63 void *handle_client(void *param)
64 {
65     char buffer[SIZE] = {0};
66     int new_socket = (int *)param;
67     read(new_socket, buffer, SIZE);
68     if (buffer[0] == QUIT)
69     {
70         printf("Disconnected from client\n");
71         pthread_exit(NULL);
72     }
73     else if (buffer[0] == LIST_FILES)
74     {
75         printf("Reading from client!\n");
76         list_file(new_socket);
77     }
78     else if (buffer[0] == DOWNLOAD)
79     {
80         char file_name[SIZE];
81         read(new_socket, file_name, SIZE);
82         FILE *fp;
83         fp = fopen(file_name, "r");
84         if (fp == NULL)
85         {
86             perror("Error in reading file.\n");
87             exit(EXIT_FAILURE);
88         }
89         send_file(fp, new_socket);
90         pthread_exit(NULL);
91         // break;
92     }
93 }

```

Hình 3-5 Code ví dụ server side bài thực hành

Client side

```

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

#define PORT 8080
#define SIZE 1024
#define LIST_FILES '1'
#define DOWNLOAD '2'
#define QUIT 'q'

```



```

char *ip = "127.0.0.1";
int e;

int sockfd;
struct sockaddr_in server_addr;

sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
{
    printf("\n Socket creation error \n");
    return -1;
}
printf("Server socket created successfully.\n");

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);
server_addr.sin_addr.s_addr = inet_addr(ip);

e = connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr));
if (e == -1)
{
    printf("\nConnection Failed \n");
    return -1;
}
printf("Connected to Server.\n");

```

```

while (1)
{
    printf("-----\n");
    printf("1. List all file on server\n");
    printf("2. Download a file on server\n");
    printf("q. Quit the program\n");
    printf("-----\n");
    printf("Enter your selection: ");
    scanf("%s", select);

    if (select[0] == QUIT)
    {
        send(sockfd, select, sizeof(select), 0);

        close(sockfd);
        return 0;
    }
    else if (select[0] == LIST_FILES)
    {
        send(sockfd, select, sizeof(select), 0);

        recv_list(sockfd);
    }
}

```

```

    else if (select[0] == DOWNLOAD)
    {
        char dir[SIZE] = "./recv_file/";
        char file_name[SIZE];
        send(sockfd, select, sizeof(select), 0);
        printf("Enter file name to download: ");
        scanf("%s", file_name);
        send(sockfd, file_name, sizeof(select), 0);
        strcat(dir, file_name);
        fp = fopen(dir, "w");
        write_file(fp, sockfd);
        // break;
    }
}

// closing the connected socket
free(fp);
close(sockfd);
return 0;
}

```

```

void recv_list(int new_socket)
{
    int valread;
    char received[1024] = "Received!";
    printf("In receive list function\n");
    while (1)
    {
        char buffer[1024] = {0};
        valread = read(new_socket, buffer, 1024);
        if (valread > 0)
        {
            send(new_socket, received, sizeof(received), 0);

            if (buffer[0] == 'q')
            {
                break;
            }
            printf("%s\n", buffer);
        }
    }
}

```

```

void wirte_file(FILE *fp, int sockfd)
{
    int n;
    char buffer[SIZE];

    while (1)
    {
        n = recv(sockfd, buffer, SIZE, 0);
        if (n <= 0)
        {
            break;
            return;
        }
        fprintf(fp, "%s", buffer);
        // fwrite(buffer, SIZE, 1, fp);
        bzero(buffer, SIZE);
    }
    printf("Data written in the file successfully.\n");
    return;
}

```

Hình 3-6 Code client side bài thực hành

```

cuongvn@cuongvn:~/Cuong/Network-Programing/Socket/StreamSocket$ ./server
Server socket created successfully.
Binding successfull.
Listening...
Connection accepted from: 127.0.0.1, PORT: 41458
Connection accepted from: 127.0.0.1, PORT: 41466
Reading from client!
Disconnected from client
Connection accepted from: 127.0.0.1, PORT: 49946
Error in reading file.
: No such file or directory

```

```

Server socket created successfully.
Connected to Server.
-----
1. List all file on server
2. Download a file on server
q. Quit the program
-----
Enter your selection: 1
In receive list function
server
recv_file
..
picture.png
send.txt
tcp_client.c
recv.txt
tcp_server.c
test2.c
thread.c
test.c
client
.
-----
1. List all file on server
2. Download a file on server
q. Quit the program
-----
Enter your selection: q
cuongvn@cuongvn:~/Cuong/Network-Programing/Socket/StreamSocket$ S

```

```

cuongvn@cuongvn:~/Cuong/Network-Programing/Socket/StreamSocket$ ./client
Server socket created successfully.
Connected to Server.
-----
1. List all file on server
2. Download a file on server
q. Quit the program
-----
Enter your selection: 2
Enter file name to download: tesc.c
Data written in the file successfully.
-----
1. List all file on server
2. Download a file on server
q. Quit the program
-----
Enter your selection:

```

Hình 3-7 Kết quả chạy bài thực hành

TÀI LIỆU THAM KHẢO

- [1] “Socket Programming in C/C++ - GeeksforGeeks.” <https://www.geeksforgeeks.org/socket-programming-cc/> (accessed Feb. 21, 2023).
- [2] “Socket Programming in C/C++: Handling multiple clients on server without multi threading,” *GeeksforGeeks*, Aug. 29, 2016. <https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/> (accessed Feb. 21, 2023).
- [3] M. Kerrisk, *The Linux programming interface: a Linux and UNIX system programming handbook*. San Francisco: No Starch Press, 2010.