計算機網路概論

Lab3 Socket Programming I

GCC Install

For Mac User

- /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)”
- brew update
- brew upgrade
- brew info gcc
- brew install gcc
- brew cleanup

For Ubuntu User

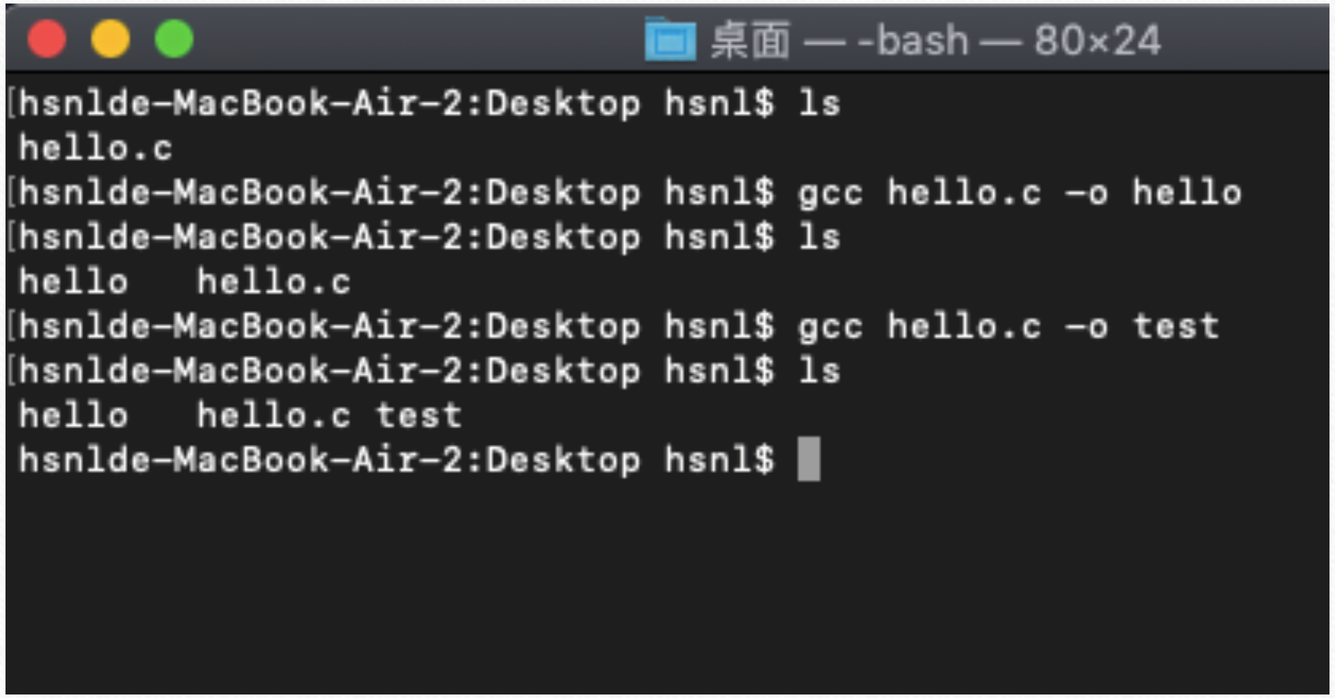
- sudo apt-get update
- sudo apt-get upgrade
- sudo apt-get install build-essential

GCC Compile

- gcc <程式碼檔名> -o <執行檔檔名>
- ./<執行檔檔名>

- Ex: 寫好的 C 程式碼檔名為 hello.c
    - (1). 在 terminal 相應的路徑下輸入 gcc hello.c –o hello
    - (2). ./hello

GCC Compile



```
[hsnlde-MacBook-Air-2:Desktop hsnl$ ls
hello.c
[hsnlde-MacBook-Air-2:Desktop hsnl$ gcc hello.c -o hello
[hsnlde-MacBook-Air-2:Desktop hsnl$ ls
hello   hello.c
[hsnlde-MacBook-Air-2:Desktop hsnl$ gcc hello.c -o test
[hsnlde-MacBook-Air-2:Desktop hsnl$ ls
hello   hello.c test
hsnlde-MacBook-Air-2:Desktop hsnl$ 
```

Common commands

- cd: 切換路徑
- mkdir: 產生目錄
- ls: 列出目前路徑下的的所有檔案
- mv: 搬移或更名檔案
- rm: 刪除檔案
- cp: 複製檔案

- 參考網站：
  - http://linux.vbird.org/linux_basic/redhat6.1/linux_06command.php

相關知識

IP Port

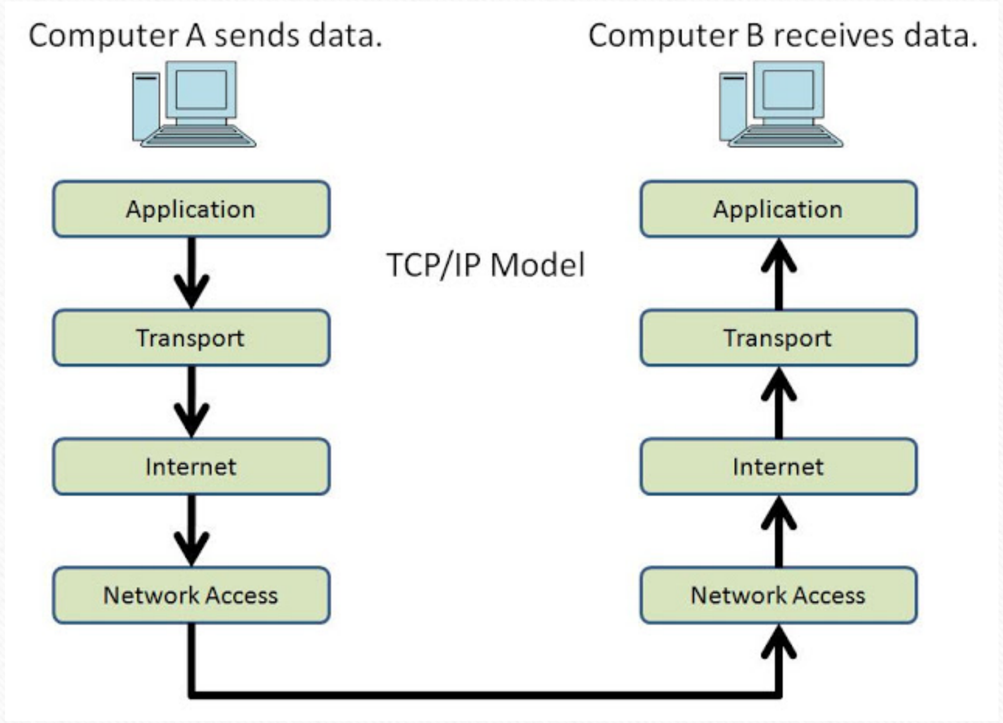
- IP (32 bits)- 每台主機在網際網路上的住址
  - 唯一且不可重複
- Port(16 bits) - 家裡的信箱
  - 不同的信箱只能接收或傳送同一個應用程式的資料。
- Ex. 104.155.203.153:443

IP Port

0-1023	Well-known ports
1024-49151	Registered ports
49152-65535	Dynamic ports

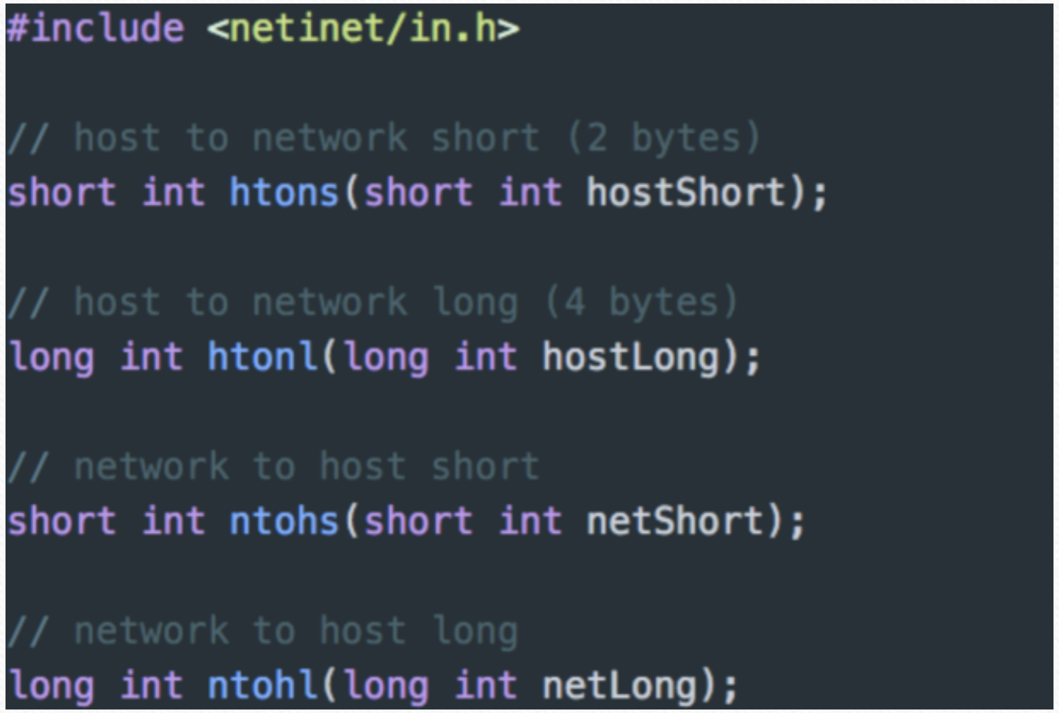
Socket

- 分為TCP & UDP
- 當有資料要傳入/傳出應用層時，必須藉由 socket 與傳輸層接洽



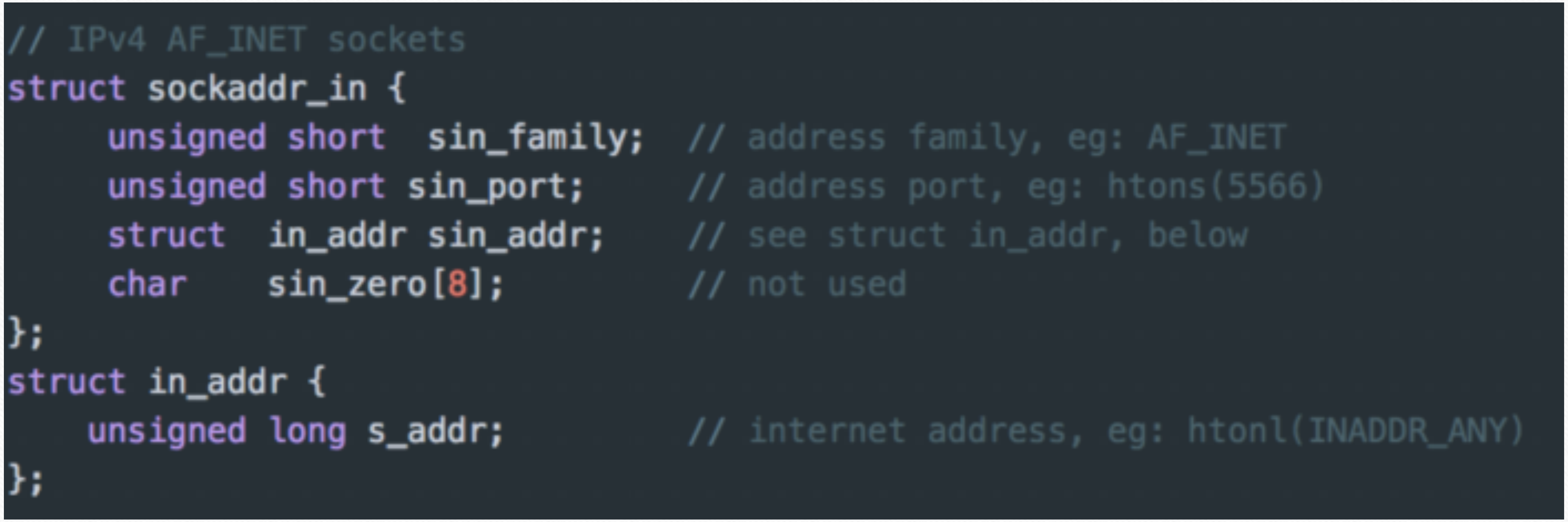
Socket 特殊格式

- 網路中使用的傳輸格式與一般不同
  - 所有格式必須經過轉換
- H: host
- S: short
- N: network
- L: long



```c
#include <netinet/in.h>

// host to network short (2 bytes)
short int htons(short int hostShort);

// host to network long (4 bytes)
long int htonl(long int hostLong);

// network to host short
short int ntohs(short int netShort);

// network to host long
long int ntohl(long int netLong);
```

Socket 特殊格式



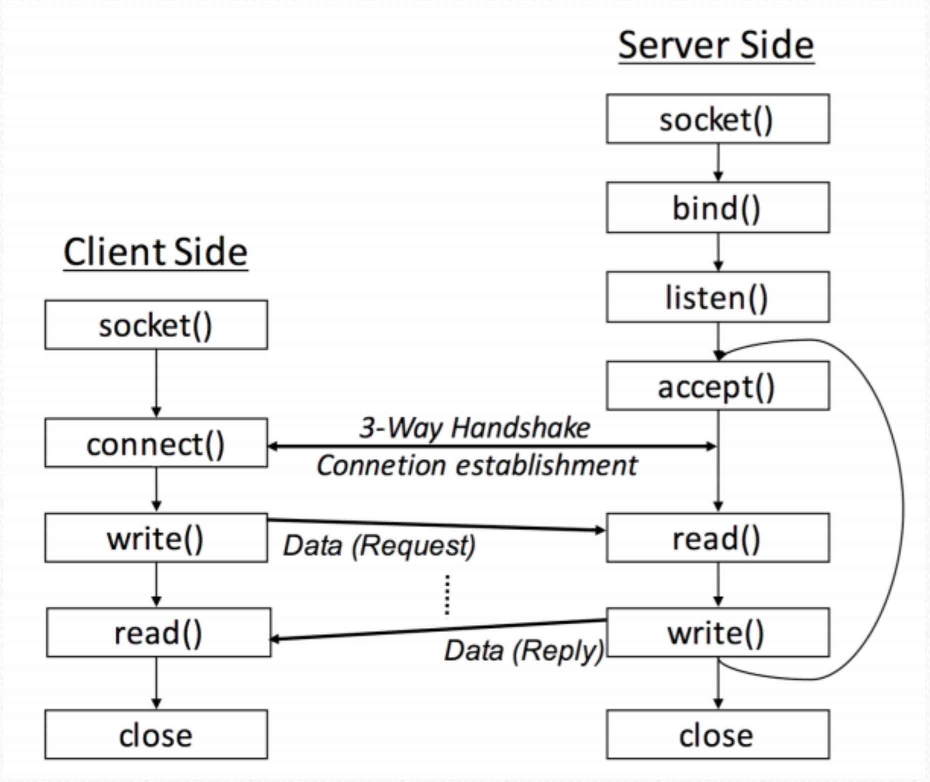
```
// IPv4 AF_INET sockets
struct sockaddr_in {
     unsigned short  sin_family;  // address family, eg: AF_INET
     unsigned short sin_port;     // address port, eg: htons(5566)
     struct  in_addr sin_addr;    // see struct in_addr, below
     char    sin_zero[8];         // not used
};
struct in_addr {
    unsigned long s_addr;         // internet address, eg: htonl(INADDR_ANY)
};
```

Related Library

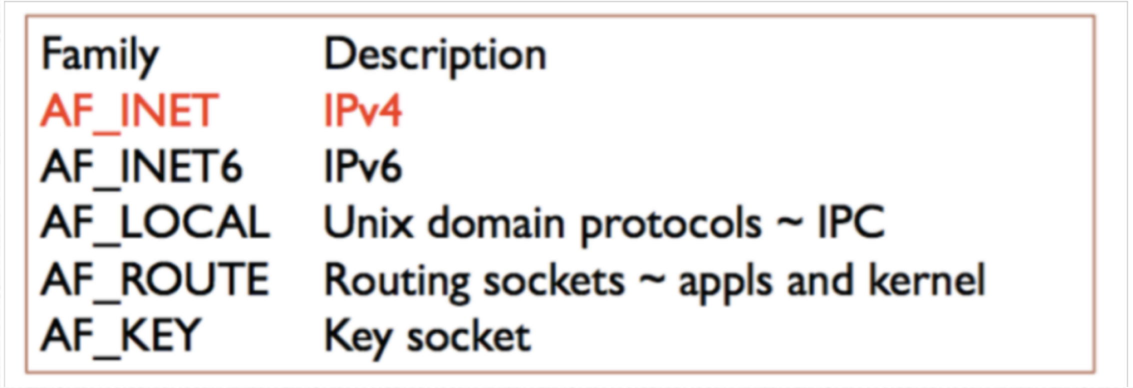
- 查詢 Library
  - 特殊變數
  - 函式
  - 函式的 input/output
- http://pubs.opengroup.org/onlinepubs/7908799/

Socket Programming

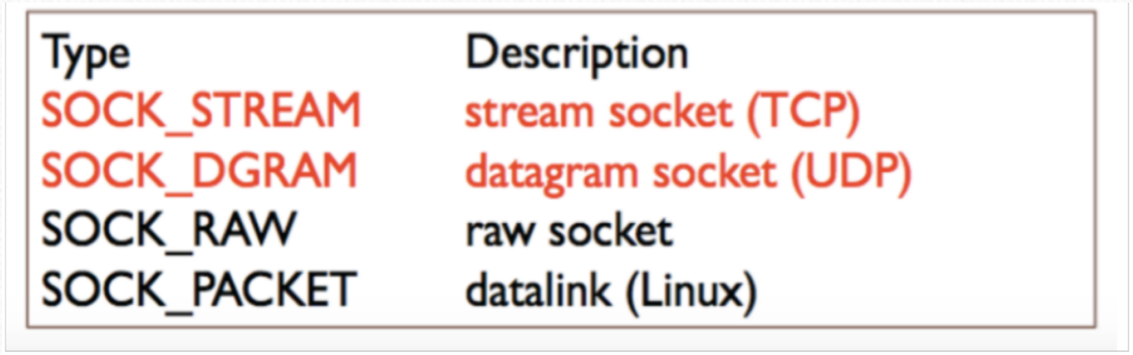
TCP Socket 運作流程



sys/socket.h 特殊定數



Family	Description
AF_INET	IPv4
AF_INET6	IPv6
AF_LOCAL	Unix domain protocols ~ IPC
AF_ROUTE	Routing sockets ~ appls and kernel
AF_KEY	Key socket

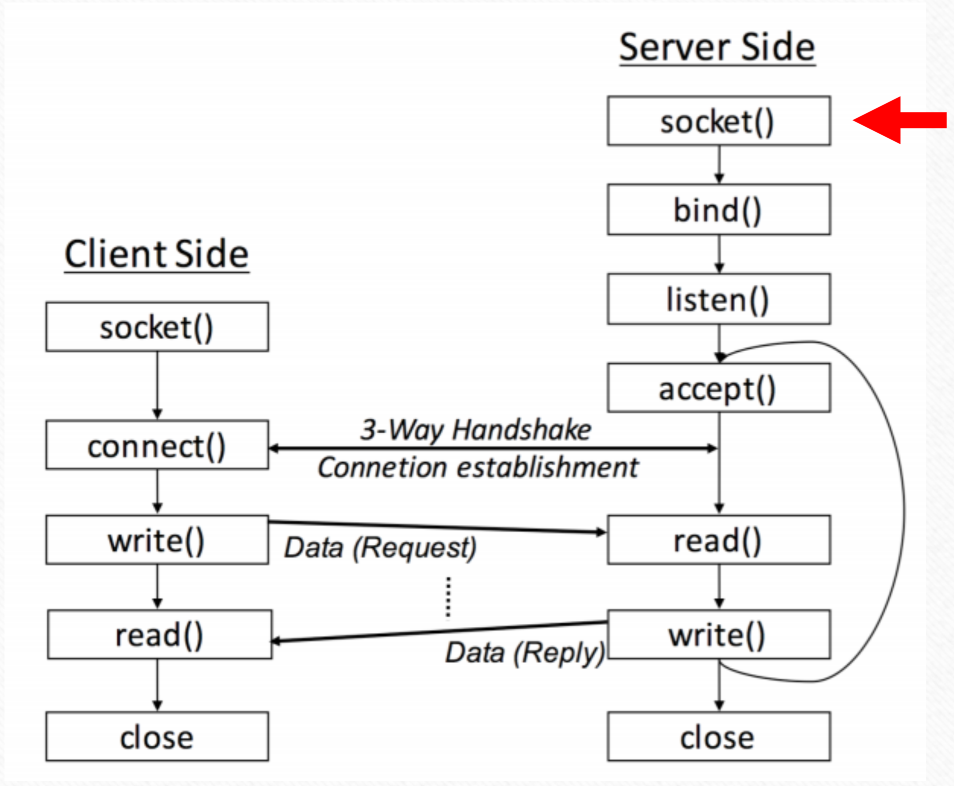


Type	Description
SOCK_STREAM	stream socket (TCP)
SOCK_DGRAM	datagram socket (UDP)
SOCK_RAW	raw socket
SOCK_PACKET	datalink (Linux)

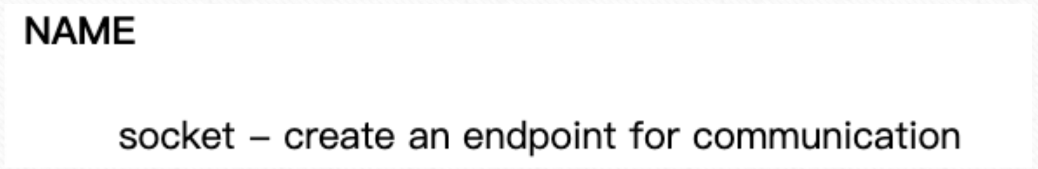
Socket Programming

Server 端

Step1. Create socket

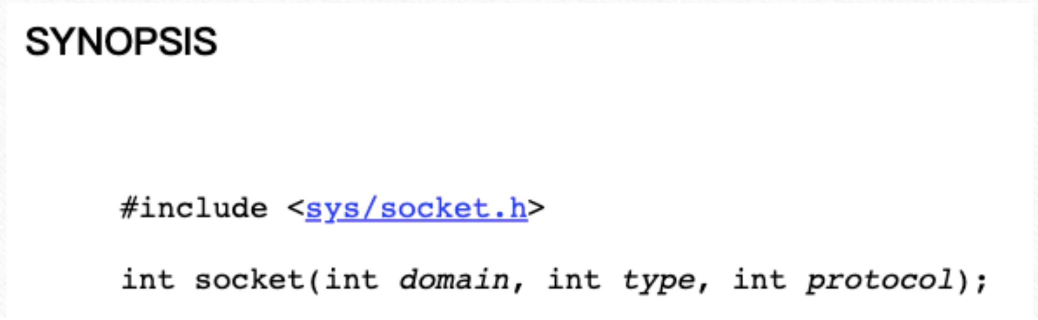


Socket( )



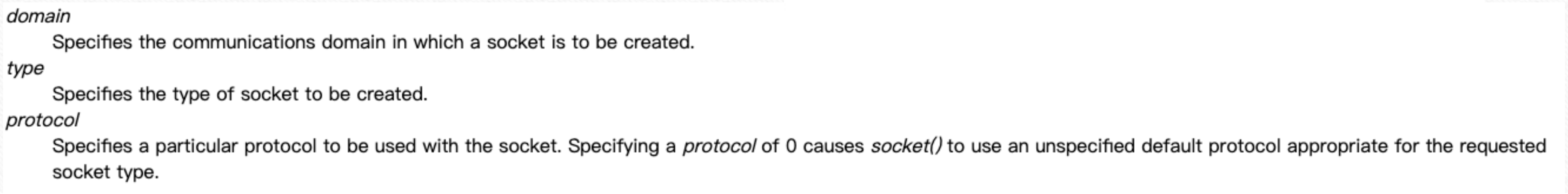
NAME

socket – create an endpoint for communication

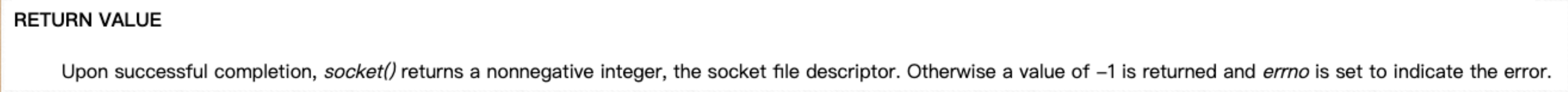


SYNOPSIS

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

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



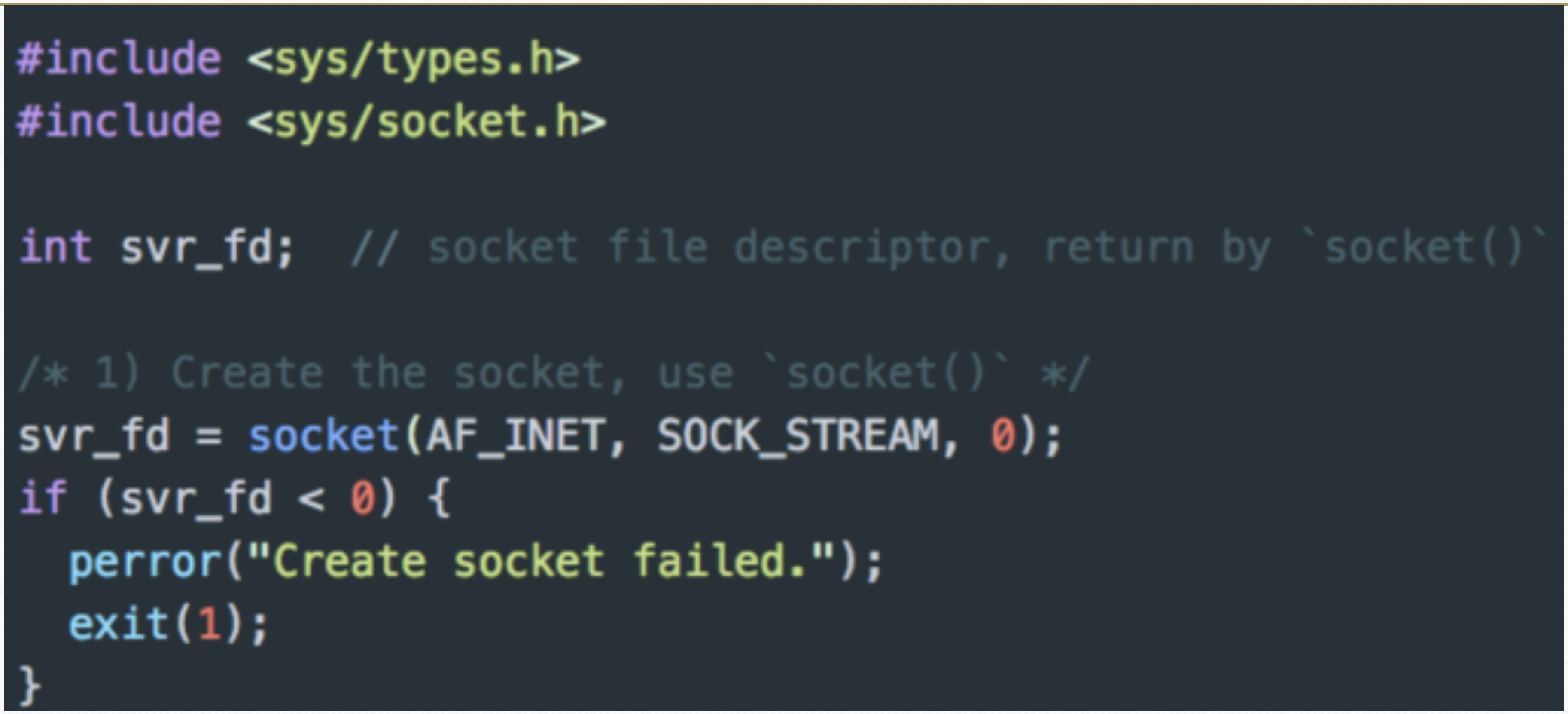
domain
    Specifies the communications domain in which a socket is to be created.
type
    Specifies the type of socket to be created.
protocol
    Specifies a particular protocol to be used with the socket. Specifying a *protocol* of 0 causes *socket()* to use an unspecified default protocol appropriate for the requested socket type.



RETURN VALUE

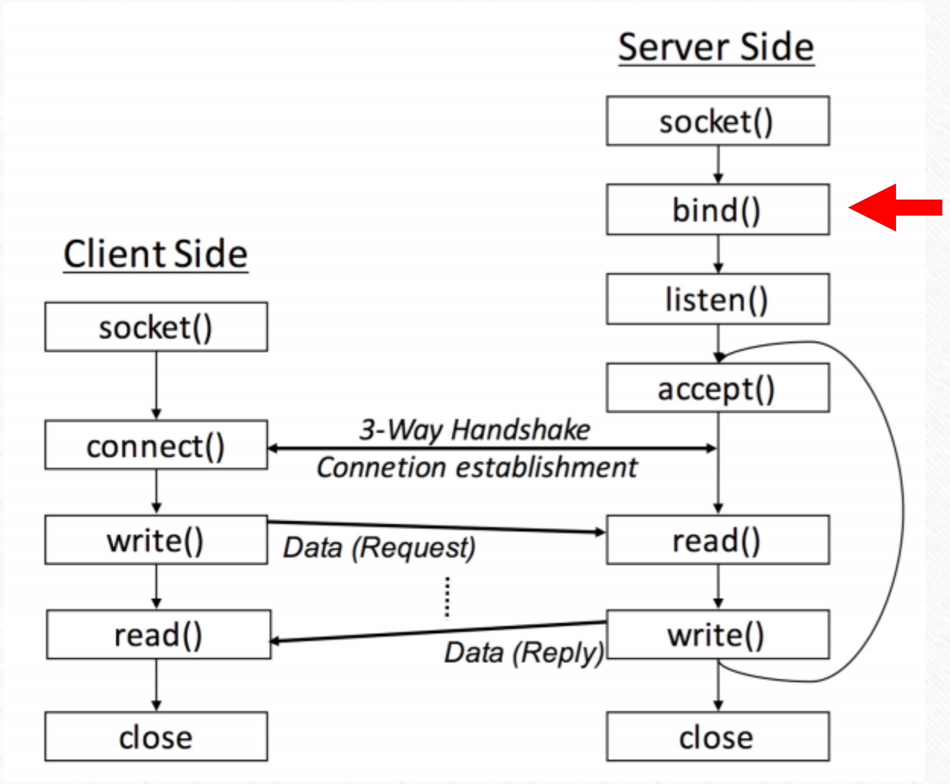
Upon successful completion, *socket()* returns a nonnegative integer, the socket file descriptor. Otherwise a value of –1 is returned and *errno* is set to indicate the error.

Socket( )

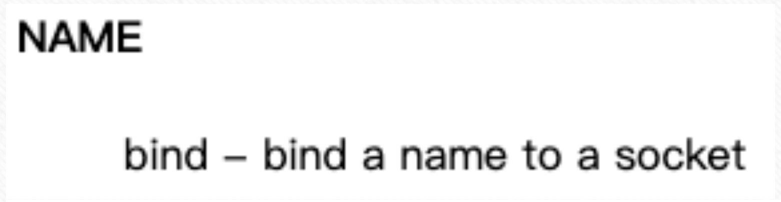


```
#include <sys/types.h>
#include <sys/socket.h>

int svr_fd;  // socket file descriptor, return by `socket()`

/* 1) Create the socket, use `socket()` */
svr_fd = socket(AF_INET, SOCK_STREAM, 0);
if (svr_fd < 0) {
  perror("Create socket failed.");
  exit(1);
}
```

Step2. Bind socket

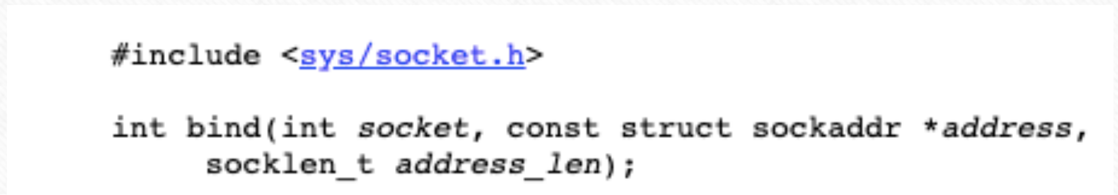


Bind( )

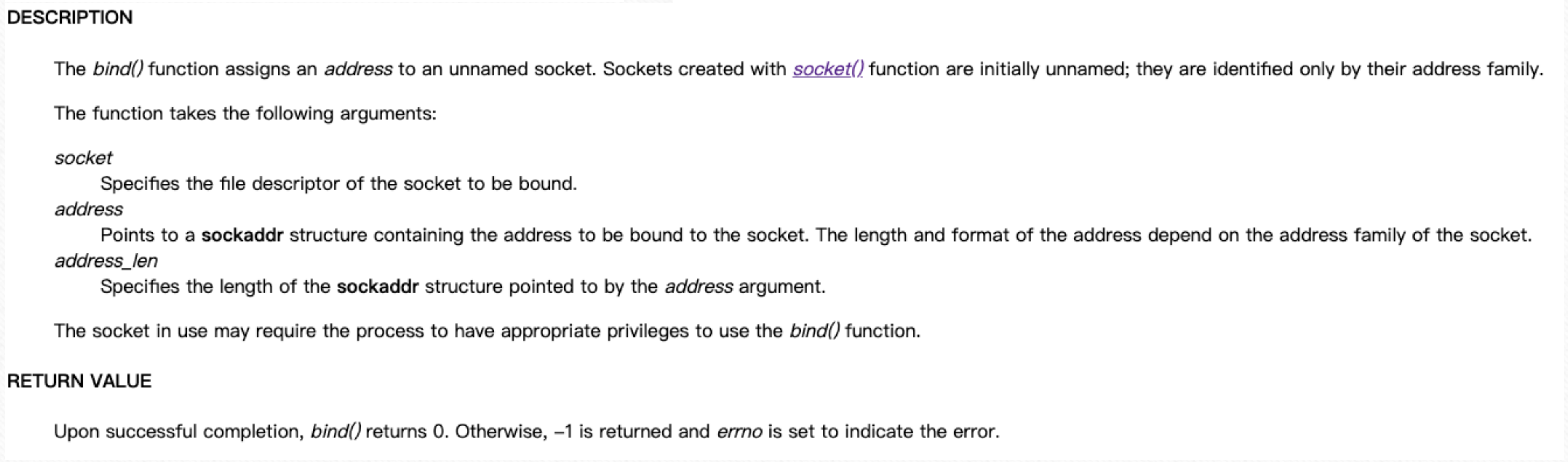


NAME

bind – bind a name to a socket



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

int bind(int socket, const struct sockaddr *address,
     socklen_t address_len);
```



DESCRIPTION

The *bind()* function assigns an *address* to an unnamed socket. Sockets created with *socket()* function are initially unnamed; they are identified only by their address family.

The function takes the following arguments:

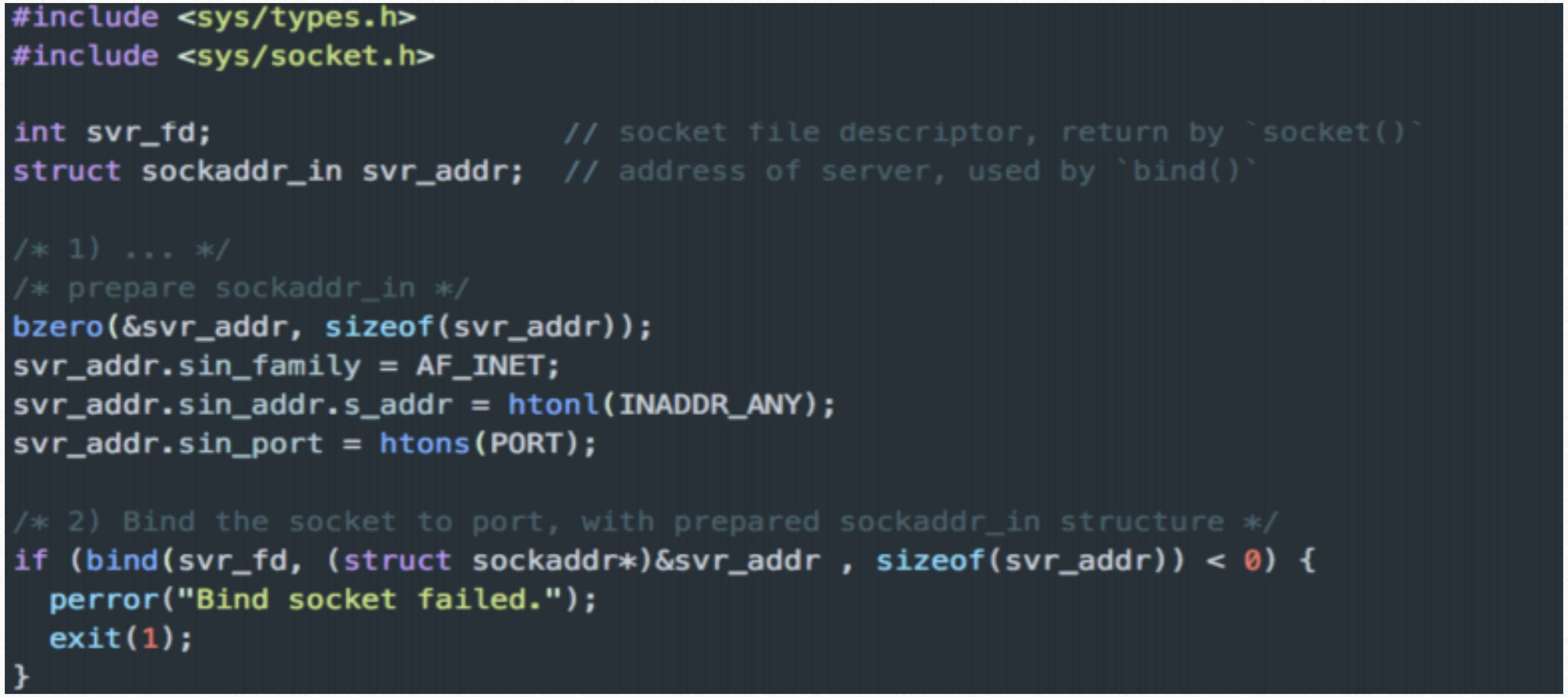
socket
Specifies the file descriptor of the socket to be bound.
address
Points to a **sockaddr** structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.
address_len
Specifies the length of the **sockaddr** structure pointed to by the *address* argument.

The socket in use may require the process to have appropriate privileges to use the *bind()* function.

RETURN VALUE

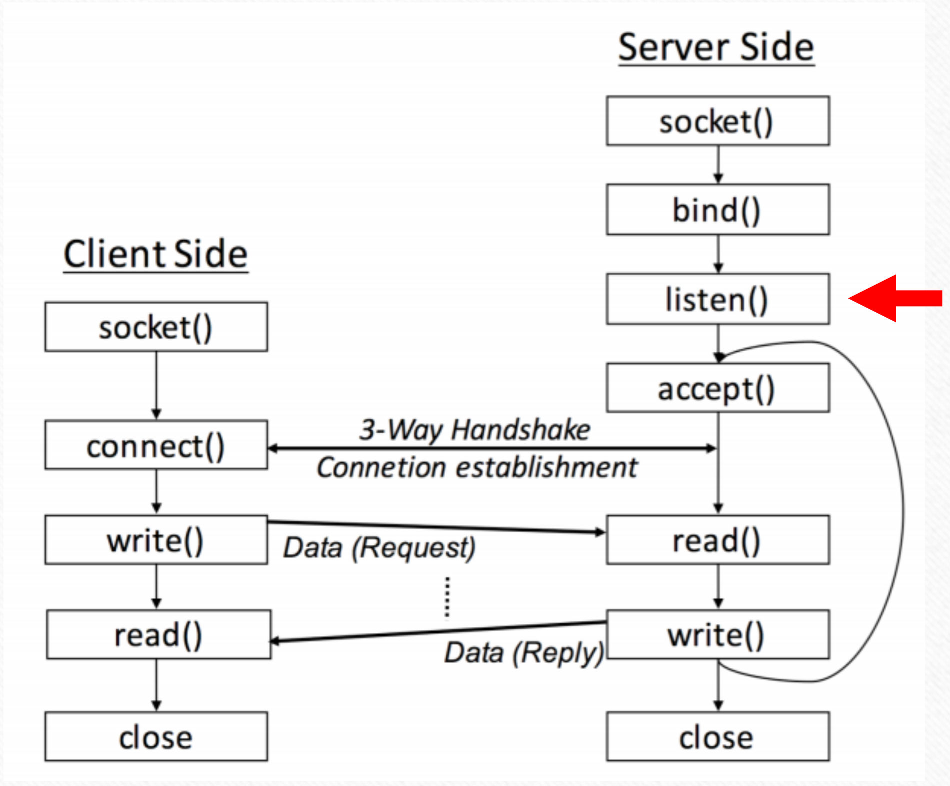
Upon successful completion, *bind()* returns 0. Otherwise, –1 is returned and *errno* is set to indicate the error.

Bind( )

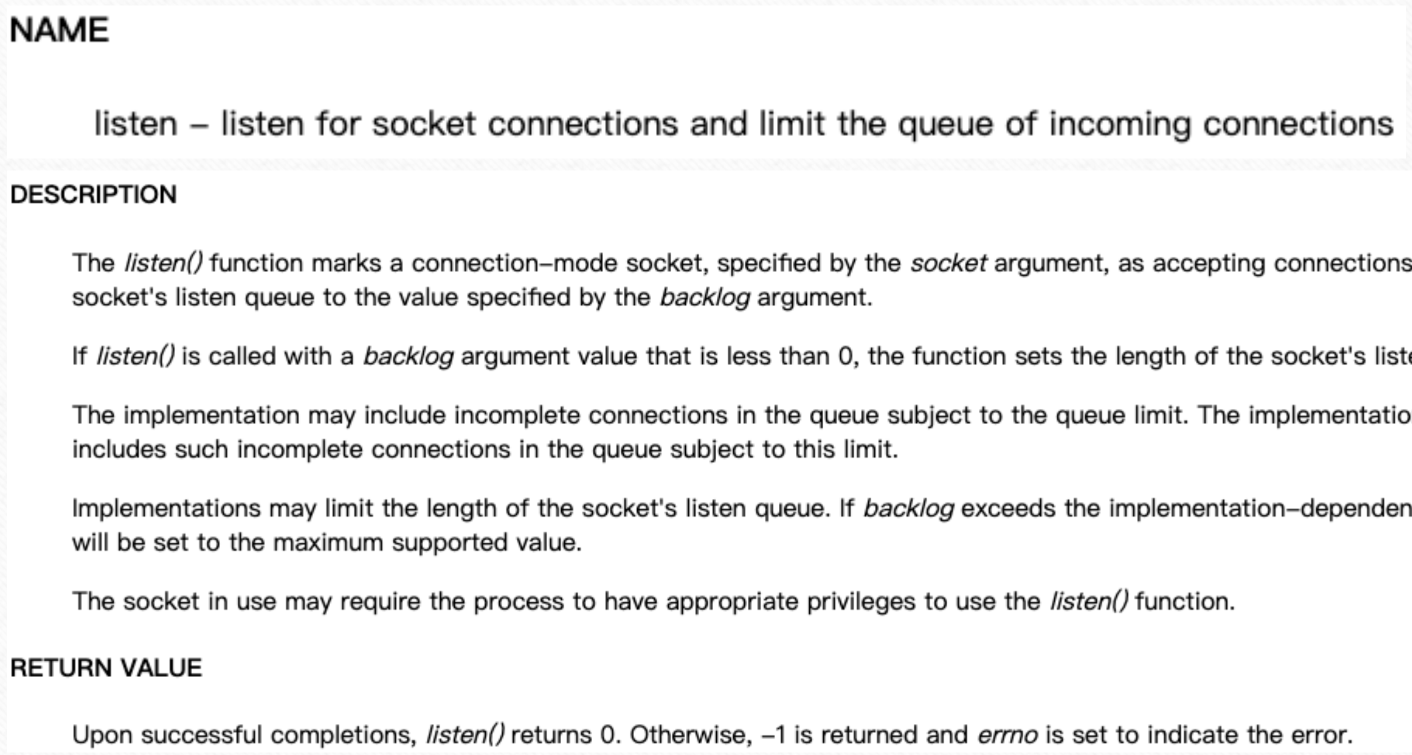


```c
#include <sys/types.h>
#include <sys/socket.h>

int svr_fd;                     // socket file descriptor, return by `socket()`
struct sockaddr_in svr_addr;  // address of server, used by `bind()`

/* 1) ... */
/* prepare sockaddr_in */
bzero(&svr_addr, sizeof(svr_addr));
svr_addr.sin_family = AF_INET;
svr_addr.sin_addr.s_addr = htonl(INADDR_ANY);
svr_addr.sin_port = htons(PORT);

/* 2) Bind the socket to port, with prepared sockaddr_in structure */
if (bind(svr_fd, (struct sockaddr*)&svr_addr , sizeof(svr_addr)) < 0) {
  perror("Bind socket failed.");
  exit(1);
}
```

Step3. Listen socket

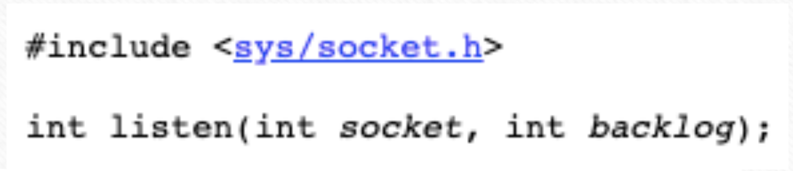


Listen( )



NAME

listen – listen for socket connections and limit the queue of incoming connections



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

int listen(int socket, int backlog);
```

DESCRIPTION

The *listen()* function marks a connection–mode socket, specified by the *socket* argument, as accepting connections, and limits the number of outstanding connections in the socket's listen queue to the value specified by the *backlog* argument.

If *listen()* is called with a *backlog* argument value that is less than 0, the function sets the length of the socket's listen queue to 0.

The implementation may include incomplete connections in the queue subject to the queue limit. The implementation may also increase the specified queue limit internally if it includes such incomplete connections in the queue subject to this limit.

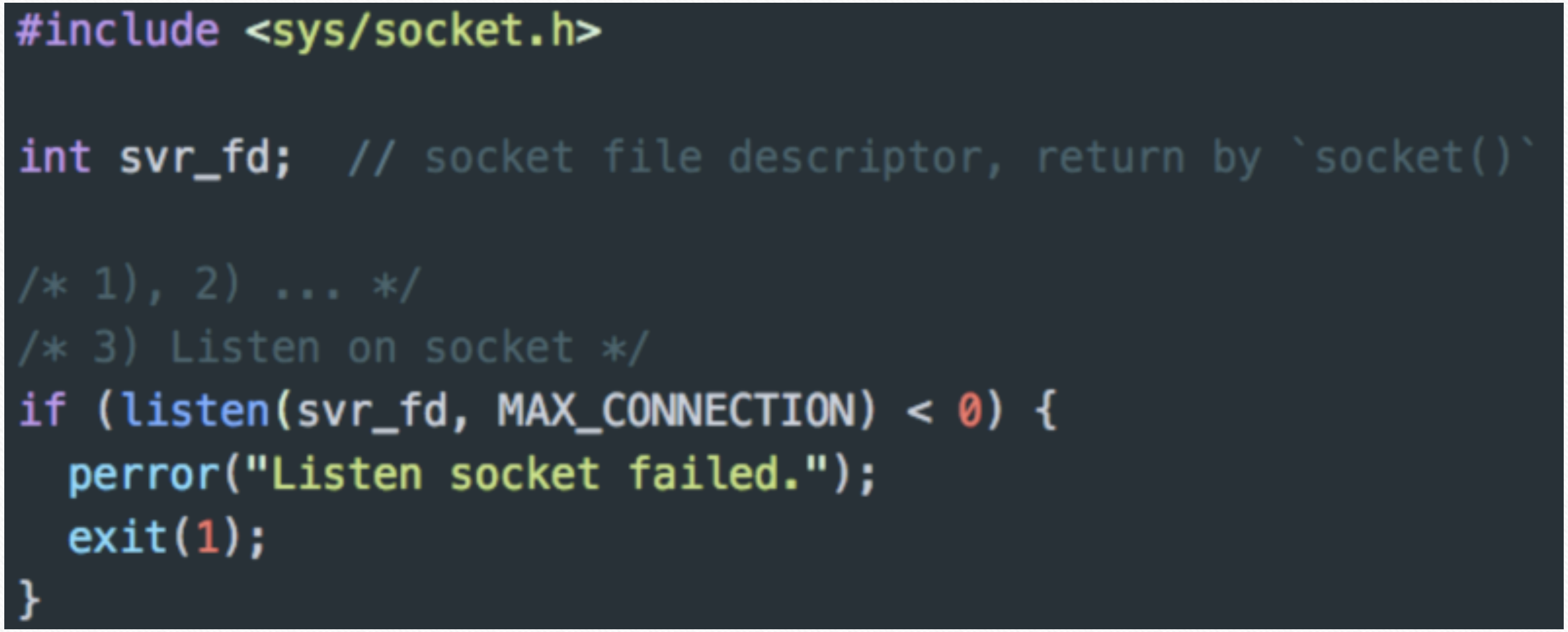
Implementations may limit the length of the socket's listen queue. If *backlog* exceeds the implementation–dependent maximum queue length, the length of the socket's listen queue will be set to the maximum supported value.

The socket in use may require the process to have appropriate privileges to use the *listen()* function.

RETURN VALUE

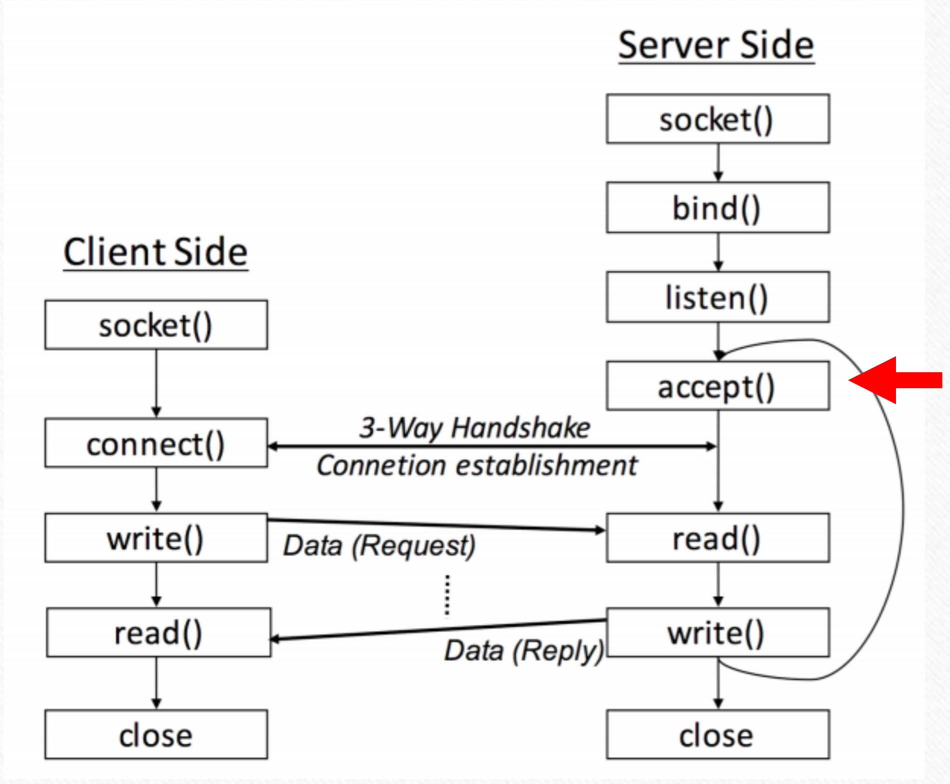
Upon successful completions, *listen()* returns 0. Otherwise, –1 is returned and *errno* is set to indicate the error.

Listen( )

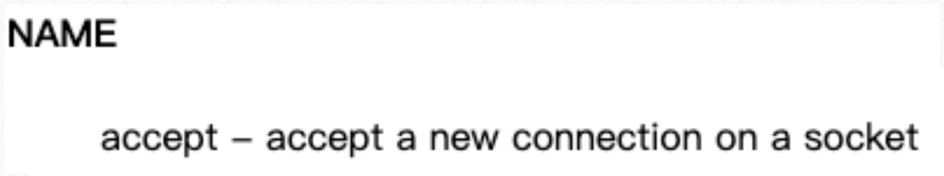


```c
#include <sys/socket.h>

int svr_fd;  // socket file descriptor, return by `socket()`

/* 1), 2) ... */
/* 3) Listen on socket */
if (listen(svr_fd, MAX_CONNECTION) < 0) {
  perror("Listen socket failed.");
  exit(1);
}
```

Step4. Accept socket

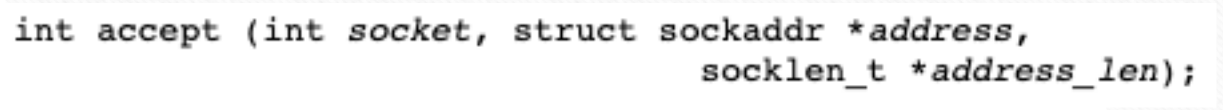


Accept( )

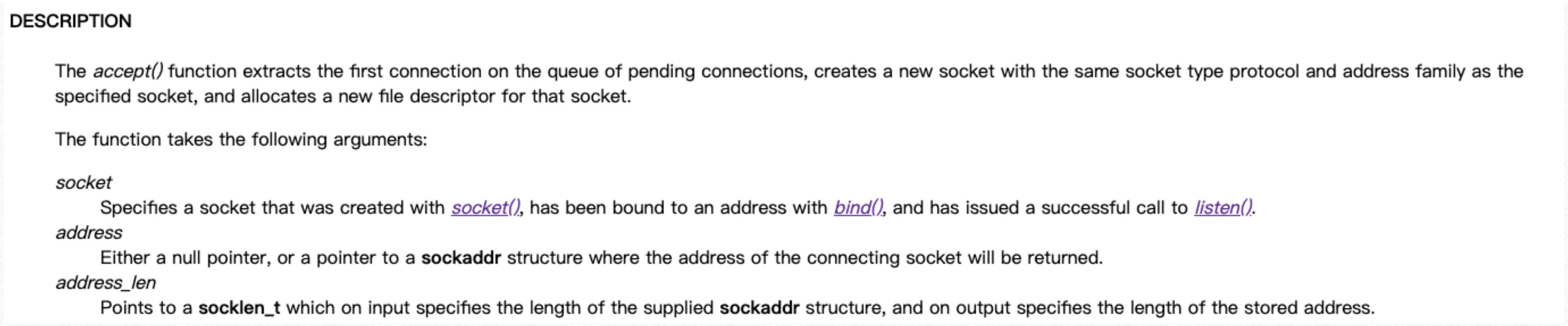


NAME

accept – accept a new connection on a socket



```
int accept (int socket, struct sockaddr *address,
                               socklen_t *address_len);
```



DESCRIPTION

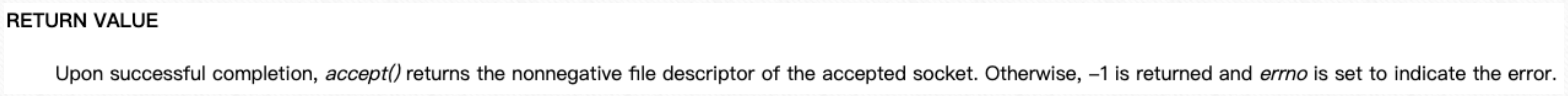
The *accept()* function extracts the first connection on the queue of pending connections, creates a new socket with the same socket type protocol and address family as the specified socket, and allocates a new file descriptor for that socket.

The function takes the following arguments:

socket
Specifies a socket that was created with *socket()*, has been bound to an address with *bind()*, and has issued a successful call to *listen()*.

address
Either a null pointer, or a pointer to a **sockaddr** structure where the address of the connecting socket will be returned.

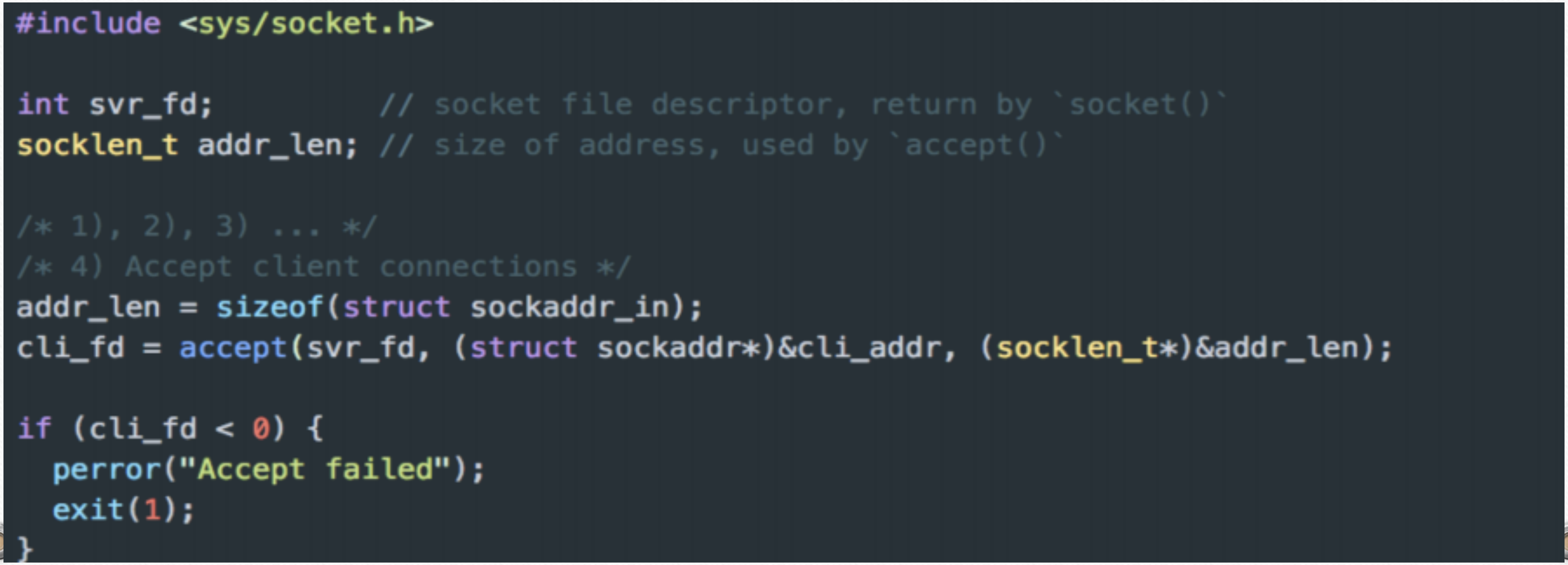
address_len
Points to a **socklen_t** which on input specifies the length of the supplied **sockaddr** structure, and on output specifies the length of the stored address.



RETURN VALUE

Upon successful completion, *accept()* returns the nonnegative file descriptor of the accepted socket. Otherwise, –1 is returned and *errno* is set to indicate the error.

Accept( )

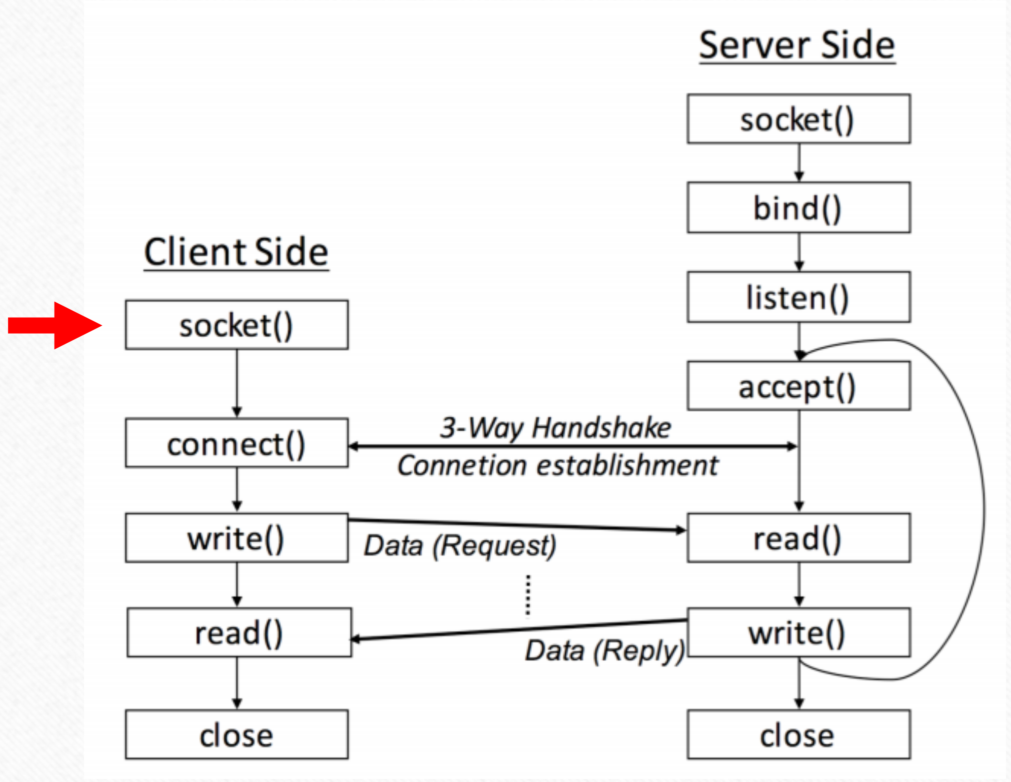


```c
#include <sys/socket.h>

int svr_fd;          // socket file descriptor, return by `socket()`
socklen_t addr_len; // size of address, used by `accept()`

/* 1), 2), 3) ... */
/* 4) Accept client connections */
addr_len = sizeof(struct sockaddr_in);
cli_fd = accept(svr_fd, (struct sockaddr*)&cli_addr, (socklen_t*)&addr_len);

if (cli_fd < 0) {
  perror("Accept failed");
  exit(1);
}
```

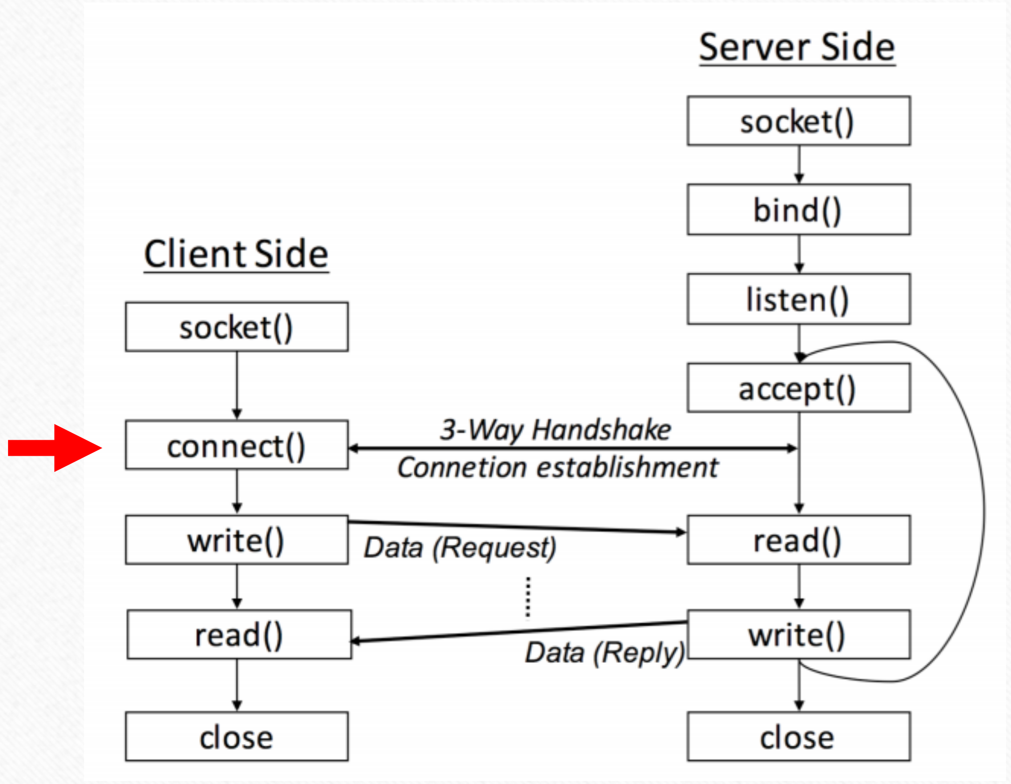
Socket Programming

Client 端

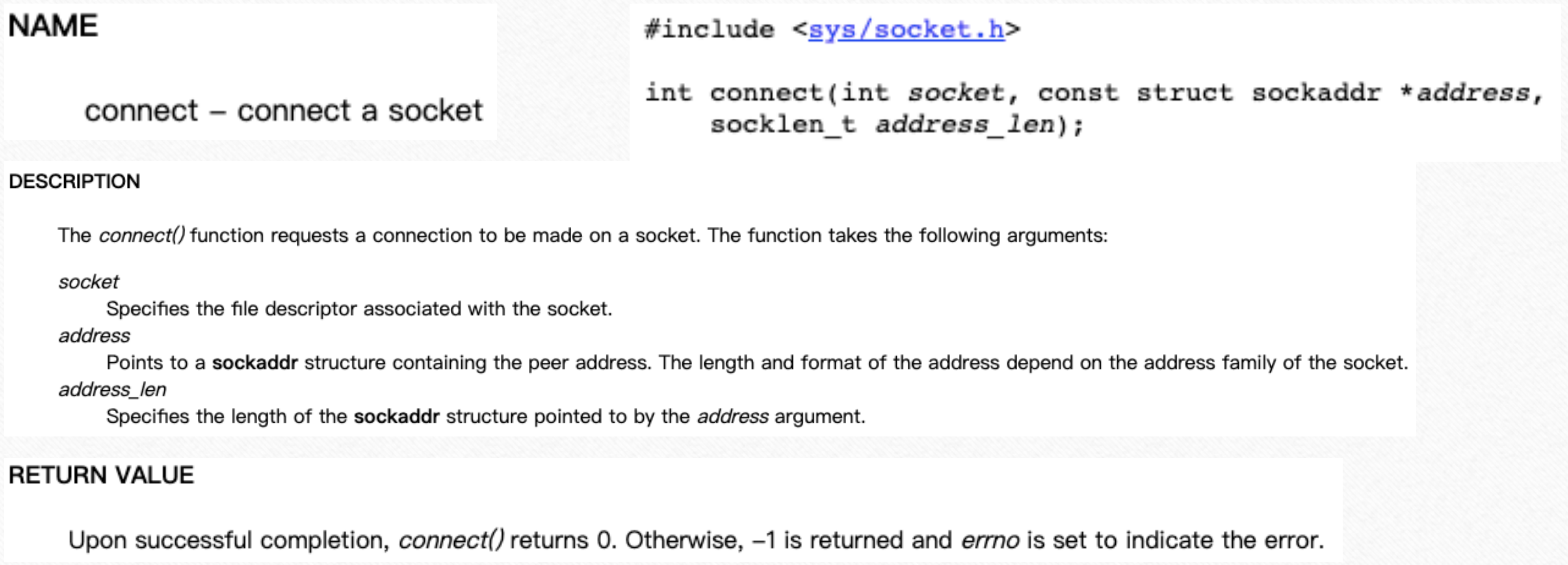
Step1. Create socket



Step2. Connect socket



Connect( )



NAME

connect – connect a socket

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

int connect(int socket, const struct sockaddr *address,
    socklen_t address_len);
```

DESCRIPTION

The *connect()* function requests a connection to be made on a socket. The function takes the following arguments:

socket
: Specifies the file descriptor associated with the socket.

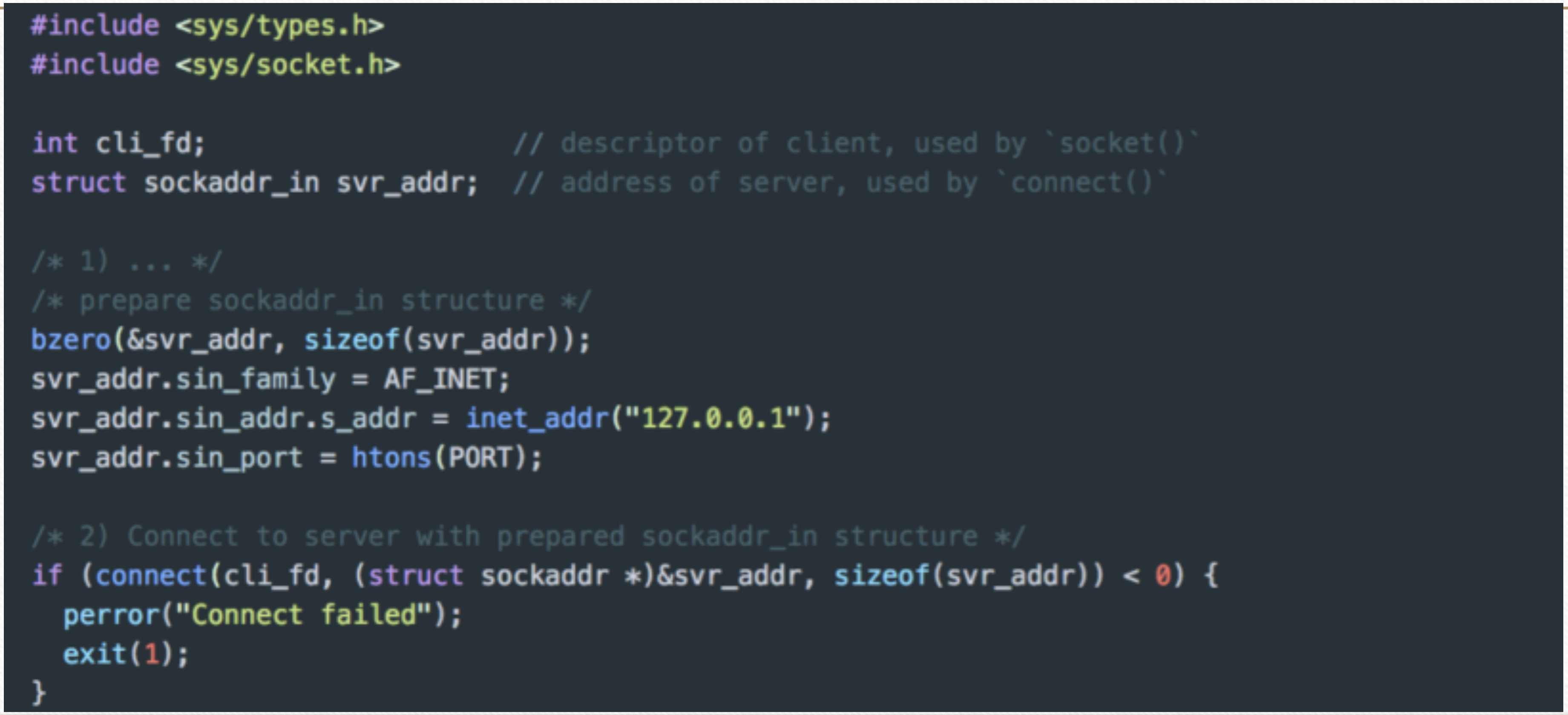
address
: Points to a **sockaddr** structure containing the peer address. The length and format of the address depend on the address family of the socket.

address_len
: Specifies the length of the **sockaddr** structure pointed to by the *address* argument.

RETURN VALUE

Upon successful completion, *connect()* returns 0. Otherwise, –1 is returned and *errno* is set to indicate the error.

Connect( )

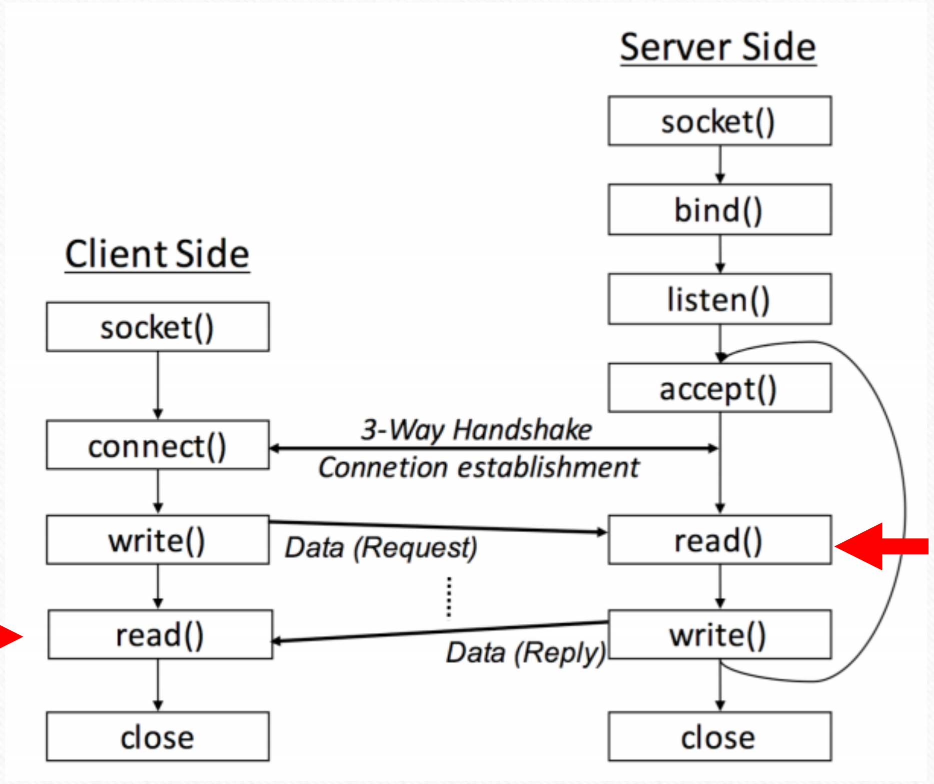


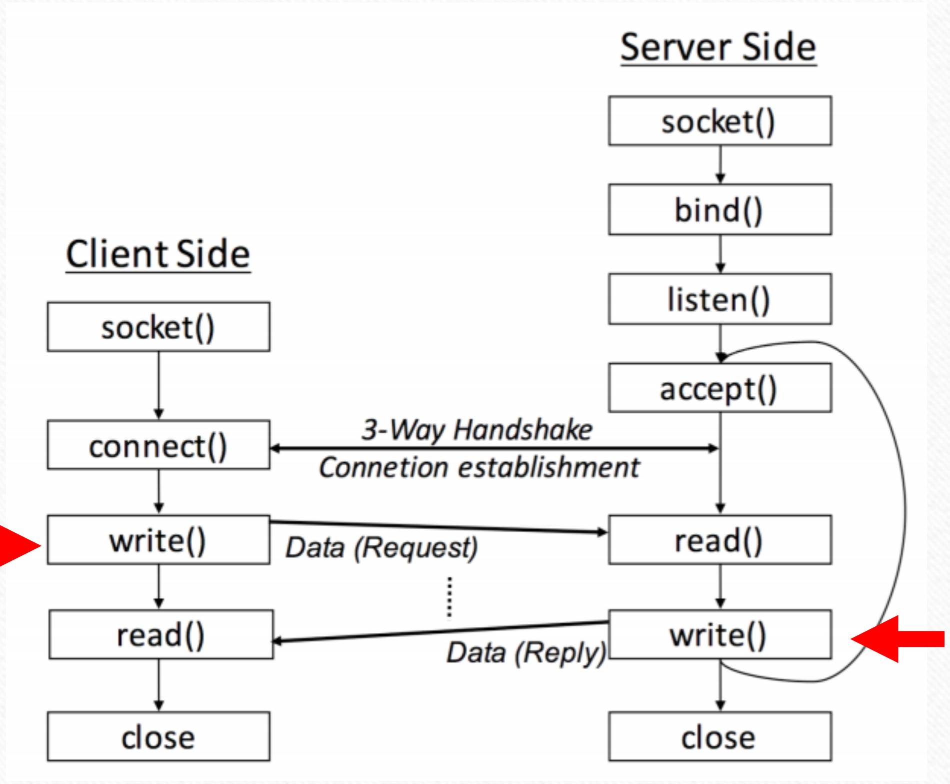
```
#include <sys/types.h>
#include <sys/socket.h>

int cli_fd;                      // descriptor of client, used by `socket()`
struct sockaddr_in svr_addr;  // address of server, used by `connect()`

/* 1) ... */
/* prepare sockaddr_in structure */
bzero(&svr_addr, sizeof(svr_addr));
svr_addr.sin_family = AF_INET;
svr_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
svr_addr.sin_port = htons(PORT);

/* 2) Connect to server with prepared sockaddr_in structure */
if (connect(cli_fd, (struct sockaddr *)&svr_addr, sizeof(svr_addr)) < 0) {
  perror("Connect failed");
  exit(1);
}
```

Socket Programming

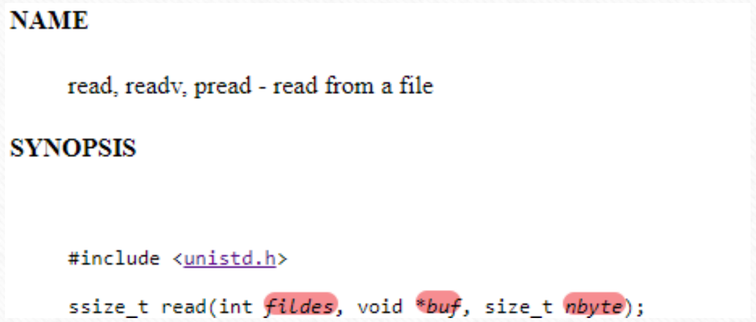
Read/Write

Read( ) / Write( )


Server Side
socket()
bind()
listen()
accept()
read()
write()
close
Client Side
socket()
connect()
write()
read()
close
3-Way Handshake
Connetion establishment
Data (Request)
Data (Reply)


Server Side
socket()
bind()
listen()
accept()
read()
write()
close
Client Side
socket()
connect()
write()
read()
close
3-Way Handshake
Connetion establishment
Data (Request)
Data (Reply)

Read( ) / Write( )

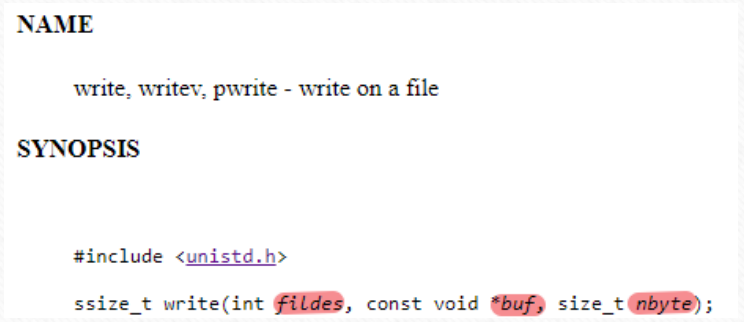


NAME

read, readv, pread - read from a file

SYNOPSIS

```
#include <unistd.h>

ssize_t read(int fildes, void *buf, size_t nbyte);
```

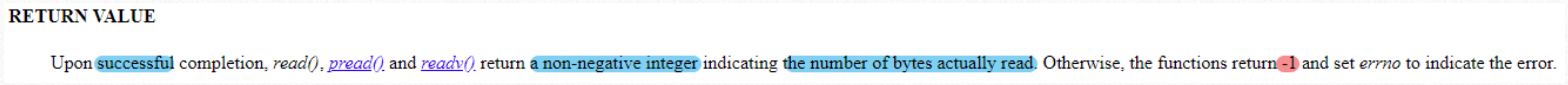


NAME

write, writev, pwrite - write on a file

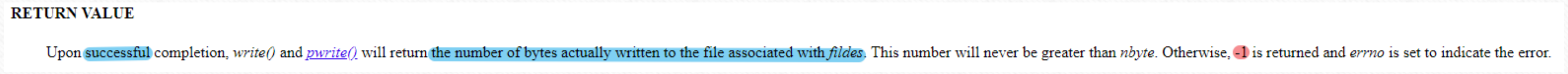
SYNOPSIS

```
#include <unistd.h>

ssize_t write(int fildes, const void *buf, size_t nbyte);
```



RETURN VALUE

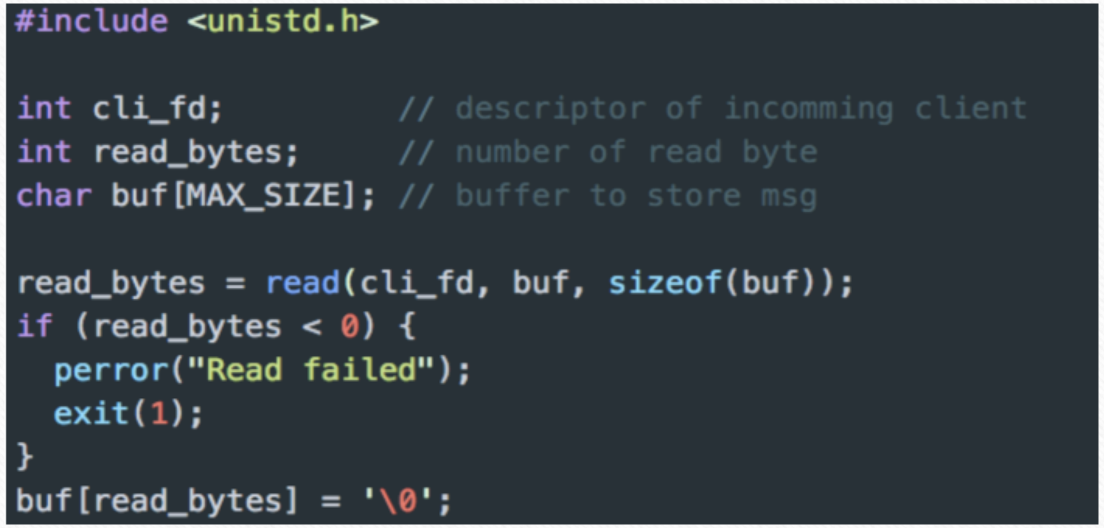
Upon successful completion, *read()*, *pread()* and *readv()* return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return -1 and set *errno* to indicate the error.



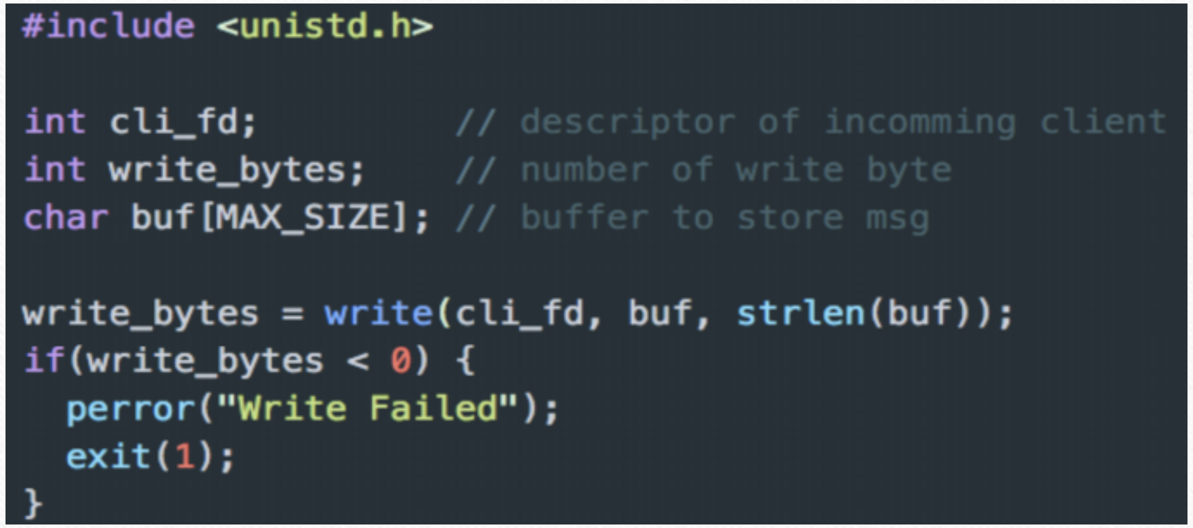
RETURN VALUE

Upon successful completion, *write()* and *pwrite()* will return the number of bytes actually written to the file associated with *fildes*. This number will never be greater than *nbyte*. Otherwise, -1 is returned and *errno* is set to indicate the error.

Read( ) / Write( )



```c
#include <unistd.h>

int cli_fd;           // descriptor of incomming client
int read_bytes;       // number of read byte
char buf[MAX_SIZE]; // buffer to store msg

read_bytes = read(cli_fd, buf, sizeof(buf));
if (read_bytes < 0) {
  perror("Read failed");
  exit(1);
}
buf[read_bytes] = '\0';
```

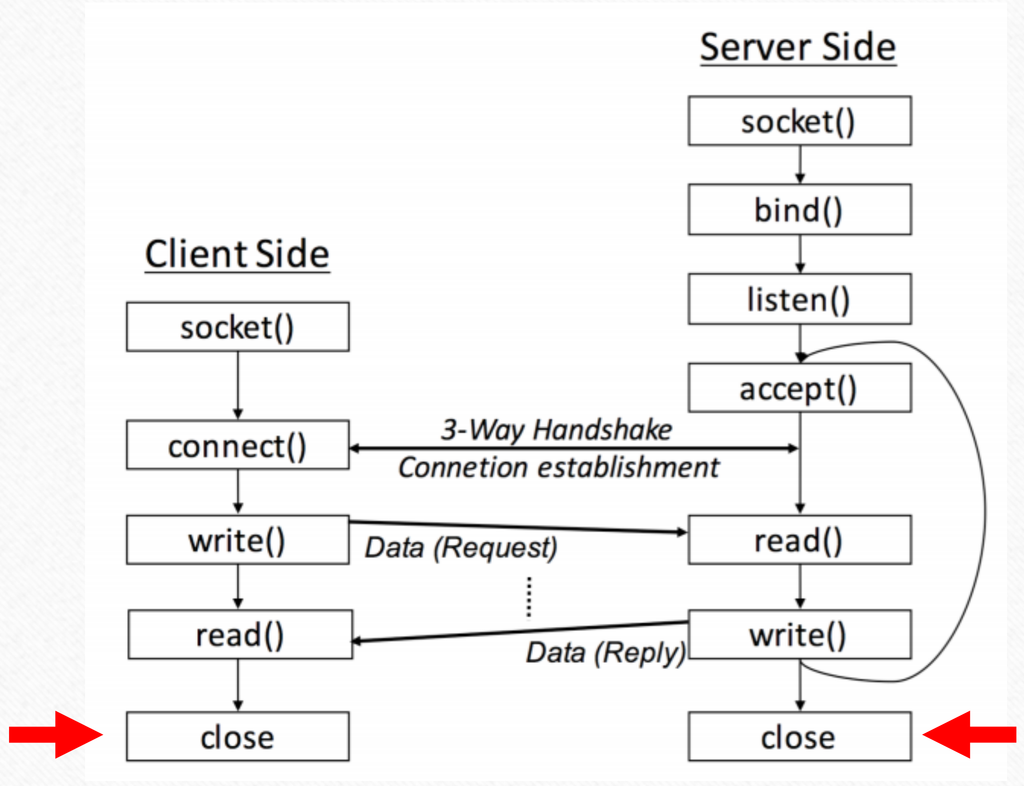


```c
#include <unistd.h>

int cli_fd;          // descriptor of incomming client
int write_bytes;     // number of write byte
char buf[MAX_SIZE]; // buffer to store msg

write_bytes = write(cli_fd, buf, strlen(buf));
if(write_bytes < 0) {
  perror("Write Failed");
  exit(1);
}
```

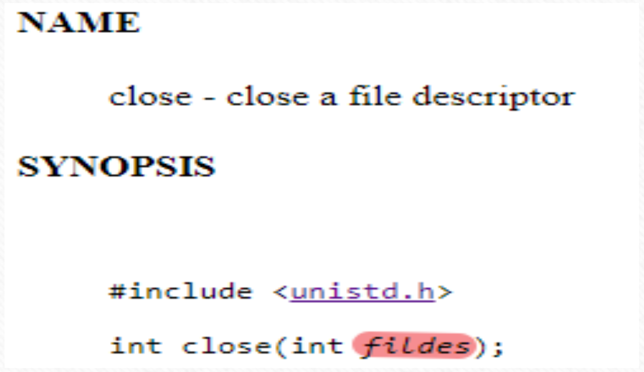
Socket Programming

Close( )

Close( )


Server Side
socket()
bind()
listen()
accept()
Client Side
socket()
connect()
3-Way Handshake
Connetion establishment
write()
Data (Request)
read()
read()
write()
Data (Reply)
close
close

Close( )

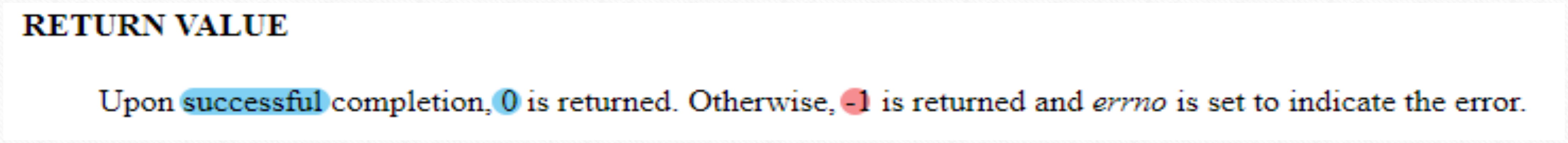


NAME

close - close a file descriptor

SYNOPSIS

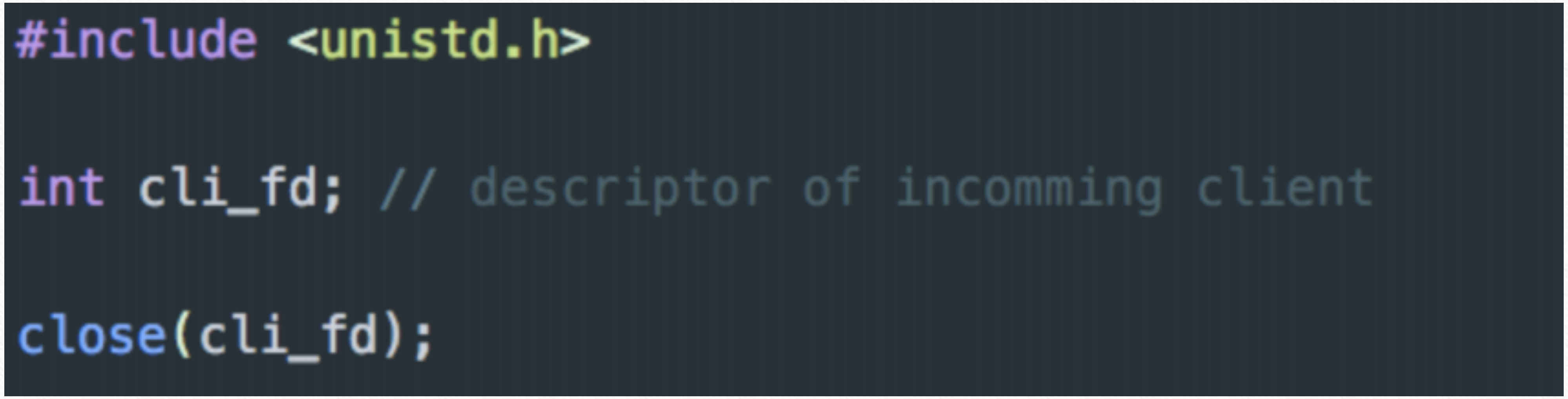
```
#include <unistd.h>

int close(int fildes);
```



RETURN VALUE

Upon successful completion, 0 is returned. Otherwise, -1 is returned and *errno* is set to indicate the error.

Close( )



```
#include <unistd.h>

int cli_fd; // descriptor of incomming client

close(cli_fd);
```

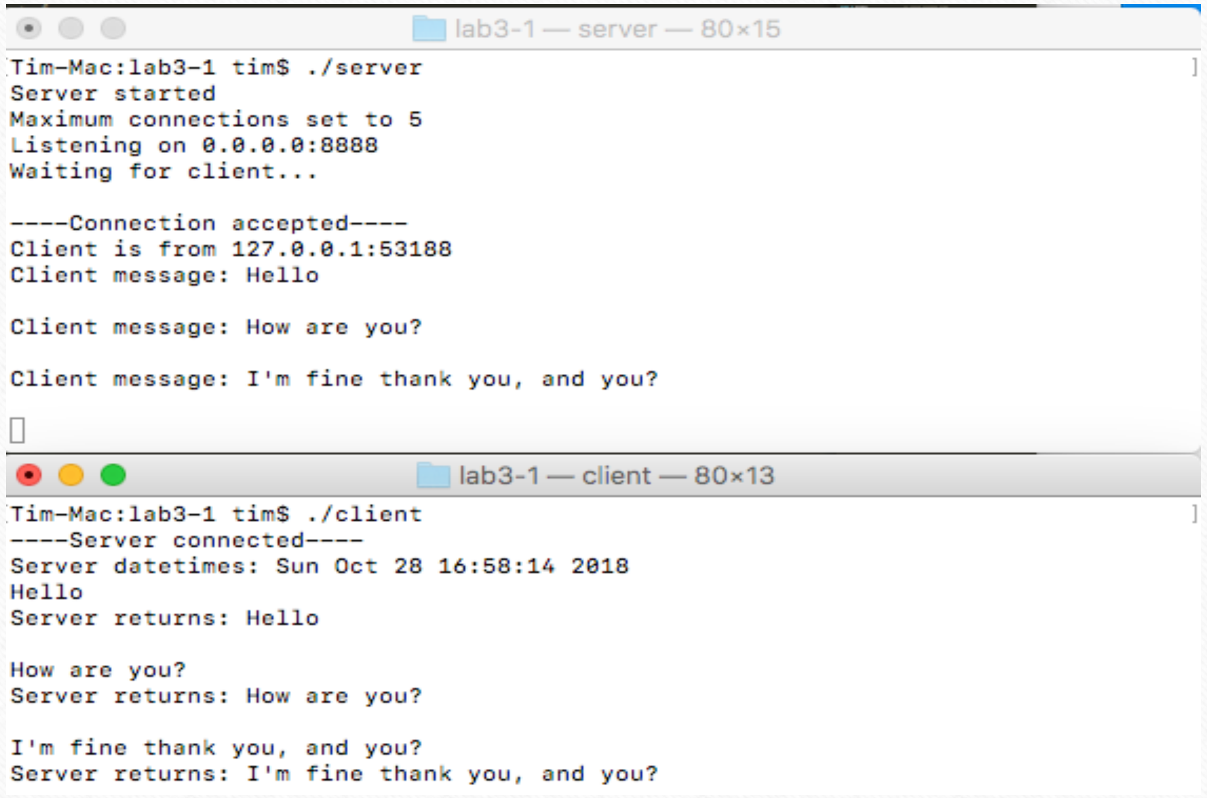
Lab 3 – Echo Server

Mission

- 提供: 能夠進行”單次“傳送訊息的 server & client
  - server 會回傳當前時間給 clinet

- 目標: 能夠進行“多次”傳送訊息的 server & client
  - server 會回傳 client 傳送的訊息
  - 截圖中，其中一則訊息為學號

Mission



```
Tim-Mac:lab3-1 tim$ ./server
Server started
Maximum connections set to 5
Listening on 0.0.0.0:8888
Waiting for client...

----Connection accepted----
Client is from 127.0.0.1:53188
Client message: Hello

Client message: How are you?

Client message: I'm fine thank you, and you?

```

```
Tim-Mac:lab3-1 tim$ ./client
----Server connected----
Server datetimes: Sun Oct 28 16:58:14 2018
Hello
Server returns: Hello

How are you?
Server returns: How are you?

I'm fine thank you, and you?
Server returns: I'm fine thank you, and you?
```

作業繳交

- 截止日期: **10/27(日) 23:59** 前
- 檔案名稱：**<學號>_lab3.zip** (ex: 108xxxxxx_lab3.zip)，**請不要壓縮成.rar**
- 資料夾內容包括：
  - server.c
  - client.c
  - 執行結果截圖（其中一則訊息為學號）
- 扣分項目：
  - 檔名錯誤：**扣10分**
  - 抄襲以零分計算
  - 遲交：遲交一週分數**打8折**、兩週**打6折**，以此類推