操作系统实验 4：远程进程 socket 通信

王靖康
515030910059
网络空间安全学院
wangjksjtu_01@sjtu.edu.cn

1 实验题目

分别编一个客户端程序和服务器程序，首先建立客户程序与服务器之间正确的 socket 连结，然后利用 send 和 recv 函数，客户程序将一个较长的文本文件（如几 k 字节）中的数据发送给服务器。要求服务器全部正确地接收到所有的数据，并将其存入一个文件。

2 实验目的

- 理解套接字通信的基本原理和机制;
- 提高用 C 语言编制程序的能力，熟悉标准库函数 API 接口;
- 深入掌握 C 语言套接字编程的方法和细节。

3 Socket 通信

3.1 进程通信

进程间通信就是在不同进程之间传播或交换信息。IPC 是一组编程接口，使得进程可以实现协作，包括数据传输、共享数据、通知事件、资源共享、进程控制等功能，从而完成用户多样化的要求。常见的进程间通信方法包括:1) 管道通信 (PIPE); 2) 消息通信; 3) 共享存储区; 4) 套接字 (Socket) 通信。

本次实验使用 C 语言在类 Unix 系统完成 socket 通信功能，实现不同程序间的数据传输。

3.2 Socket 通信原理

两个程序通过网络的一个双向通信连接完成数据的交换，该连接的每端称为一个 socket(IP 地址 + 端口)。Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，是一组接口，为应用层提供连接服务。

Socket 的通信流程如图 1 所示。服务端和客户端分别创建 socket，在服务端完成端口绑定（bind），网络监听（listen）和接受请求（accept）后, 客户端可以连接服务端（connect），从而建立通信。建立连接后，客户端和服务端程序可以通过读或写操作完成进程通信。最后，通信完成后，双方可关闭 socket 通信（close）。

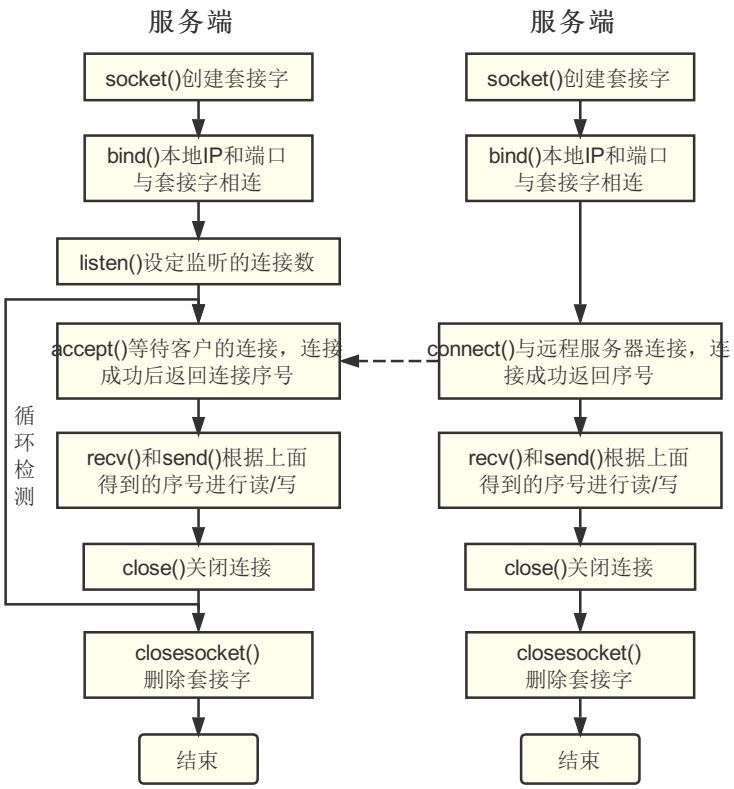

Figure 1: Socket 通信流程图 (左: 服务端，右: 客户端)

3.3 编程接口介绍

Socket 是 “open—write/read—close” 模式的一种实现，提供了这些操作对应的函数接口。下面以 TCP 为例，介绍几个基本的 socket 接口函数。

socket() 函数: int socket(int domain, int type, int protocol); 该函数用于创建一个 socket 描述符（socket descriptor），唯一标识一个 socket。该描述符跟文件描述符一样，通过它进行读写操作。其中，domain 为协议域或协议族（family）。常用的协议族有，AF_INET、AF_INET6、AF_LOCAL 等等。协议族决定了 socket 的地址类型，在通信中必须采用对应的地址。本次实验使用 AF_INET。

bind() 函数: int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen); 其中，sockfd 为描述符，addr 为要绑定的协议地址，addrlen 为对应的地址的长度。通常服务器在启动的时候都会绑定一个地址（ip 地址 + 端口号），用于提供服务，客户就可以通过它来接连服务器；而客户端使用系统自动分配一个端口号和自身的 ip 地址。因此通常服务器端在 listen 之前会调用 bind()，而客户端是在 connect() 时由系统随机生成端口。

listen/connect() 函数: int listen(int sockfd, int backlog); int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen); 其中，sockfd 为描述符，backlog 表示可以排队的最大连接个数，sockaddr 为服务器的 socket 地址，addrlen 表示 socket 地址的长度。

accept() 函数: int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen); 参数意义同上。服务器端调用 socket()、bind()、listen() 后，监听指定的 socket 地址。客户端调

用 socket()、connect() 之后向服务器发送一个连接请求。服务器监听到这个请求之后，就会调用 accept() 函数取接收请求，返回一个新的描述符，代表双方的 TCP 连接。

close() 函数: int close(int fd); 关闭 socket 连接，fd 表示建立的 socket 描述符。值得注意的是，close 操作只是使相应 socket 描述字的引用计数-1，当引用计数为 0 的时候，才会触发 TCP 客户端向服务器发送终止连接请求。

4 编程实现

4.1 数据结构

本次实验采用 AF_INET 协议族，因此需要绑定的 sockfd 协议地址为:

```
struct sockaddr_in {
    sa_family_t    sin_family;  // 协议族
    in_port_t      sin_port;    // 绑定端口
    struct in_addr sin_addr;    // 监听地址
};
```

实验用到的变量或常量定义为:

```
#define BUFFER_SIZE 4096     // 缓存区大小
#define IP_ADDR "127.0.0.1" // 服务端地址
#define SERVER_PORT 8888     // socket端口

char buffer [BUFFER_SIZE];  // 缓存区，用于传输文件数据
char message [BUFFER_SIZE]; // 缓存区，用于传输文件名

int flag = 1;               // 服务端文件存在
```

4.2 模块与接口设计

本次实验分客户端和服务端两部分，基本流程相同，均为先配置 socket，按照编程接口章节介绍和图 1 所示。之后建立 socket 连接，进行数据通信，完成文件传输功能。最后使用 close 断开 socket 连接。值得注意的是，由于服务端可能与多个客户端建立连接，故需要为每个客户端均生成一个线程，完成与该客户端的 socket 通信。另外，本实现支持了一下功能:

- 服务端不中断，可同时与多个客户端建立 socket 连接，进行文件传输。
- 用户端建立一次连接期间可进行多次文件传输。
- 程序异常处理，包括 socket 建立、socket 连接，数据通信，文件不存在等异常的处理，并实现了客户端和服务端的针对不同情况正确应对。

下面分别对程序中的一些重要代码进行说明。

1. 建立 socket 连接

服务端和客户端的建立 socket 连接的过程不完全相同，详细过程见接口函数介绍部分和图 1, 具体 C 语言实现如图 2 和图 3 所示。

```c
    int socket_desc , new_socket , c , *new_sock;
    struct sockaddr_in server , client;
    char *message;

    // Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1) {
        printf("[Error]: Could not create socket");
    }

    // Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);

    // Bind
    if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0) {
        puts("[Error]: Bind failed");
        return 1;
    }
    puts("[Info]: Bind done");

    // Listen
    listen(socket_desc , 3);

    // Accept and incoming connection
    puts("[Info]: Waiting for incoming connections...");
```

Figure 2: 服务端 socket 建立过程

```c
int main(int argc , char *argv[]) {
    int socket_desc;
    struct sockaddr_in server;
    char buffer[BUFFER_SIZE];
    char message[512], filename[512];

    // Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1) {
        printf("Could not create socket");
    }

    server.sin_addr.s_addr = inet_addr(IP_ADDR);
    server.sin_family = AF_INET;
    server.sin_port = htons(SERVER_PORT);

    // Connect to remote server
    if (connect(socket_desc , (struct sockaddr *)&server , sizeof(server)) < 0) {
        puts("[Error]: Connection error!");
        return 1;
    }

    puts("[Info]: Connected to server");
```

Figure 3: 客户端 socket 建立过程

2. socket 传输文件数据

服务端在收到客户端想要传输的文件名后，在本地进行文件查询。若未找到文件，像客户端返回未找到标识，客户端可重新请求文件。若找到文件，则按缓冲区大小分次传输数据，客户端同样分次接受后将所有数据保存到本地文件中（如图 4, 5 所示）。另外，程序通过两个 while 循环嵌套实现了同一个 socket 连接可多次分片传输多个文件的功能，使得程序更具有拓展性。

```
while ((read_size = recv(sock, client_message, BUFFER_SIZE, 0)) > 0) {
    // Send the message back to client
    fflush(stdout);
    FILE *file_pointer = fopen(client_message, "r");
    if (file_pointer == NULL) {
        printf("[Error]: File: %s not found!\n", client_message);
        bzero(buffer, BUFFER_SIZE);
        strcpy(buffer, "!=!");
        // printf("%s", buffer);
        // fflush(stdout);
        send(sock, buffer, sizeof(buffer), 0);
    }
    else  {
        bzero(buffer, BUFFER_SIZE);
        int file_block_length = 0;
        while((file_block_length = fread(buffer, sizeof(char), BUFFER_SIZE,
        file_pointer)) > 0) {
            printf("[Info]: File block length = %d\n", file_block_length);
            if (send(sock, buffer, file_block_length, 0) < 0) {
                printf("[Info]: Send file %s failed!\n", client_message);
                break;
            }
            bzero(buffer, sizeof(buffer));
        }
        printf("[Info]: File %s transferring finished!\n", client_message);
        fclose(file_pointer);
    }
}
```

Figure 4: 服务端文件传输过程

3. 服务端多线程处理请求

由于服务端需要同时为多个客户端提供服务，在本程序中通过多个线程保持服务端与多个客户端的 socket 连接，使得多个连接相互独立，互不干扰（如图 6 所示，通过 pthread 库实现）。

5 功能测试

本部分包含了客户端和服务端的功能测试，以及异常情况的测试。如图 7、8 和 9 所示，1）服务端可以不间断运行，并与多个客户端可以建立 socket 通信。2）服务端和客户端能够很好地对异常情况进行处理，程序不会崩溃。3）服务端分多次传输一个文件，客户端分多次存储一个文件，传输的文件与源文件完全相同。

```
while (1) {
    length = recv(socket_desc, buffer, BUFFER_SIZE, 0);
    if (length < 0) {
        printf("[Error]: Failed to receive file `%s` from server\n",
        message);
    }
    else {
        // printf("%s", buffer);
        if (buffer[0] == '!' && buffer[1] == '=' && buffer[2] == '!') {
            printf("[Error]: File `%s` not found in server\n", message);
            flag = 0;
            break;
        }
        else {
            // printf("%s", buffer);
            int write_length = fwrite(buffer, sizeof(char), length,
            file_pointer);
            if (write_length < length) {
                printf("[Error]: Failed to write to %s\n", filename);
                break;
            }
            if (length < BUFFER_SIZE) break;
        }
    }
}
if (flag) printf("[Info]: Save to file %s finished!\n", filename);
fclose(file_pointer);
```

Figure 5: 客户端文件传输过程

```
while ((new_socket = accept(socket_desc, (struct sockaddr *)&client,
(socklen_t*)&c))) {
    puts("[Info]: Connection accepted");

    // Reply to the client
    message = "Hello, nice to meet you!\n";
    write(new_socket , message , strlen(message));

    pthread_t sniffer_thread;
    new_sock = malloc(1);
    *new_sock = new_socket;

    if (pthread_create( &sniffer_thread, NULL, connection_handler, (void*)
    new_sock) < 0) {
        perror("[Error]: Could not create thread");
        return 1;
    }

    // Now join the thread , so that we dont terminate before the thread
    // pthread_join( sniffer_thread , NULL);
    puts("[Info]: Handler assigned");
}
```

Figure 6: 服务端为 socket 连接创建线程

```
wangjk@asus-wjk:~/programs/OS/lab4$ ls -al
total 60
drwxrwxr-x 2 wangjk wangjk  4096 May 27 19:12 .
drwxrwxr-x 9 wangjk wangjk  4096 May 26 21:24 ..
-rwxrwxr-x 1 wangjk wangjk 13480 May 27 19:11 client
-rw-rw-r-- 1 wangjk wangjk  3157 May 27 19:12 client.c
-rwxrwxr-x 1 wangjk wangjk 13832 May 27 18:58 server
-rw-rw-r-- 1 wangjk wangjk  3793 May 27 18:57 server.c
-rw-rw-r-- 1 wangjk wangjk 11473 May 26 21:23 test.in
```

Figure 7: 初始代码目录情况

```
wangjk@asus-wjk:~/programs/OS/lab4$ ./client
[Info]: Connected to server
[Info]: Hello, nice to meet you!
[Info]: Path of file to be read: test.in
[Info]: Send message to server
[Info]: Path of file to be saved: test.out
[Info]: Save to file test.out finished!
[Info]: Path of file to be read: test.123
[Info]: Send message to server
[Info]: Path of file to be saved: test.234
[Error]: File `test.123` not found in server
[Info]: Path of file to be read: client.c
[Info]: Send message to server
[Info]: Path of file to be saved: test.client
[Info]: Save to file test.client finished!
[Info]: Path of file to be read: ^C
wangjk@asus-wjk:~/programs/OS/lab4$
```

```
wangjk@asus-wjk:~/programs/OS/lab4$ ./server
[Info]: Bind done
[Info]: Waiting for incoming connections...
[Info]: Connection accepted
[Info]: Handler assigned
[Info]: File block length = 4096
minal ]: File block length = 4096
[Info]: File block length = 3281
[Info]: File test.in transferring finished!
[Error]: File: test.123 not found!
[Info]: File block length = 3157
[Info]: File client.c transferring finished!
[Info]: Client disconnected
```

Figure 8: 客户端和服务端日志

```
wangjk@asus-wjk:~/programs/OS/lab4$ ls -al
total 76
drwxrwxr-x 2 wangjk wangjk  4096 May 27 19:14 .
drwxrwxr-x 9 wangjk wangjk  4096 May 26 21:24 ..
-rwxrwxr-x 1 wangjk wangjk 13480 May 27 19:11 client
-rw-rw-r-- 1 wangjk wangjk  3157 May 27 19:12 client.c
-rwxrwxr-x 1 wangjk wangjk 13832 May 27 18:58 server
-rw-rw-r-- 1 wangjk wangjk  3793 May 27 18:57 server.c
-rw-rw-r-- 1 wangjk wangjk     0 May 27 19:14 test.234
-rw-rw-r-- 1 wangjk wangjk  3157 May 27 19:14 test.client
-rw-rw-r-- 1 wangjk wangjk 11473 May 26 21:23 test.in
-rw-rw-r-- 1 wangjk wangjk 11473 May 27 19:14 test.out
wangjk@asus-wjk:~/programs/OS/lab4$ diff test.in test.out
wangjk@asus-wjk:~/programs/OS/lab4$ diff client.c test.client
```

Figure 9: 文件传输完成

6 总结与思考

6.1 错误分析

Socket 通信编程中经常会遇到一些难以定位的错误，这是以下几个原因造成的: 1）Socket 通信中错误的编程容易导致双方均发生阻塞，导致无法定位。2）在阻塞状态缓冲区没有刷新，导致利用输出定位错误变得较为困难。3）交互异常情况较为复杂，容易发生一些边界情况判断失误，造成死锁。4）由于逻辑问题造成死循环，导致程序无法正确退出。

针对于以上情况，在反复的实验下，我总结了一些 socket 编程调试的技巧（也适用于进程间通信）。

- 使用 fflush(stdout) 手动刷新缓冲区，使得可以使用输出驱动的调试。

- 重视异常情况的处理。由于 socket 多个环节出错都有可能导致程序崩溃，所以比较难以确定程序执行的逻辑和错误点。另一方面，发生的一些错误信息往往是不可读的。因此，对每个可能出错的地点进行条件盘对，将异常情况合理的处理，非常有利于 bug 的定位。常见的错误有：端口占用、不能创建 socket、文件读写错误等。
- 将客户端和服务端输出结合起来 debug，有利用明确程序执行情况，更利用错误调试。（任何一方的错误阻塞都会导致程序崩溃）

6.2 个人总结

本次实验总体上说难度不是很大，但在实验进行过程中代码的调试也花费了较多的时间。经过实验二和实验四两个实验的学习，使我门对于进程通信有了较为全面的了解。同时，使用 C 语言实现管道、socket 通信、信号量、消息通信机制等使得我对 C 语言的基本库函数和文件 I/O 操作有了更深入的理解，提高了代码编写与调试水平。最后，十分感谢刘老师上课的指导和帮助，使得我们对进程间通信几种方式有了较为深入而全面的理解。

7 附录

7.1 程序源代码

```c
// client.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>    //strlen
#include <sys/socket.h>
#include <arpa/inet.h> //inet_addr

#define BUFFER_SIZE 4096
#define IP_ADDR "127.0.0.1"
#define SERVER_PORT 8888

int main(int argc , char *argv[]) {
    int socket_desc;
    struct sockaddr_in server;
    char buffer[BUFFER_SIZE];
    char message[512], filename[512];

    // Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1) {
        printf("Could␣not␣create␣socket");
    }

    server.sin_addr.s_addr = inet_addr(IP_ADDR);
    server.sin_family = AF_INET;
    server.sin_port = htons(SERVER_PORT);

    // Connect to remote server
```

```
29      if (connect(socket_desc , (struct sockaddr *)&server , sizeof(server)
            ) < 0) {
30          puts("[Error]: Connection error!");
31          return 1;
32      }
33
34      puts("[Info]: Connected to server");
35
36      // Print the greeting info
37      int length = 0;
38      if (length = recv(socket_desc, buffer, BUFFER_SIZE, 0)) {
39          if (length < 0) {
40              printf("[Error]: Failed to receive file `%s` from server",
                    message);
41          }
42          printf("[Info]: %s", buffer);
43      }
44
45      // Send some data
46      printf("[Info]: Path of file to be read: ");
47      while (scanf("%s", message)) {
48          if (send(socket_desc, message, BUFFER_SIZE, 0) < 0) {
49              puts("[Error]: Failed to send message");
50              return 1;
51          }
52          puts("[Info]: Send message to server");
53
54          //Receive a reply from the server
55          bzero(buffer, sizeof(buffer));
56          printf("[Info]: Path of file to be saved: ");
57          scanf("%s", filename);
58
59          FILE *file_pointer = fopen(filename, "w");
60          if (file_pointer == NULL) {
61              printf("[Error]: Can not create file %s\n", filename);
62              break;
63              exit(1);
64          }
65          int flag = 1;
66          while (1) {
67              length = recv(socket_desc, buffer, BUFFER_SIZE, 0);
68              if (length < 0) {
69                  printf("[Error]: Failed to receive file `%s` from server\
                        n",  message);
70              }
71              else {
72                  // printf("%s", buffer);
```

```
                if (buffer[0] == '!' && buffer[1] == '=' && buffer[2] ==
                    '!') {
                    printf("[Error]: File `%s` not found in server\n",
                        message);
                    flag = 0;
                    break;
                }
                else {
                    // printf("%s", buffer);
                    int write_length = fwrite(buffer, sizeof(char),
                        length, file_pointer);
                    if (write_length < length) {
                        printf("[Error]: Failed to write to %s\n",
                            filename);
                        break;
                    }
                    if (length < BUFFER_SIZE) break;
                }
            }
        }
        if (flag) printf("[Info]: Save to file %s finished!\n", filename)
            ;
        fclose(file_pointer);
        bzero(message, sizeof(message));
        printf("[Info]: Path of file to be read: ");
        fflush(stdout);
    }

    puts("[Info]: Connection disrupted!");
    puts("[Info]: Bye-Bye");
    return 0;
}
```

```
// server.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>    // strlen
#include <sys/socket.h>
#include <arpa/inet.h> // inet_addr
#include <unistd.h>    // write

#include <pthread.h> // for threading , link with lpthread

#define BUFFER_SIZE 4096
#define PORT 8888

void *connection_handler(void *);

```

```
int main(int argc , char *argv[])
{
    int socket_desc , new_socket , c , *new_sock;
    struct sockaddr_in server , client;
    char *message;

    // Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1) {
        printf("[Error]: Could not create socket");
    }

    // Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);

    // Bind
    if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) <
        0) {
        puts("[Error]: Bind failed");
        return 1;
    }
    puts("[Info]: Bind done");

    // Listen
    listen(socket_desc , 3);

    // Accept and incoming connection
    puts("[Info]: Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);
    while ((new_socket = accept(socket_desc, (struct sockaddr *)&client,
        (socklen_t*)&c))) {
        puts("[Info]: Connection accepted");

        // Reply to the client
        message = "Hello, nice to meet you!\n";
        write(new_socket , message , strlen(message));

        pthread_t sniffer_thread;
        new_sock = malloc(1);
        *new_sock = new_socket;

        if (pthread_create( &sniffer_thread, NULL, connection_handler, (
            void*) new_sock) < 0) {
            perror("[Error]: Could not create thread");
            return 1;
        }
```

```
61
62          // Now join the thread , so that we dont terminate before the
                thread
63          // pthread_join( sniffer_thread , NULL);
64          puts("[Info]: Handler assigned");
65      }
66
67      if (new_socket<0)
68      {
69          perror("[Error]: Accept failed");
70          return 1;
71      }
72
73      return 0;
74  }
75
76
77  /*
78   * This will handle connection for each client
79   * */
80  void *connection_handler(void *socket_desc)
81  {
82      // Get the socket descriptor
83      int sock = *(int*)socket_desc;
84      int read_size;
85      char client_message[BUFFER_SIZE];
86      char buffer[BUFFER_SIZE];
87
88      //Receive a message from client
89      while ((read_size = recv(sock, client_message, BUFFER_SIZE, 0)) > 0)
            {
90          // Send the message back to client
91          fflush(stdout);
92          FILE *file_pointer = fopen(client_message, "r");
93          if (file_pointer == NULL) {
94              printf("[Error]: File: %s not found!\n", client_message);
95              bzero(buffer, BUFFER_SIZE);
96              strcpy(buffer, "!=!");
97              // printf("%s", buffer);
98              // fflush(stdout);
99              send(sock, buffer, sizeof(buffer), 0);
100         }
101         else  {
102             bzero(buffer, BUFFER_SIZE);
103             int file_block_length = 0;
104             while((file_block_length = fread(buffer, sizeof(char),
                    BUFFER_SIZE, file_pointer)) > 0) {
```

```
105                 printf("[Info]: File block length = %d\n",
                        file_block_length);
106                 if (send(sock, buffer, file_block_length, 0) < 0) {
107                     printf("[Info]: Send file %s failed!\n",
                            client_message);
108                     break;
109                 }
110                 bzero(buffer, sizeof(buffer));
111             }
112             printf("[Info]: File %s transferring finished!\n",
                    client_message);
113             fclose(file_pointer);
114         }
115     }
116 
117     if (read_size == 0) {
118         puts("[Info]: Client disconnected");
119         fflush(stdout);
120     }
121     else if (read_size == -1) {
122         perror("[Error]: Receive failed");
123     }
124 
125     // Free the socket pointer
126     free(socket_desc);
127     return 0;
128 }
```