

深圳大学实验报告

| | | | |
|-------|-----------------|----|------------|
| 课程名称 | 计算机网络 | | |
| 实验名称 | 实验五 Socket 网络编程 | | |
| 学 院 | 计算机与软件学院 | | |
| 专 业 | 数计 | | |
| 指导教师 | 黄耀东 | | |
| 报 告 人 | 詹耿羽 | 学号 | 2023193026 |
| 实验时间 | 2025.4.1 | | |
| 提交时间 | 2025.4.12 | | |

教务处制

实验目的与要求：

- 理解 TCP 套接字的定义
- 掌握基本的 TCP 套接字编程方法
- 了解简单网络应用的编程思路
- 了解网络编程相关的一些库

实验要求

具有 Internet 连接的主机

c 语言

开发工具：Visual Studio

方法、步骤：

1. 通过套接字在 Client 和 Server 之间建立 TCP Socket 连接
2. 在服务端使用 send() 函数向客户端发送视频文件
3. 在客户端使用 recv() 函数接收服务端发送的视频文件
4. 更改程序实现一对一的聊天窗口

实验过程及内容:

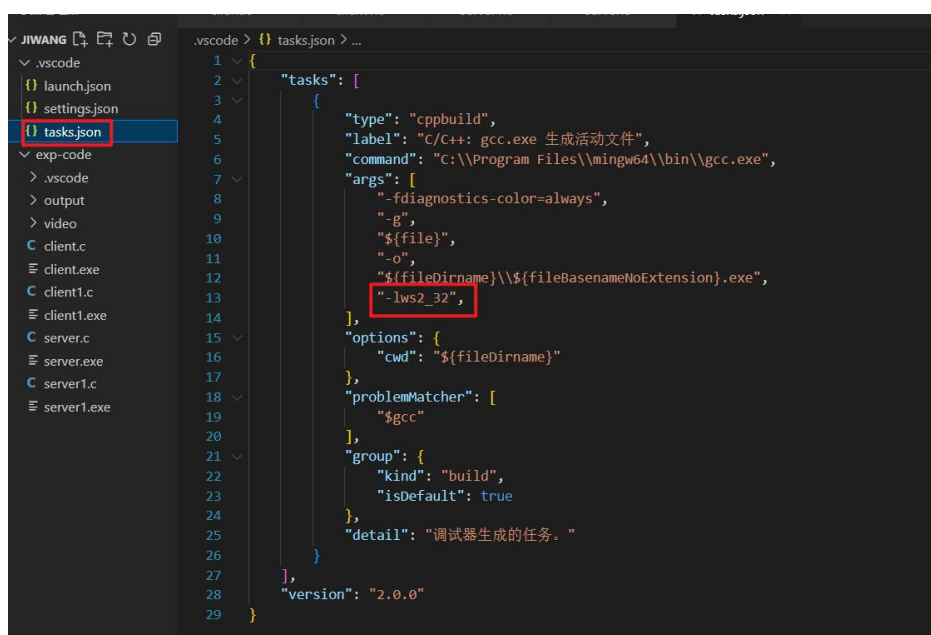
1. 下载代码和 video:

在 bb 系统下载代码和所需要传送的文件:



2. 环境配置

在 task.json 配上 “-lws2_32”, 以便于在 vscode 能够运行文件



3. 创建客户端和服务端套接字

首先, 在客户端和服务端都需要创建套接字。客户端和服务端使用 SOCK_STREAM 类型的套接字, 表示基于 TCP 协议的流连接。

客户端:

```
int sock = 0;
struct sockaddr_in serv_addr;
char buffer[BUFFER_SIZE] = {0};

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("Socket creation error");
    return -1;
}
```

- 通过 socket() 函数创建 TCP 套接字。
- 配置服务器地址结构: serv_addr.sin_family 设置为 AF_INET 表示使用 IPv4 协议,

sin_port 为服务器端口, sin_addr 为服务器的 IP 地址。

- 使用 connect()函数连接到服务器。

服务端:

```
35     int server_fd, new_socket;
36     struct sockaddr_in server_address;
37     int opt = 1;
38     int addrlen = sizeof(server_address);
39     char buffer[BUFFER_SIZE] = {0};
40
41     // 创建套接字
42     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
43     {
44         perror("socket failed");
45         exit(EXIT_FAILURE);
46     }
47     else
48         printf("Create Server Socket Success.\n");
49
```

- 服务端通过 socket()函数创建套接字。
- 配置服务端的 IP 地址和端口号, 调用 bind()函数将套接字绑定到特定地址和端口。
- 使用 listen()函数开始监听客户端连接。

4. 客户端发送请求并接收视频文件

客户端通过发送文件请求来获取视频文件。首先, 客户端发送一个请求字符串来告诉服务器所请求的文件名。

发送请求:

```
53     /*****
54     /***** 任务1: 发送到服务器, 获取视频文件 *****/
55
56     /*****
57
58     bytes_sent = send(sock, req, REQUEST_SIZE, 0);
59     if (bytes_sent < 0)
60         printf("ERROR in send\n");
61     bytes_sent = send(sock, &s_stop_byte, sizeof(s_stop_byte), 0);
62     if (bytes_sent < 0)
63         printf("ERROR in send\n");
64     printf("send req: %s\n", req);
65
```

客户端发送请求视频文件名给服务器。

接收文件大小:

```
66     // 接收文件的大小
67     int file_size;
68     unsigned long file_size_buf;
69     int bytes_recv = 0;
70     bytes_recv = recv(sock, (char *)&file_size_buf, INT_SIZE, 0);
71     file_size = ntohl(file_size_buf);
72     printf("file_size %d \n", file_size);
73
```

客户端接收从服务器发送的文件大小 (以字节为单位)。

接收视频数据: 客户端使用 recv()函数分块接收视频数据, 直到接收到完整的文件为止。

```

82     int recv_count = 0;
83     while (recv_count < file_size)
84     {
85         /***** 任务2: 使用buffer接收视频文件? *****/
86         bytes_recv = (recv_count + BUFFER_SIZE <= file_size) ? BUFFER_SIZE : file_size - recv_count;
87         bytes_recv = recv(sock, video_segement + recv_count, bytes_recv, 0);
88
89         recv_count += bytes_recv;
90         printf("%s recv progress: %d / %d\n", req, recv_count, file_size);
91     }
92
93
94
95

```

recv()函数将数据分块接收到 video_segement 缓冲区，直到接收到完整的文件。
接收结束标志：客户端接收一个字节的结束符（STOP_BYTE），表示文件传输已完成。

```

96     unsigned char r_stop_byte;
97     if (recv(sock, &r_stop_byte, 1, 0) != 1 || r_stop_byte != STOP_BYTE)
98     {
99         printf("ERROR in receiving stop byte 0x%02X \n", r_stop_byte); // 检查文件结束符
100         r_stop_byte = 'e'; // 重置

```

此字节是 0xFF，标志着传输结束。
保存文件：最后，客户端将接收到的视频数据写入本地文件。

```

101     // 写入文件
102     char file_path[40] = DOWNLOAD_PATH;
103     strcat(file_path, req);
104     printf("file path %s \n", file_path);
105     FILE *fp = fopen(file_path, "wb"); // 以二进制模式打开文件，并返回文件指针
106     if (fp == NULL)
107     {
108         perror("fopen");
109         exit(EXIT_FAILURE);
110     }
111     fwrite(video_segement, 1, file_size, fp);
112
113     /****数据接收完成阶段****/
114
115     // 释放内存

```

使用 fwrite()将视频数据写入指定路径的文件。

5. 服务端发送视频文件

服务端读取客户端请求的视频文件，然后将文件大小和数据分块发送给客户端。

读取文件：

```

103     FILE *fp = fopen(file_path, "rb"); // 以二进制模式打开文件，并返回文件指针
104     if (fp == NULL)
105     {
106         perror("File open failed");
107         exit(EXIT_FAILURE);
108     }
109

```

服务端打开指定路径的视频文件，读取文件内容。

发送文件大小：

```

// 发送文件的大小到客户端
unsigned long file_size_buf = htonl(file_size);
int bytes_sent = 0;
int reqLen = sizeof(req);
bytes_sent = send(new_socket, (char *)&file_size_buf, INT_SIZE, 0);
if (bytes_sent < 0)
    printf("ERROR in send\n");

```

服务端发送视频文件的大小（以字节为单位）给客户端。

分块发送视频文件：

```
// 发送视频片段
int send_count = 0;
while (send_count < file_size)
{
    /*******
    /******* 任务2: 发送指定视频文件（按照固定大小buffer方式） *****/
    /*******

    char buffer[BUFFER_SIZE] = {0};

    int bytes_read = (send_count + BUFFER_SIZE <= file_size) ? BUFFER_SIZE : file_size - send_count;
    fread(buffer, bytes_read, 1, fp);

    bytes_sent = send(new_socket, buffer, bytes_read, 0);
    if (bytes_sent < 0)
        printf("ERROR in send video\n");

    send_count += bytes_sent;
    printf("video send progress: %d / %d\n", send_count, file_size);
}
```

服务端使用 send()函数分块发送视频数据，每次发送一个固定大小的缓冲区。

发送结束标志：

```
// 发送文件结束符
unsigned char s_stop_byte = 0xFF;
bytes_sent = send(new_socket, &s_stop_byte, sizeof(s_stop_byte), 0);
if (bytes_sent < 0)
    printf("ERROR in send\n");
```

在所有数据发送完毕后，服务端发送一个结束符 0xFF，告诉客户端传输已经完成。

6. 扩展任务：终止视频传输

在扩展任务中，客户端和服务端需要根据某个信号来终止视频传输。例如，客户端可以发送一个特定的消息（例如 END_REQUEST）来告诉服务端终止文件传输。

服务端接收终止信号：

服务器接收到 END_REQUEST 请求后，可以立即结束文件传输，并关闭连接。

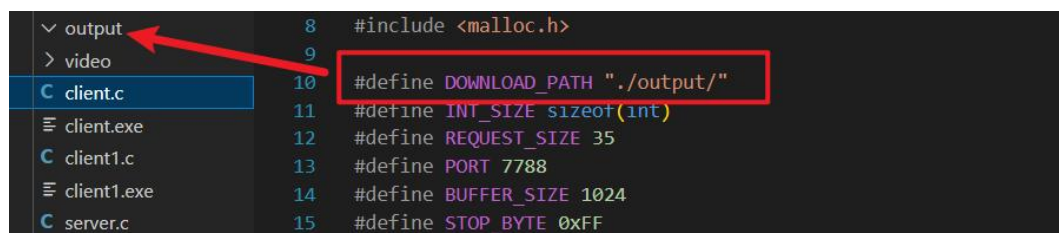
7. 资源清理

最后，关闭文件和套接字，释放分配的内存，并调用 WSACleanup()来清理 Windows Socket 库。

```
154     /***结束阶段***/
155     closesocket(server_fd);
156     closesocket(new_socket);
157     WSACleanup();
158
```

8. 开始下载

这里将下载路径设置为:当前目录下的 output 文件夹。

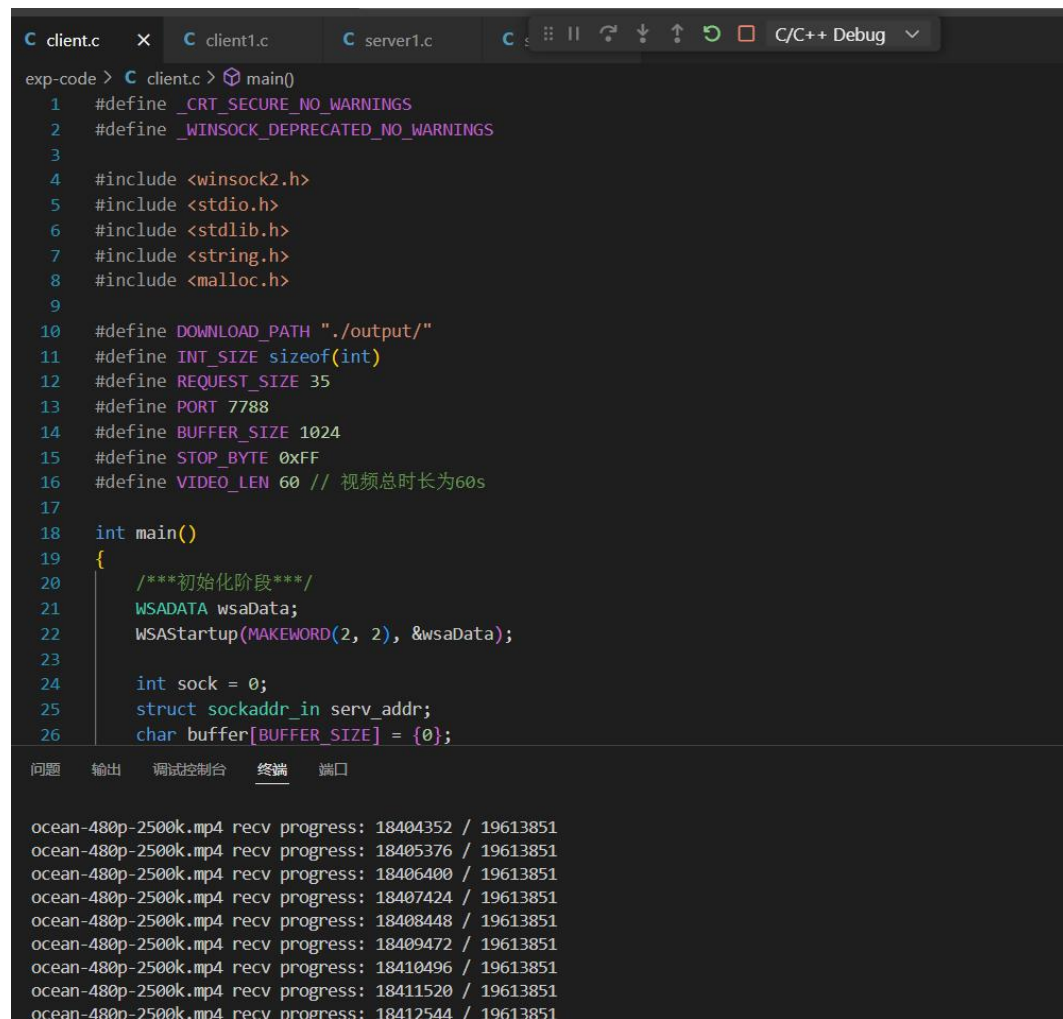


```
8  #include <malloc.h>
9
10 #define DOWNLOAD_PATH "./output/"
11 #define INT_SIZE sizeof(int)
12 #define REQUEST_SIZE 35
13 #define PORT 7788
14 #define BUFFER_SIZE 1024
15 #define STOP_BYTE 0xFF
```


运行 server.c 代码:

```
PS E:\jiwang> & 'c:\Users\詹耿羽\.vscode\extensions\ms-vscode  
n-ybzhpk4x.dut' '--stdout=Microsoft-MIEngine-Out-ltfieb5d.bak  
Exe=C:\Program Files\mingw64\bin\gdb.exe' '--interpreter=mi'  
Create Server Socket Success.  
Server Bind Port Success.  
Server Listening.....  
█
```

运行 client.c 代码, 视频开始下载:



```
C client.c X C client1.c C server1.c C C/C++ Debug v
exp-code > C client.c > main()
1  #define _CRT_SECURE_NO_WARNINGS
2  #define _WINSOCK_DEPRECATED_NO_WARNINGS
3
4  #include <winsock2.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <malloc.h>
9
10 #define DOWNLOAD_PATH "./output/"
11 #define INT_SIZE sizeof(int)
12 #define REQUEST_SIZE 35
13 #define PORT 7788
14 #define BUFFER_SIZE 1024
15 #define STOP_BYTE 0xFF
16 #define VIDEO_LEN 60 // 视频总时长为60s
17
18 int main()
19 {
20     /**初始化阶段**/
21     WSADATA wsaData;
22     WSStartup(MAKEWORD(2, 2), &wsaData);
23
24     int sock = 0;
25     struct sockaddr_in serv_addr;
26     char buffer[BUFFER_SIZE] = {0};

问题 输出 调试控制台 终端 端口
ocean-480p-2500k.mp4 recv progress: 18404352 / 19613851
ocean-480p-2500k.mp4 recv progress: 18405376 / 19613851
ocean-480p-2500k.mp4 recv progress: 18406400 / 19613851
ocean-480p-2500k.mp4 recv progress: 18407424 / 19613851
ocean-480p-2500k.mp4 recv progress: 18408448 / 19613851
ocean-480p-2500k.mp4 recv progress: 18409472 / 19613851
ocean-480p-2500k.mp4 recv progress: 18410496 / 19613851
ocean-480p-2500k.mp4 recv progress: 18411520 / 19613851
ocean-480p-2500k.mp4 recv progress: 18412544 / 19613851
```

下载完成:

服务器通过 `accept()`接收客户端的连接请求。

2. 多线程消息接收

为了实现一对一实时聊天，使用多线程技术来处理客户端与服务器的消息收发。我们通过创建新的线程来处理每个客户端的消息接收。

客户端接收消息线程：

客户端通过 `RecvThread` 线程接收来自服务器的消息。如果接收到的是文本消息，则直接打印输出；如果是文件请求，则启动相应的文件接收处理逻辑。

```
15 unsigned __stdcall RecvThread(void *sock) {
16     char buffer[BUFFER_SIZE];
17
18     while(1) {
19         int bytes_recv = recv(*(SOCKET*)sock, buffer, BUFFER_SIZE, 0);
20         if(bytes_recv <= 0) break;
21
22         switch(buffer[0]) {
23             case CMD_FILE:
24                 printf("\n[服务器] 开始传输文件...\n");
25                 // 原有文件接收逻辑...
26                 break;
27
28             case CMD_MSG:
29                 buffer[bytes_recv] = '\0';
30                 printf("\n[服务器]: %s\n> ", buffer+1);
31                 break;
32         }
33     }
34     return 0;
35 }
```

服务器接收消息线程：

服务器端同样使用 `RecvThread` 来接收来自客户端的消息，并根据消息类型进行相应处理。

```

21 // 消息接收线程
22 unsigned __stdcall RecvThread(void *arg) {
23     SOCKET sock = ((struct ThreadArgs*)arg)->sock;
24     char buffer[BUFFER_SIZE];
25
26     while(1) {
27         int bytes_rcv = recv(sock, buffer, BUFFER_SIZE, 0);
28         if(bytes_rcv <= 0) break;
29
30         switch(buffer[0]) { // 第一个字节为命令类型
31             case CMD_FILE:
32                 printf("\n[客户端] 文件请求: %s\n", buffer+1);
33                 // 原有文件处理逻辑...
34                 break;
35
36             case CMD_MSG:
37                 buffer[bytes_rcv] = '\0';
38                 printf("\n[客户端]: %s\n> ", buffer+1);
39                 break;
40         }
41     }
42     closesocket(sock);
43     free(arg);
44     return 0;
45 }

```

3. 消息发送和接收

客户端和服务端之间通过指定命令字节来区分不同类型的消息。客户端发送文本消息和文件请求时，通过指定第一个字节来标识消息类型。服务器根据消息类型进行不同的处理。

客户端发送文本消息：

客户端将输入的消息包裹在 CMD_MSG 命令字节后，并通过 send() 函数发送给服务器。

```

59     if(strncmp(input, "FILE:", 5) == 0) {
60         char *filename = input + 5;
61         char packet[BUFFER_SIZE];
62         packet[0] = CMD_FILE;
63         strcpy(packet+1, filename);
64         send(sock, packet, strlen(filename)+2, 0);
65     } else {

```

客户端发送文件请求：

如果客户端需要发送文件，它会将文件名放在 CMD_FILE 命令字节后，并将其发送到服务器。

```

65         } else {
66             char packet[BUFFER_SIZE];
67             packet[0] = CMD_MSG;
68             strcpy(packet+1, input);
69             send(sock, packet, strlen(input)+2, 0);
70         }
71     }
72

```

服务器响应消息:

服务器接收到客户端的消息后, 会通过 send()函数将响应消息发送回客户端。

```

83         char packet[BUFFER_SIZE];
84         packet[0] = CMD_MSG;
85         strcpy(packet+1, msg);
86
87         if(send(client_sock, packet, strlen(msg)+2, 0) <= 0) break;
88     }

```

4. 资源清理

在通信完成后, 客户端和服务端都需要关闭套接字, 并清理资源。服务器端关闭每个客户端连接的套接字, 客户端在退出前关闭与服务器的套接字连接。

客户端关闭套接字:

```

2         closesocket(sock);
3
4         WSACleanup();

```

服务器关闭套接字:

```

01         closesocket(server_fd);
02
03         WSACleanup();
04         return 0;
05     }

```

5. 开始运行聊天:

运行 server 代码:

```
资源管理器  ...  C client.c  C client1.c  C server1.c  C server.c  ...  运行  调试  窗口  输出  问题

JIIWANG  ...  exp-code > C server1.c > main()
47 int main() {
62 while(1) {
64     struct sockaddr_in client_addr;
65     int addr_len = sizeof(client_addr);
66     SOCKET client_sock = accept(server_fd, (struct sockaddr*)&client_addr, &addr_len);
67
68     struct ThreadArgs *args = malloc(sizeof(struct ThreadArgs));
69     args->sock = client_sock;
70     args->addr = client_addr;
71
72     printf("客户端已连接: %s:%d\n",
73           inet_ntoa(client_addr.sin_addr),
74           ntohs(client_addr.sin_port));
75     _beginthreadex(NULL, 0, RecvThread, args, 0, NULL);
76
77     // 主线程处理控制台输入
78     while(1) {
79         printf("> ");
80         char msg[BUFFER_SIZE-1];
81         fgets(msg, sizeof(msg), stdin);
82     }
83 }
84 }
```

问题 输出 调试控制台 终端 窗口

PS E:\jiiwang> & 'c:\Users\詹耿羽\.vscode\extensions\ms-vscode.cpptools-1.25.0-win32-x64\debugAdapters\bin\WindowsDebugAdapter.exe' --stdout=Microsoft-MIEngine-Out-sinztbtm.qbp' --stderr=Microsoft-MIEngine-Error-v25ph0nt.f5
Exe=C:\Program Files\mingw64\bin\gdb.exe' --interpreter=mi'
服务器已启动, 等待连接...

运行 client 代码:

```
C client.c  C client1.c  C server1.c  C  ...  C/C++ Debug  ...  运行  调试  窗口  输出  问题

exp-code > C client1.c > main()
37 int main() {
47
48     connect(sock, (struct sockaddr*)&addr, sizeof(addr));
49     printf("已连接到服务器, 输入消息开始聊天, 输入FILE:文件名发送文件\n");
50
51     _beginthreadex(NULL, 0, RecvThread, &sock, 0, NULL);
52
53     while(1) {
54         printf("> ");
55         char input[BUFFER_SIZE-1];
56         fgets(input, sizeof(input), stdin);
57         input[strcspn(input, "\n")] = '\0';
58
59         if(strncmp(input, "FILE:", 5) == 0) {
60             char *filename = input + 5;
61             char packet[BUFFER_SIZE];
62             packet[0] = CMD_FILE;
63             strcpy(packet+1, filename);
64             send(sock, packet, strlen(filename)+2, 0);
65         }
66     }
67 }
```

问题 输出 调试控制台 终端 窗口

PS E:\jiiwang> & 'c:\Users\詹耿羽\.vscode\extensions\ms-vscode.cpptools-1.25.0-win32-x64\debugAdapters\bin\WindowsDebugAdapter.exe' --stdout=Microsoft-MIEngine-Out-mia20gl0.sne' --stderr=Microsoft-MIEngine-Error-tlktc513.eao' --pi
Exe=C:\Program Files\mingw64\bin\gdb.exe' --interpreter=mi'
已连接到服务器, 输入消息开始聊天, 输入FILE:文件名发送文件
>

可以看到客户端和服务端已经连接成功:

The screenshot shows a VS Code editor with a C++ project. The file `server1.c` is open, displaying a server program that listens for connections and spawns a thread to handle each client. The terminal output shows the server starting and successfully accepting a connection from `127.0.0.1:50951`.

```
exp-code > C server1.c > main()
47 int main() {
62 while(1) {
64     struct sockaddr_in client_addr;
65     int addr_len = sizeof(client_addr);
66     SOCKET client_sock = accept(server_fd, (struct sockaddr*)&client_addr, &addr_len);
67     struct ThreadArgs *args = malloc(sizeof(struct ThreadArgs));
68     args->sock = client_sock;
69     args->addr = client_addr;
70     printf("客户端已连接: %s:%d\n",
71           inet_ntoa(client_addr.sin_addr),
72           ntohs(client_addr.sin_port));
73     _beginthreadex(NULL, 0, RecvThread, args, 0, NULL);
74     // 主线程处理控制台输入
75     while(1) {
76         printf("> ");
77         char msg[BUFFER_SIZE-1];
78         fgets(msg, sizeof(msg), stdin);
79     }
80 }
```

问题 输出 调试控制台 终端 端口

```
PS E:\jiwang> & 'c:\Users\詹耿羽\.vscode\extensions\ms-vscode.cpptools-1.25.0-win32-x64\debugAdapters\bin\Win-qwkwm04y.jx4' '--stdout=Microsoft-MIEngine-Out-sinztbtm.qbp' '--stderr=Microsoft-MIEngine-Error-v25ph0nt.f5
Exe=C:\Program Files\mingw64\bin\gdb.exe' '--interpreter=mi'
服务器已启动, 等待连接...
客户端已连接: 127.0.0.1:50951
> |
```

服务端输入：计算机网络：

The screenshot shows a terminal window where the user has entered the text '计算机网络' (Computer Network) in response to the server's prompt.

```
80     fgets(msg, sizeof(msg), stdin);
问题 输出 调试控制台 终端 端口
PS E:\jiwang> & 'c:\Users\詹耿羽\.vscode\extensions\ms
n-qwkwm04y.jx4' '--stdout=Microsoft-MIEngine-Out-sinztbt
Exe=C:\Program Files\mingw64\bin\gdb.exe' '--interprete
服务器已启动, 等待连接...
客户端已连接: 127.0.0.1:50951
> 计算机网络
> |
```

客户端那边收到了服务端发来的计算机网络：


```

    fgets(msg, sizeof(msg), stdin);

问题  输出  调试控制台  终端  端口

PS E:\jiwang> & 'c:\Users\詹耿羽\.vscode\extensions\ms-vscode.cpptools
n-yw20qpg1.lso' '--stdout=Microsoft-MIEngine-Out-mia20gl0.sne' '--stdr
Exe=C:\Program Files\mingw64\bin\gdb.exe' '--interpreter=mi'
已连接到服务器，输入消息开始聊天，输入FILE:文件名发送文件
>
[服务器]: 计算机网络
> 
```

客户端发送 1~2~3~gogogo:

```

已连接到服务器，输入消息开始聊天，输入FILE:文件名发送文件
>
[服务器]: 计算机网络
> 1~2~3~gogogo
> 
```

服务端那边也收到了客户端发来的 1~2~3~gogogo:

```

n-qwkwm04y.jx4' '--stdout=Microsoft-MIEngine-Out-sin
Exe=C:\Program Files\mingw64\bin\gdb.exe' '--interpre
服务器已启动，等待连接...
客户端已连接: 127.0.0.1:50951
> 计算机网络
>
[客户端]: 1~2~3~gogogo
> 
```

数据处理分析：
暂不涉及。

深圳大学学生实验报告用纸

实验结论：

本实验通过实现基于 TCP 的客户端和服务端文件传输，演示了如何使用 `send()` 和 `recv()` 函数进行数据的分块传输，以及如何通过 `Socket` 实现基本的文件传输。扩展任务中的终止信号处理使得该过程更具交互性和灵活性，增强了程序的鲁棒性。

通过本实验，实现了一个简单的基于 TCP 协议的聊天应用，其中包括消息的发送、接收以及文件传输。通过多线程技术，确保了客户端和服务端能够同时进行消息的发送和接收，不会发生阻塞。整个应用结构清晰，易于扩展。

| |
|------------------|
| |
| 指导教师批阅意见： |
| 成绩评定： |
| 指导教师签字： 年 月 日 |
| 备注： |

注： 1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。