

# 北京邮电大学实验报告

实验名称	计算机网络实验二		学 院	网络空间 安全学院	指导教师	刘建毅
班 级	班内序号	学 号		学生姓名	成绩	
20222118	04	2022211570		项枫		
01	06	2022211575		张艺翱		
实 验 内 容	<p>本实验旨在使用 Socket 编程实现一个基于 TCP 协议的文件传输系统，包含服务器端和客户端两个程序。客户端发送文件传输请求，服务器端接收请求并将指定文件传输给客户端。</p> <p>具体包括以下内容：</p> <p>（1）准备服务器端和客户端的源代码：编写服务器和客户端程序。</p> <p>（2）编译代码：使用 gcc 编译服务器端和客户端代码。</p> <p>（3）运行服务器端程序：在服务器端机器上运行服务器程序。</p> <p>（4）运行客户端程序：在客户端机器上运行客户端程序并发送文件请求。</p> <p>（5）测试文件传输：进行文件传输测试，验证不同类型文件的传输效果。</p> <p>（6）验证文件一致性：检查传输后客户端接收文件与服务器原始文件的一致性。</p>					
学生 实验 报告 (附页)	见附页					
实 验 成 绩 评 定	<p>评语：</p> <p>成绩：</p> <p>指导教师签名：</p> <p>年 月 日</p>					

注：评语要体现每个学生的工作情况，可以加页。

## 目录

《计算机网络》实验报告 .....	2
<b>实验二：基于 SOCKET 的文件传输 .....</b>	<b>2</b>
一、 实验内容 .....	3
二、 实验目的 .....	3
三、 实验环境 .....	3
四、 软件设计 .....	3
（一） 数据结构 .....	3
（二） 模块功能 .....	4
（三） 程序流程 .....	4
五、 实验步骤 .....	7
六、 实验总结和心得体会 .....	11
（一） 调试过程中遇到的问题及解决过程 .....	11
（二） 收获与提高 .....	11
（三） 实验设计和安排的不足 .....	11
附录：程序源代码 .....	12

## 《计算机网络》实验报告

### 实验二：基于 SOCKET 的文件传输

组员信息：

姓名	班级	学号	分工
项枫	2022211801	2022211570	代码构建、实验实施及截图（50%）
张艺翱	2022211801	2022211575	资料整理、流程图及报告撰写（50%）

## 一、 实验内容

本实验旨在使用 Socket 编程实现一个基于 TCP 协议的文件传输系统，包含服务器端和客户端两个程序。客户端发送文件传输请求，服务器端接收请求并将指定文件传输给客户端。

具体包括以下内容：

- (1) 准备服务器端和客户端的源代码：编写服务器和客户端程序。
- (2) 编译代码：使用 gcc 编译服务器端和客户端代码。
- (3) 运行服务器端程序：在服务器端机器上运行服务器程序。
- (4) 运行客户端程序：在客户端机器上运行客户端程序并发送文件请求。
- (5) 测试文件传输：进行文件传输测试，验证不同类型文件的传输效果。
- (6) 验证文件一致性：检查传输后客户端接收文件与服务器原始文件的一致性。

## 二、 实验目的

通过本实验，掌握基于 TCP 协议的 Socket 编程方法，能够实现客户端与服务器之间的文件传输。还将学习如何进行网络编程中的错误处理，提升数据传输的可靠性与稳定性。

## 三、 实验环境

操作系统：Linux

编程语言：C/C++

开发工具：gcc

测试设备：两台联网的计算机，一台作为服务器，另一台作为客户端

## 四、 软件设计

### (一) 数据结构

程序中使用的主要数据结构包括：

1. 服务器端 (server.c)：

`struct sockaddr_in server_addr`: 用于存储服务器的地址信息。

`struct sockaddr_in client_addr`: 用于存储客户端的地址信息。

`char buffer[BUFFER_SIZE]`: 用于存储文件传输的临时缓冲区。

## 2. 客户端 (client.c):

`struct sockaddr_in servaddr`: 用于存储服务器的地址信息。

`char recv_buf[1048576]`: 用于存储从服务器接收的数据。

`char send_buf[1048576]`: 用于存储发送到服务器的数据。

## (二) 模块功能

### (1) 服务器端:

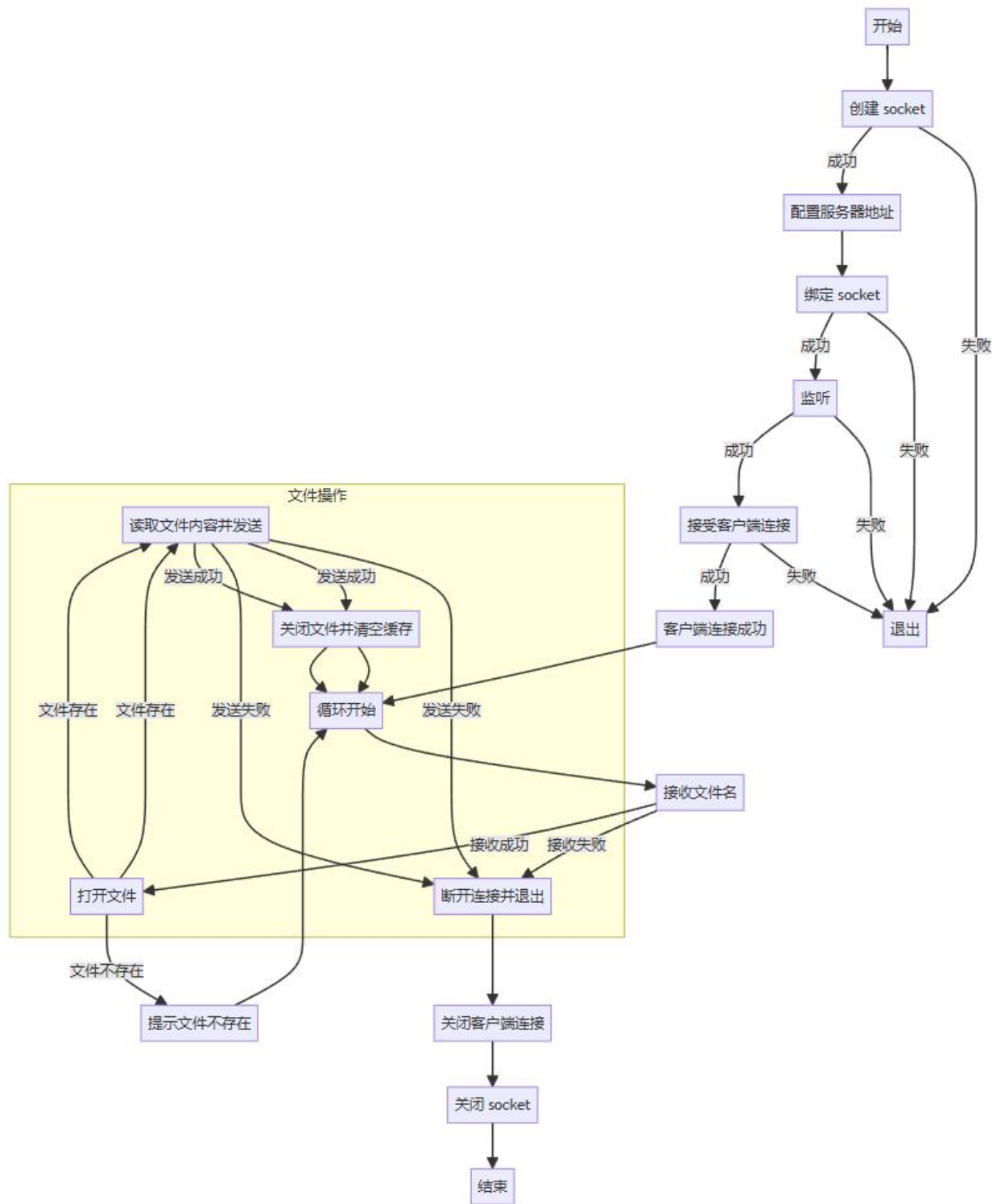
- 创建 `Socket`, 绑定地址, 监听连接请求。
- 接收客户端传来的文件名, 打开文件并读取内容, 通过 `Socket` 发送给客户端。
- 在文件发送完毕后, 关闭文件和 `Socket`。

### (2) 客户端:

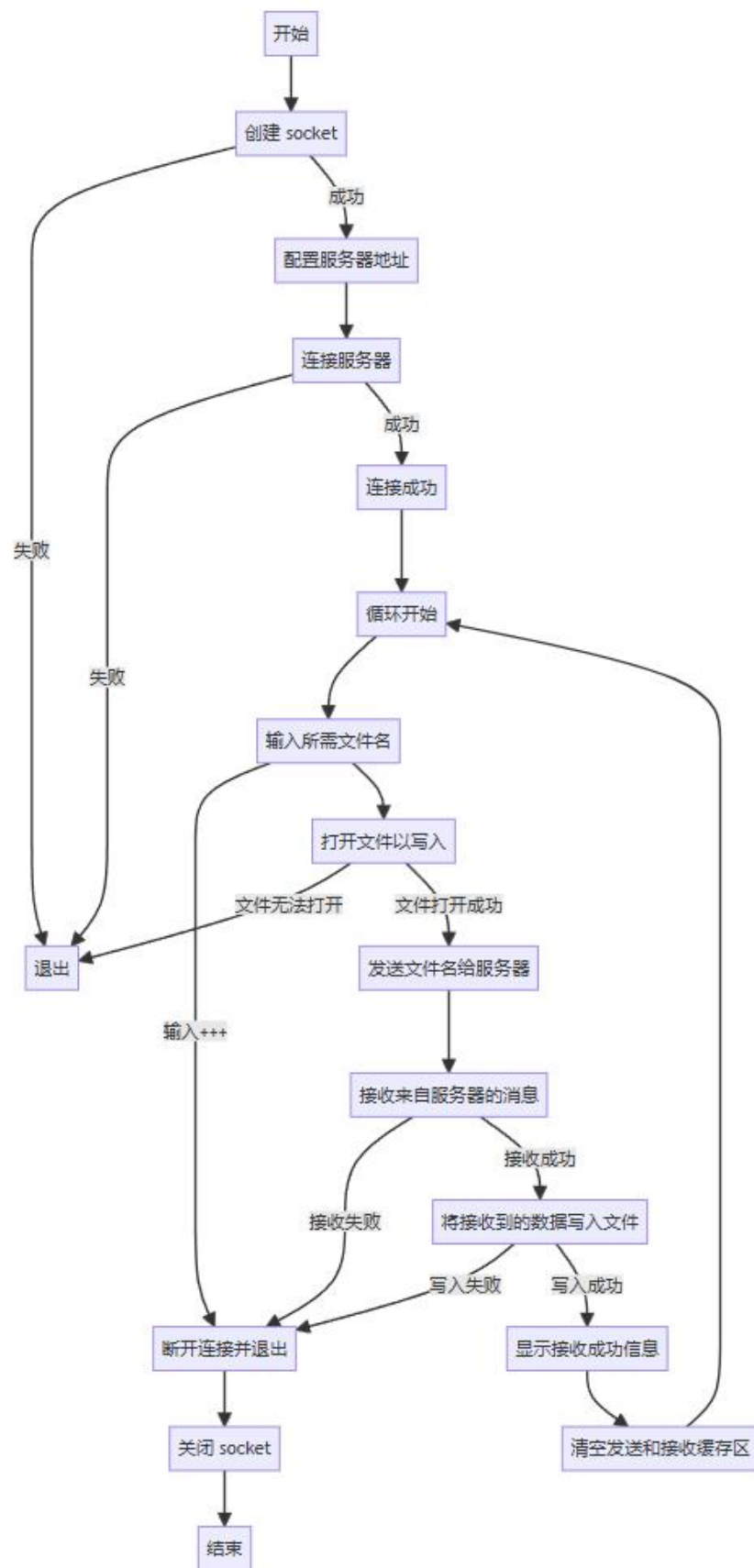
- 创建 `Socket` 并连接到服务器。
- 发送文件名给服务器, 接收服务器传来的文件内容, 并将其写入新文件。
- 文件传输完成后, 关闭文件和 `Socket`。

## (三) 程序流程

### 1. 服务器端流程图:



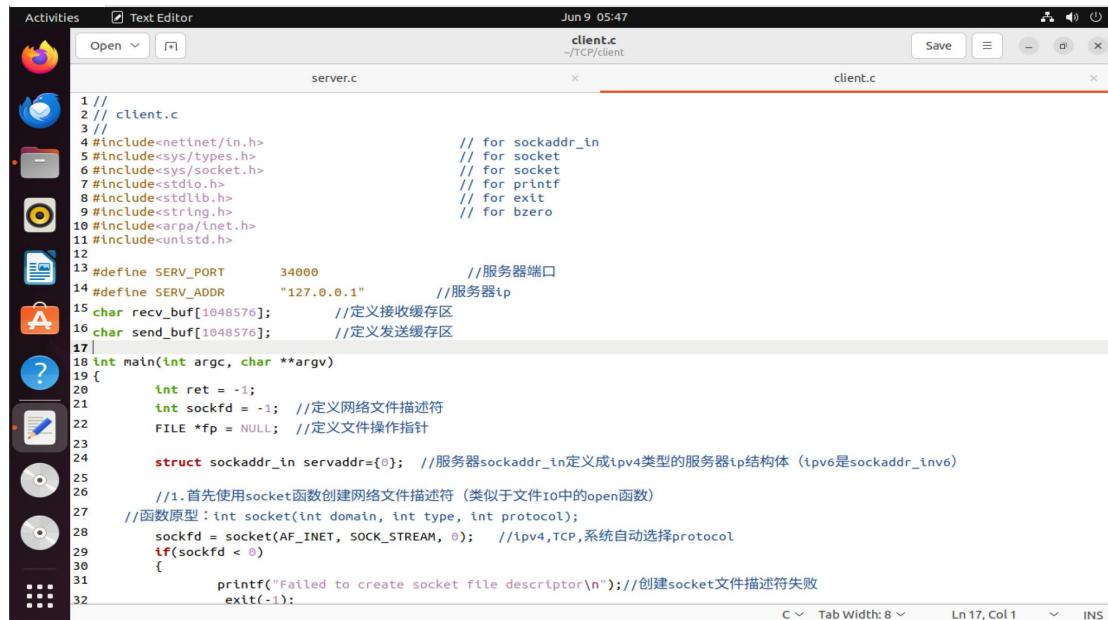
2. 客户端流程图：



## 五、 实验步骤

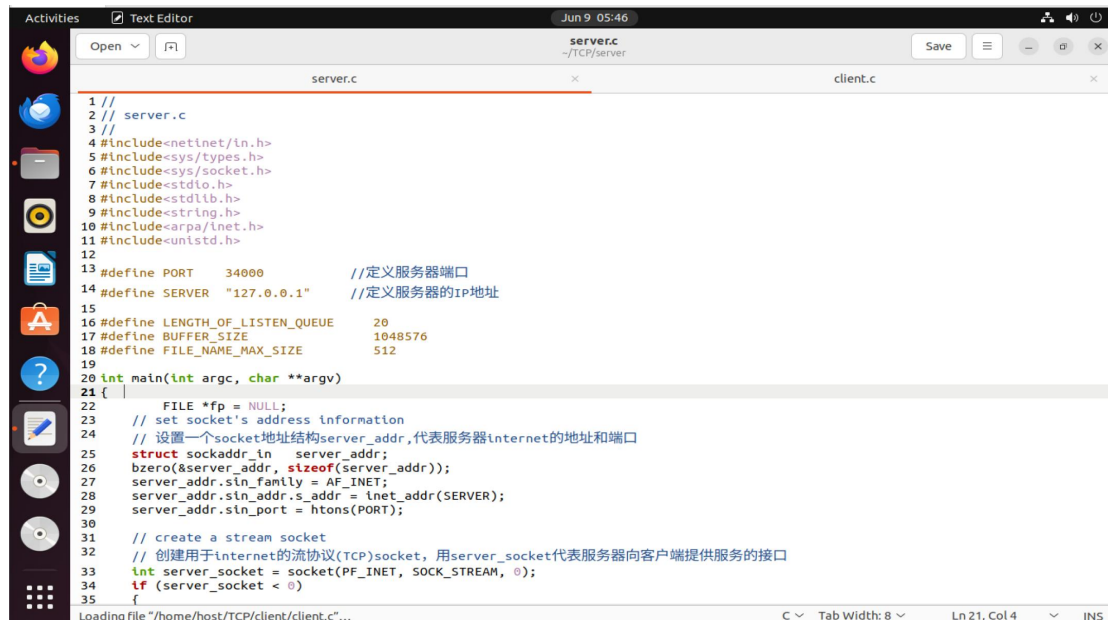
### 1. 编译代码

客户端源代码:



```
1 //
2 // client.c
3 //
4 #include<netinet/in.h>           // for sockaddr_in
5 #include<sys/types.h>           // for socket
6 #include<sys/socket.h>         // for socket
7 #include<stdio.h>              // for printf
8 #include<stdlib.h>             // for exit
9 #include<string.h>             // for bzero
10 #include<arpa/inet.h>          // for bzero
11 #include<unistd.h>
12
13 #define SERVER_PORT    34000      //服务器端口
14 #define SERVER_ADDR    "127.0.0.1" //服务器ip
15 char recv_buf[1048576];         //定义接收缓存区
16 char send_buf[1048576];        //定义发送缓存区
17
18 int main(int argc, char **argv)
19 {
20     int ret = -1;
21     int sockfd = -1; //定义网络文件描述符
22     FILE *fp = NULL; //定义文件操作指针
23
24     struct sockaddr_in servaddr={0}; //服务器sockaddr_in定义成ipv4类型的服务器ip结构体 (ipv6是sockaddr_in6)
25
26     //1.首先使用socket函数创建网络文件描述符 (类似于文件io中的open函数)
27     //函数原型: int socket(int domain, int type, int protocol);
28     sockfd = socket(AF_INET, SOCK_STREAM, 0); //ipv4,TCP,系统自动选择protocol
29     if(sockfd < 0)
30     {
31         printf("Failed to create socket file descriptor\n");//创建socket文件描述符失败
32         exit(-1);
33     }
```

服务器端源代码:

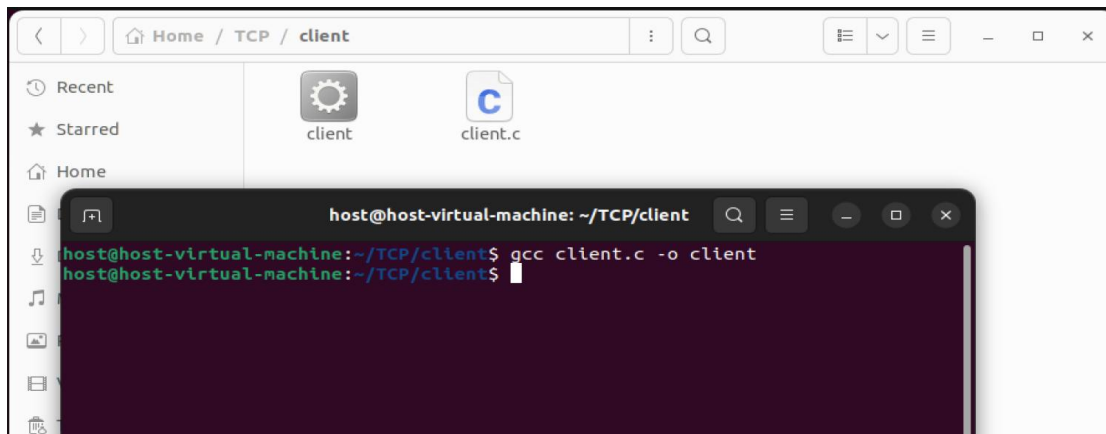
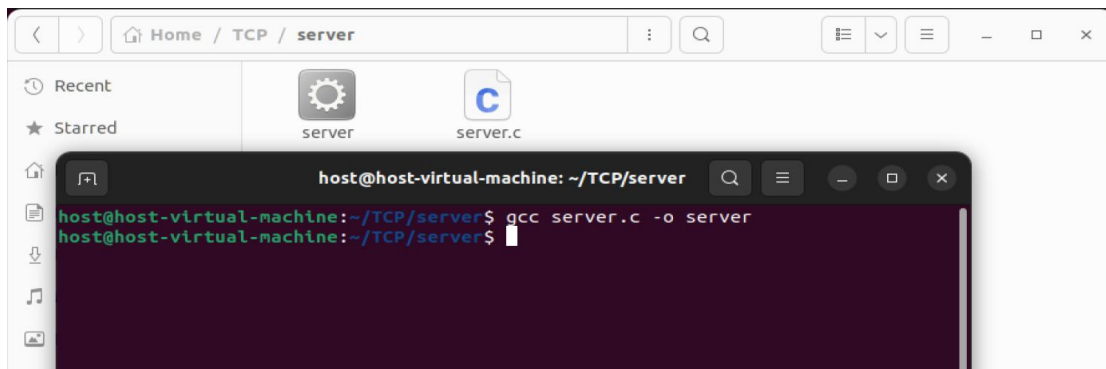


```
1 //
2 // server.c
3 //
4 #include<netinet/in.h>
5 #include<sys/types.h>
6 #include<sys/socket.h>
7 #include<stdio.h>
8 #include<stdlib.h>
9 #include<string.h>
10 #include<arpa/inet.h>
11 #include<unistd.h>
12
13 #define PORT    34000      //定义服务器端口
14 #define SERVER  "127.0.0.1" //定义服务器的IP地址
15
16 #define LENGTH_OF_LISTEN_QUEUE    20
17 #define BUFFER_SIZE    1048576
18 #define FILE_NAME_MAX_SIZE    512
19
20 int main(int argc, char **argv)
21 {
22     FILE *fp = NULL;
23     // set socket's address information
24     // 设置一个socket地址结构server_addr,代表服务器internet的地址和端口
25     struct sockaddr_in server_addr;
26     bzero(&server_addr, sizeof(server_addr));
27     server_addr.sin_family = AF_INET;
28     server_addr.sin_addr.s_addr = inet_addr(SERVER);
29     server_addr.sin_port = htons(PORT);
30
31     // create a stream socket
32     // 创建用于internet的流协议(TCP)socket, 用server_socket代表服务器向客户端提供服务的接口
33     int server_socket = socket(PF_INET, SOCK_STREAM, 0);
34     if(server_socket < 0)
35     {
```

使用 gcc 编译服务器端和客户端代码:

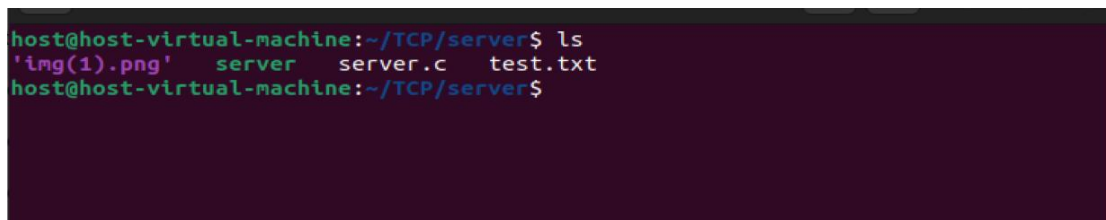
gcc server.c -o server

gcc client.c -o client



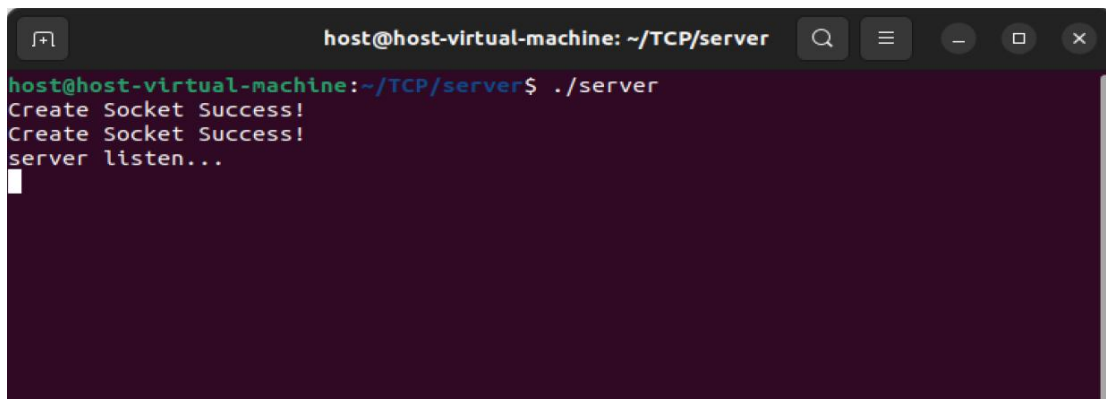
## 2. 启动程序

(1) 在 server 端准备好文件: test.txt、img(1).png



(2) 服务端启动 server 服务端程序:

```
./server
```



(3) 客户端启动 client 服务端程序:



```
./client
```

```
host@host-virtual-machine:~/TCP/client$ ./client
sockfd=3
The client succeeded to connect to the server:[127.0.0.1]
Please enter the required documents:

```

3. 输入需要传输的文件名并发送请求

测试传输文本文件、图片文件等不同类型的文件。

(1) 进行文本传输测试:

```
host@host-virtual-machine: ~/TCP/client
host@host-virtual-machine:~/TCP/client$ ./client
sockfd=3
The client succeeded to connect to the server:[127.0.0.1]
Please enter the required documents:
test.txt
Recieve File:
13 bytes recieved, and stored in test.txt

host@host-virtual-machine: ~/TCP/server
host@host-virtual-machine:~/TCP/server$ ./server
Create Socket Success!
Create Socket Success!
server listen...
The client is successfully connected,new_server_socket = 4
Client ip = 127.0.0.1
Client port = 52710
Waiting... :
Recieve messeage from client:'test.txt'
File: test.txt open success!
file_block_length = 13
File: test.txt Transfer Finished!
```

(2) 进行图片传输测试:

```
Please enter the required documents:
img(1).png
Recieve File:
2948 bytes recieved, and stored in img(1).png

Waiting... :
Recieve messeage from client:'img(1).png'
File: img(1).png open success!
file_block_length = 2948
File: img(1).png Transfer Finished!
```

4. 测试不存在的文件的传输请求

```
Please enter the required documents:
123.txt

Waiting... :
Recieve messeage from client:'123.txt'
File: 123.txt Not Found!
```

5. 断开连接:

```
Please enter the required documents:
+++
Close the connection and exit
host@host-virtual-machine:~/TCP/client$
```

```
Waiting... :
The client disconnects, exit !!!
host@host-virtual-machine:~/TCP/server$
```

## 6. 验证文件一致性

比较客户端接收到的文件与服务器端原始文件的大小和内容是否一致。

Server:

```
host@host-virtual-machine:~/TCP/server$ ll
total 44
drwxrwxr-x 2 host host 4096 Jun 9 05:42 ./
drwxrwxr-x 4 host host 4096 Jun 9 02:40 ../
-rw-rw-r-- 1 host host 2948 Jun 9 05:37 'img(1).png'
-rwxrwxr-x 1 host host 16856 Jun 9 05:27 server*
-rw-rw-r-- 1 host host 5157 Jun 9 05:27 server.c
-rw-rw-r-- 1 host host 13 Jun 9 03:49 test.txt
host@host-virtual-machine:~/TCP/server$
```

Client:

```
host@host-virtual-machine:~/TCP/client$ ll
total 44
drwxrwxr-x 2 host host 4096 Jun 9 05:42 ./
drwxrwxr-x 4 host host 4096 Jun 9 02:40 ../
-rw-rw-r-- 1 host host 4096 Jun 9 05:09 123.txt
-rw-rw-r-- 1 host host 0 Jun 9 04:33 .bak
-rwxrwxr-x 1 host host 16832 Jun 9 05:32 client*
-rw-rw-r-- 1 host host 3707 Jun 9 05:32 client.c
-rw-rw-r-- 1 host host 2948 Jun 9 05:40 'img(1).png'
-rw-rw-r-- 1 host host 13 Jun 9 05:07 test.txt
host@host-virtual-machine:~/TCP/client$
```

可以看到，客户端接收到的文件与服务器端原始文件的大小和内容一致。

## 7. 差错处理功能

(1) 服务器端:

文件不存在时，返回错误信息给客户端:

```
Waiting... :
Recieve message from client:'123.txt'
File: 123.txt Not Found!
```

文件读取错误时，返回错误信息。

(2) 客户端:

接收文件内容过程中断开连接，提示传输失败:

```
Waiting... :
The client disconnects, exit !!!
host@host-virtual-machine:~/TCP/server$
```

如若 server 服务器未开启，client 端会连接失败:

```
host@host-virtual-machine:~/TCP/client$ ./client
sockfd=3
The client failed to connect to the server
host@host-virtual-machine:~/TCP/client$
```

## 六、实验总结和心得体会

### （一）调试过程中遇到的问题及解决过程

#### 1. 问题一：Socket 连接失败

解决方法：检查服务器和客户端的 IP 地址和端口号配置是否正确，确保防火墙未阻挡相关端口。

#### 2. 问题二：文件传输过程中数据丢失

解决方法：增加数据校验机制，确保每个数据包的正确接收和写入。

### （二）收获与提高

通过本次实验，对 SOCKET 编程有了更深入的理解，尤其是在 TCP 连接的建立、数据传输及异常处理方面，积累了实际编程经验。此外，还学习了如何进行跨设备的网络调试，对协议软件设计有了更清晰的认识。

### （三）实验设计和安排的不足

- 传输文件类型的多样性欠缺，可以增加对其他文件类型的支持。
- 没有实现数据传输的加密和压缩，可以进一步提升程序的安全性和效率。

---

## 附录：程序源代码

### 1. 客户端代码

```
//
// client.c
//
#include<netinet/in.h>           // for sockaddr_in
#include<sys/types.h>           // for socket
#include<sys/socket.h>          // for socket
#include<stdio.h>               // for printf
#include<stdlib.h>              // for exit
#include<string.h>              // for bzero
#include<arpa/inet.h>
#include<unistd.h>

#define SERV_PORT  34000        //服务器端口
#define SERV_ADDR  "127.0.0.1" //服务器 ip
char recv_buf[1048576];        //定义接收缓存区
char send_buf[1048576];        //定义发送缓存区

int main(int argc, char **argv)
{
    int ret = -1;
    int sockfd = -1; //定义网络文件描述符
    FILE *fp = NULL; //定义文件操作指针

    struct sockaddr_in servaddr={0}; //服务器 sockaddr_in 定义成 ipv4 类型的服务器 ip 结构体 (ipv6
是 sockaddr_in6)

    //1. 首先使用 socket 函数创建网络文件描述符（类似于文件 IO 中的 open 函数）
    //函数原型: int socket(int domain, int type, int protocol);
    sockfd = socket(AF_INET, SOCK_STREAM, 0); //ipv4, TCP, 系统自动选择 protocol
    if(sockfd < 0)
    {
        printf("Failed to create socket file descriptor\n");//创建 socket 文件描述符失败
        _exit(-1);
    }
    printf("sockfd=%d\n", sockfd);
    //2. 使用 connect 函数连接服务器
    //函数原型: int connect(int socket, const struct sockaddr *address, socklen_t address_len);
    servaddr.sin_family = AF_INET; // 定义 servaddr 的
domain 地址族为 ipv4
    servaddr.sin_port = htons(SERV_PORT); // 定义 servaddr 的
```

portnum 为 SERV\_PORT(8010), host to net short

```
servaddr.sin_addr.s_addr = inet_addr(SERV_ADDR);           // 定义 servaddr 的 address 为  
SERV_ADDR(192.168.1.23)  person----->u32
```

```
ret = connect(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr));  
if(ret < 0)  
{  
    printf("The client failed to connect to the server\n");  
    _exit(-1);  
}  
printf("The client succeeded to connect to the server:[%s]\n", SERV_ADDR);  
  
//下面客户端和服务端互相收发  
while(1)  
{  
    //3. 使用 send 函数发送数据  
    printf("Please enter the required documents: \n");  
    scanf("%s", send_buf);  
    if(!strcmp(send_buf, "+++"))break;    //输入+++客户端断开连接  
    fp = fopen(send_buf, "w+");  
    if (fp == NULL)  
    {  
        printf("File:\t%s Can Not Open To Write!\n", send_buf);  
        _exit(-1);  
    }  
    //printf("File:\t%s Open Success,Waiting To Write...\n", send_buf);  
  
    ret = send(sockfd, send_buf, strlen(send_buf), 0);  
    //printf("send buffer:%s, send len:%d\n", send_buf, ret);  
  
    //4. 使用 recv 函数接收来自服务端的消息  
    ret = recv(sockfd, recv_buf, sizeof(recv_buf), 0);  
    if(ret < 1)  
    {  
        printf("The server is disconnected\n");  
        break;  
    }  
    //printf("recv buffer:\n%s\nrecv len:%d\n", recv_buf, ret);  
    //printf("将接收到的数据写入文件中:\n");  
    //调用 fwrite 函数将 recv_buf 缓存中的数据写入文件中  
    int write_length = fwrite(recv_buf, sizeof(char), ret, fp);  
  
    if (write_length < ret)  
    {
```

```

        printf("File write failed!\n");
        break;
    }
    printf("Recieve File:\n");
    //printf("Recieve File:\t%s From Server[%s] Received successfully!\n", send_buf,
SERV_ADDR);

    fseek(fp, 0, SEEK_END);
    size_t size = ftell(fp);
    printf("%Zu bytes recieved, and stored in %s\n", size, send_buf);
    memset(send_buf,0,sizeof(send_buf)); //清空接收缓存区
    memset(recv_buf,0,sizeof(recv_buf)); //清空接收缓存区
    fclose(fp);
}
printf("Close the connection and exit\n");
close(sockfd); //关闭 socket 文件描述符
return 0;
}

```

## 2. 服务器端代码

```

//
// server.c
//
#include<netinet/in.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<arpa/inet.h>
#include<unistd.h>

#define PORT        34000        //定义服务器端口
#define SERVER      "127.0.0.1"  //定义服务器的 IP 地址

#define LENGTH_OF_LISTEN_QUEUE    20
#define BUFFER_SIZE                1048576
#define FILE_NAME_MAX_SIZE        512

int main(int argc, char **argv)
{
    FILE *fp = NULL;

    // set socket's address information
    // 设置一个 socket 地址结构 server_addr,代表服务器 internet 的地址和端口

```

```

struct sockaddr_in server_addr;
bzero(&server_addr, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr(SERVER);
server_addr.sin_port = htons(PORT);

// create a stream socket
// 创建用于 internet 的流协议(TCP)socket, 用 server_socket 代表服务器向客户端提供服务的接口
int server_socket = socket(PF_INET, SOCK_STREAM, 0);
if (server_socket < 0)
{
    printf("Create Socket Failed!\n");
    exit(1);
}

printf("Create Socket Success!\n");

//注意: 由 TCP 套接字状态 TIME_WAIT 引起在结束本次会话后 close 立刻开启下次会话会 Bind 失败。
//该状态在套接字关闭后约保留 2 到 4 分钟。在 TIME_WAIT 状态退出之后, 套接字被删除, 该地址才能被
重新绑定而不出问题。

//因此下面两句话的加入可以解决这个问题
int on = 1;
setsockopt( server_socket, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on) );

// 把 socket 和 socket 地址结构绑定
if (bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)))
{
    printf("Binded success!\n");
    exit(1);
}

printf("Create Socket Success!\n");

// server_socket 用于监听
if (listen(server_socket, LENGTH_OF_LISTEN_QUEUE))
{
    printf("Server Listen Failed!\n");
    exit(1);
}

printf("server listen...\n");

// 定义客户端的 socket 地址结构 client_addr, 当收到来自客户端的请求后, 调用 accept
// 接受此请求, 同时将 client 端的地址和端口等信息写入 client_addr 中
struct sockaddr_in client_addr;
socklen_t length = sizeof(client_addr);

```

```

// 接受一个从 client 端到达 server 端的连接请求,将客户端的信息保存在 client_addr 中
// 如果没有连接请求,则一直等待直到有连接请求为止,这是 accept 函数的特性,可以
// 用 select()来实现超时检测
// accpet 返回一个新的 socket,这个 socket 用来与此次连接到 server 的 client 进行通信
// 这里的 new_server_socket 代表了这个通信通道
int new_server_socket = accept(server_socket, (struct sockaddr*)&client_addr, &length);
if (new_server_socket < 0)
{
    printf("Server Accept Failed!\n");
    exit(1);
}
printf("The client is successfully connected,new_server_socket = %d\n",new_server_socket);
printf("Client ip = %s\n",inet_ntoa(client_addr.sin_addr));
printf("Client port = %d\n",ntohs(client_addr.sin_port));

// 服务器端一直运行用以持续为客户端提供服务
char buffer[BUFFER_SIZE];
while(1)
{
    bzero(buffer, sizeof(buffer));
    printf("Waiting...: \n");
    length = recv(new_server_socket, buffer, BUFFER_SIZE, 0);
    if(length > 0)
    {
        printf("Recieve messeage from client:'%s'\n", buffer);
    }
    else
    {
        printf("The client disconnects, exit! ! ! \n");
        break;
    }
    char file_name[FILE_NAME_MAX_SIZE + 1];
    bzero(file_name, sizeof(file_name));
    strncpy(file_name, buffer,strlen(buffer) > FILE_NAME_MAX_SIZE ? FILE_NAME_MAX_SIZE :
strlen(buffer));

    fp = fopen(file_name, "r");
    if (fp == NULL)
    {
        printf("File:\t%s Not Found!\n", file_name);
    }
    else
    {
        printf("File:\t%s open success!\n", file_name);
    }
}

```



```

bzero(buffer, BUFFER_SIZE);
int file_block_length = 0;
//循环将文件 file_name(fp)中的内容读取到 buffer 中
while( (file_block_length = fread(buffer, sizeof(char), BUFFER_SIZE, fp)) > 0)
{
    printf("file_block_length = %d\n", file_block_length);

    // 发送 buffer 中的字符串到 new_server_socket, 实际上就是发送给客户端
    if (send(new_server_socket, buffer, file_block_length, 0) < 0)
    {
        printf("Send File:\t%s Failed!\n", file_name);
        break;
    }

    //清空 buffer 缓存区
    bzero(buffer, sizeof(buffer));
}
fclose(fp); //关闭文件描述符 fp
printf("File:\t%s Transfer Finished!\n", file_name);
}
}

close(new_server_socket); //关闭 accept 文件描述符
close(server_socket); //关闭 socket 文件描述符
return 0;
}

```