

吉 林 大 学

软件学院

实 验 报 告

实验名称	原始套接字编程				
课程名称	计算机网络课程设计				
姓名	霍鑫壮	学号	13200430	成绩	
提交日期	2023/4/6	座位号			

1. 实验目的

掌握原始套接字编程。

2. 实验内容

1, 利用 RAW SOCKET 捕获网络数据包的程序模型

2, 能够抓取并发服务器程序的服务器端或客户端的应用层数据, 即: 时间值, 打印输出。

3. 实验分析

本实验需要实现原始套接字编程, 在之前客户端服务器正常通信的基础上额外写一个原始套接字的程序, 在客户端服务器运行过程中同时运行原始套接字程序, 监听前两者通信过程中的数据, 接收数据并打印出来。

4. 问题解答

服务器端:

1, 首先我们分离出第一个功能, 创建服务器端套接字 (SOCKET createServerSocket()), 在该函数中, 我们创建套接字, 绑定套接字以及套接字开始监听。

2, 我们将主线程需要的功能分离出函数 (SOCKET mainThread(SOCKET Sock)), 在该函数中, 我们进行一个 while 循环, 在每次循环中都接收客户端请求, 实现多线程功能。同时创建子线程, 子线程进行固定的线程函数。

3, 子线程进行的线程函数 (DWORD WINAPI clientChildThread(LPVOID ipParameter)) 进行主要的与客户端通信流程, 在其中我们不断地进行 while 循环, 接收来自客户端的信息, 根据指令的不同调用不同的函数与客户端进行通信。

客户端:

1, 首先我们分离出第一个功能, 创建客户端套接字, 与服务器端建立连接 (SOCKET createClientSocket()), 在该函数中, 我们创建套接字, 连接服务器端套接字。

2, 将客户端与服务器通信的功能分离出来 (int

talkSocket(SOCKET Sock)), 在该函数中, 客户端与服务器正常调用 send, recv 通信。

原始套接字:

1, 首先我们分离出第一个功能, 创建原始套接字 (SOCKET createSocket()), 在该函数中, 我们创建原始套接字, 设置套接字选项, 绑定 IP 和端口号, 将网卡设置为混听模式。

2, 将原始套接字接收信息功能分离出来 (void listen(SOCKET Sock)), 在该函数中, 我们监听客户端与服务器端的通信, 接收数据并打印出来。

4.3 核心代码 (有必要的注释)

服务器端:

```
//int main()
//功能说明: 程序 main 函数, 执行全部过程
//参数说明: 无传入参数, 无返回值
int main() {
    SOCKET Sock = createServerSocket();
    if (Sock == NULL) {
        printf("error createSocket");
        return 0;
    }
    //初始化结构体数组
    for (int i = 0; i < BUFSIZ; i++) {
        memset(&infos[i], 0, sizeof(infos[i]));
        infos[i].socket = -1;
    }
    SOCKET clifd = mainThread(Sock);
    if (clifd == NULL) {
        printf("error mainThread");
        return 0;
    }
    closesocket(clifd);
    closesocket(Sock);
    int ret2 = stopServerMain();
    if (ret2 == -1) {
        printf("error stopMain");
        return 0;
    }
    return 0;
```

```
}
//DWORD WINAPI clientChildThread(LPVOID ipParameter)
//功能说明：子线程函数，所有创建的子线程都要执行的功能
//参数说明：线程函数的参数是任意类型，进入内部强转即可，
返回类型中 WINAPI 是一个宏，所代表的符号是__stdcall，函数名前
加上这个符号表示这个函数的调用约定是标准调用约定，windows
API 函数采用这种调用约定。
DWORD WINAPI clientChildThread(LPVOID ipParameter) {
    struct SockInfo* pinfo = (struct SockInfo*) ipParameter;
    char buf[20] = { 0 };
    //inet_ntop 将一个十进制网络字节序转换为点分十进制 IP
    格式的字符串。
    inet_ntop(AF_INET, &pinfo->addr.sin_addr, buf,
sizeof(buf));
    //ntohs 将一个 16 位数由网络字节顺序转换为主机字节顺序
    cout << "客户端 IP: " << buf << "端口: " <<
ntohs(pinfo->addr.sin_port) << "连接成功" << endl;
    char recvbuf[BUFSIZ] = { 0 };
    char sendbuf[BUFSIZ] = { 0 };
    while (true) {
        memset(recvbuf, 0, sizeof(recvbuf));
        if (0 < recv(pinfo->socket, recvbuf, BUFSIZ, 0)) {
            //成功了
            printf("recv:%s\n", recvbuf);
        }
        else {
            closesocket(pinfo->socket);
            pinfo->socket = -1;
            break;
        }
        if (strcmp(recvbuf, "当前时间") == 0) {
            time_t now = time(0); //把 now 转换为字符串形式
            ctime_s(sendbuf, BUFSIZ, &now);
            if (SOCKET_ERROR == send(pinfo->socket, sendbuf,
strlen(sendbuf), 0)) {
                printf("server send");
                break;
            }
        }
    }
}
```

```
        memset(sendbuf, 0, sizeof(sendbuf));
    }
    else if (strcmp(recvbuf, "退出连接") == 0) {
        memset(sendbuf, 0, sizeof(sendbuf));
        closesocket(pinfo->socket);
        pinfo->socket = -1;
        cout << "端口为" << ntohs(pinfo->addr.sin_port)
<< "的客户端退出连接成功" << endl;
        break;
    }
    else {
        strcpy_s(sendbuf, BUFSIZ, recvbuf);
        if (SOCKET_ERROR == send(pinfo->socket, sendbuf,
strlen(sendbuf), 0)) {
            printf("server send");
            break;
        }
        memset(sendbuf, 0, sizeof(sendbuf));
    }
}
return NULL;
}
```

客户端:

```
//int main()
//功能说明: 程序main函数, 执行全部过程
//参数说明: 无传入参数, 无返回值
int main() {
    SOCKET Sock = createClientSocket();
    if (Sock == NULL) {
        printf("error createClientSocket");
        return 0;
    }
    int ret1 = talkSocket(Sock);
    if (ret1 == -1) {
        printf("error connectSocket");
        return 0;
    }
    int ret2 = stopClientMain();
}
```

```
        if (ret2 == -1) {
            printf("error stopClientMain");
            return 0;
        }
        return 0;
    }
//int talkSocket(SOCKET Sock)
//功能说明：与服务器端通信的主要函数
//参数说明：Sock 为与服务器端通信的套接字，返回 int 类型
数据
int talkSocket(SOCKET Sock) {
    char recvbuf[BUFSIZ] = { 0 };
    char sendbuf[BUFSIZ] = { 0 };
    while (true) {
        //发送消息
        printf("send>");
        memset(sendbuf, 0, sizeof(sendbuf));
        gets_s(sendbuf, BUFSIZ);
        if (SOCKET_ERROR == send(Sock, sendbuf,
strlen(sendbuf), 0)) {
            printf("server send");
            closesocket(Sock);
            return -1;
        }
        if (0 < recv(Sock, recvbuf, BUFSIZ, 0)) {
            printf("recv:%s\n", recvbuf);
            memset(recvbuf, 0, sizeof(recvbuf));
        }
        else {
            closesocket(Sock);
            return -1;
        }
    }
    getchar();
    return 0;
}
原始套接字：
//void listen(SOCKET Sock)
```

```

//功能说明：原始套接字接收两端通信内容函数
//参数说明：Sock 为原始套接字，无返回值
void listen(SOCKET Sock) {
    memset(buffer, 0, BUFFERLENGTH);
    bytesRecv = recvfrom(Sock, buffer, BUFFERLENGTH, 0,
(struct sockaddr*)&from, &fromSize);
    if (bytesRecv == SOCKET_ERROR) {
        printf("error recvfrom");
        WSAGetLastError();
        closesocket(Sock);
        WSACleanup();
        getchar();
        return;
    }
    ip = (struct IP*)buffer;
    if (ip->protocol == 6) { //过滤其他协议，只留下 TCP
协议
        char SAddrbuf[20] = { 0 };
        char DAddrbuf[20] = { 0 };
        inet_ntop(AF_INET, &ip->sourceAddr, SAddrbuf,
sizeof(SAddrbuf));
        inet_ntop(AF_INET, &ip->destinAddr, DAddrbuf,
sizeof(DAddrbuf));
        tcp = (struct TCP*)(buffer + (4 * ip->headLen &
0xf0 >> 4)); //得到 TCP 头
        cout << "\n\n\n";
        cout <<
"Network+++++++
+++++\n"; //网络层数据
        cout << "IP 报文字节数: " << bytesRecv << "\n";
        cout << "源 IP: " << SAddrbuf << "\n";
        cout << "目的 IP: " << DAddrbuf << "\n";
        cout <<
"Transportation+++++++
+++++\n"; //运输层数据
        cout << "源端口: " << ntohs(tcp->sourcePort) <<
"\n";
        cout << "目的端口: " << ntohs(tcp->destinPort) <<

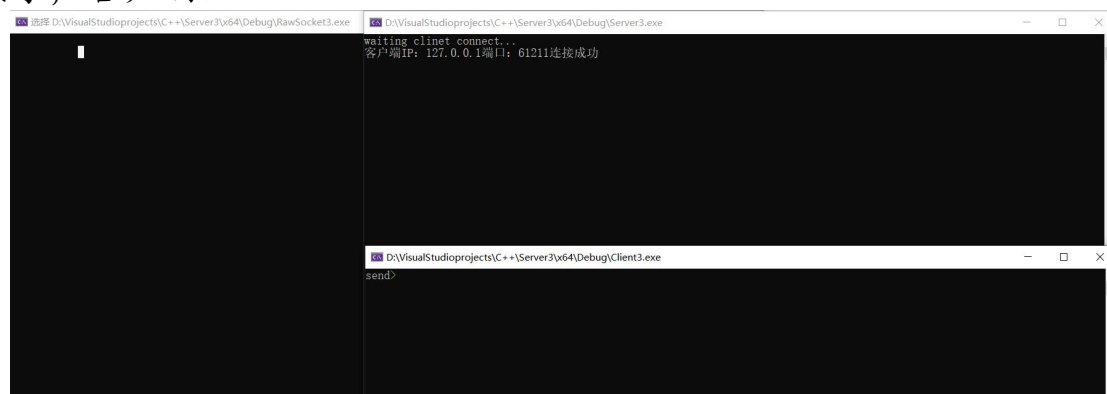
```

```
"\n";

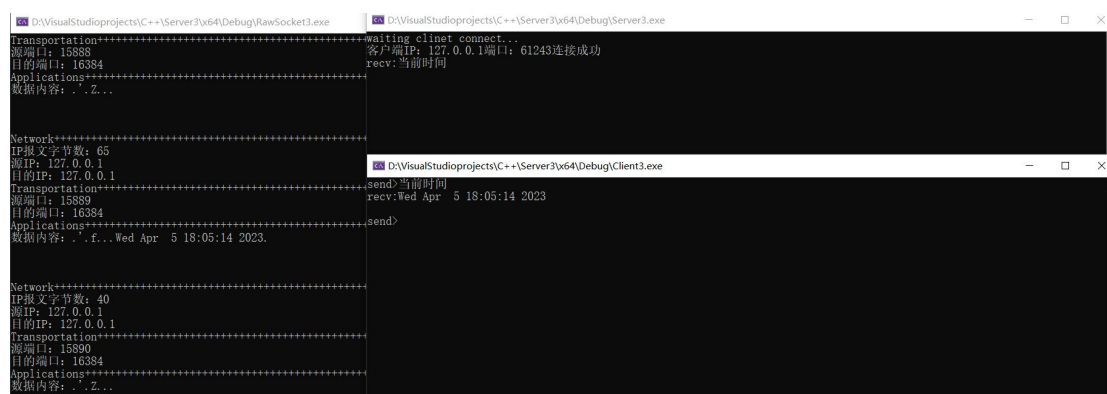
    cout <<
    "Applications++++++++\n";
    char* start = buffer + 5 + 4 * ((tcp->headLen &
    0xf0) >> 4 | 0); //计算数据头指针, 从何处开始数据
    int dataSize = bytesRecv - 5 - 4 * ((tcp->headLen
    & 0xf0) >> 4 | 0); //计算数据长度
    cout << "数据内容: ";
    memcpy(buffer, start, dataSize);
    for (int i = 0; i < dataSize; i++) {
        if (buffer[i] >= 32 && buffer[i] < 255) {
            printf("%c", (unsigned char)buffer[i]);
        }
        else {
            printf(". ");
        }
    }
    cout << "\n";
}
```

4.4 测试方法、测试数据与测试结果

1, 首先我们测验程序主要功能是否正确, 打开服务器端, 原始套接字, 客户端:



2, 客户端与服务器端通信:



```
D:\VisualStudioProjects\C++\Server3\Debug\RawSocket3.exe
Transportation+++++
源端口: 15888
目的端口: 16384
数据内容: ..Z...

Network+++++
IP报文字节数: 65
源IP: 127.0.0.1
目的IP: 127.0.0.1
Transportation+++++
源端口: 15889
目的端口: 16384
Applications+++++
数据内容: ..f...Wed Apr 5 18:05:14 2023.

Network+++++
IP报文字节数: 40
源IP: 127.0.0.1
目的IP: 127.0.0.1
Transportation+++++
源端口: 15890
目的端口: 16384
Applications+++++
数据内容: ..Z...

D:\VisualStudioProjects\C++\Server3\Debug\Server3.exe
waiting c\inet connect...
客户端IP: 127.0.0.1端口: 61243连接成功
recv: 当前时间

D:\VisualStudioProjects\C++\Server3\Debug\Client3.exe
send>当前时间
recv:Wed Apr 5 18:05:14 2023
send>
```

4.5 程序的使用说明

当使用时，先打开服务器端，然后打开客户端，最后打开原始套接字。若在客户端输入“当前时间”，点击回车即可获得服务器传输回来的时间，同时可在原始套接字查看双方通信内容。若在客户端输入“退出连接”，点击回车客户端即与服务器退出连接。

4.6 总结

该程序运行效果简洁高效，程序运行效果符合实验题目要求，满足完整性。

遇到的问题：

1，如何实现原始套接字接收数据？使用 `recvfrom` 函数，通过对 `buff` 强转为 `ip` 结构体获取 `TCP` 头和数据部分。

2，客户端服务器之间如何实现通信？本实验通过两者 `send`, `recv` 函数调用，服务器端将对应时间发送给客户端。

吉 林 大 学

软件学院

实 验 报 告

实验名称	FTP 综合应用编程				
课程名称	计算机网络课程设计				
姓名	霍鑫壮	学号	13200430	成绩	
提交日期	2023/4/6	座位号			

1. 实验目的

掌握 FTP 应用编程。

2. 实验内容

在前三个实验的基础（简单套接字，并发套接字，原始套接字）上，将其改造成一个能传输指定文件名称的点对点文件传输软件。简要实验流程为：客户端等待键盘输入文件名称，然后将文件名称传输给服务器，服务器在预先设置好的文件夹下查找该文件，如果发现同名文件，开始传输回客户端，客户端接收完文件后将文件以输入的文件名称保存在本地某个目录即可，否则告诉客户端文件不存在。

3. 实验分析

本实验是综合应用实验，我们需要分别编写服务器端和客户端代码，在服务器端我们需要实现的功能有：创建套接字，在主线程中随时创建子线程，每一个子线程中都要进行与客户端通信，完成文件传输等功能。在客户端我们需要实现的功能有：创建套接字，与服务器端套接字连接，进入主循环完成相互通信过程。

4. 问题解答

服务器端：

1，首先我们分离出第一个功能，创建服务器端套接字（`SOCKET createServerSocket()`），在该函数中，我们创建套接字，绑定套接字以及套接字开始监听。

2，我们将主线程需要的功能分离出函数（`SOCKET mainThread(SOCKET Sock)`），在该函数中，我们进行一个 `while` 循环，在每次循环中都接收客户端请求，实现多线程功能。同时创建子线程，子线程进行固定的线程函数。

3，子线程进行的线程函数（`DWORD WINAPI clientChildThread(LPVOID ipParameter)`）进行主要的与客户端通信流程，在其中我们不断地进行 `while` 循环，接收来自客户端的信息，根据指令的不同调用不同的函数与客户端进行通信。

4，指令为 `list` 时分离出功能（`int List(struct SockInfo*`

pinfo)), 在该函数中, 实现传输给客户端服务器端文件夹中所有文件名称的功能。

5, 指令为 down 时分离出功能 (int Down(struct SockInfo* pinfo)), 在该函数中, 实现服务器传输给客户端指定的文件内容的功能, 文件由客户端指定。

6, 指令为 quit 时分离出功能 (void Quit(struct SockInfo* pinfo)), 在该函数中, 实现服务器与该客户端断开连接的功能。

7, 将获取指定目录下所有文件名的功能分离出函数 (string getFiles(string path))。

8, 将用于判断指定文件名是否存在于目录的功能分离出函数 (BOOL IsExist(string filename))。

9, 将读取文件的操作的功能分离出函数 (string readFile(string filename))。

客户端:

1, 首先我们分离出第一个功能, 创建客户端套接字, 与服务器端建立连接 (SOCKET createClientSocket()), 在该函数中, 我们创建套接字, 连接服务器端套接字。

2, 我们将客户端主循环分离出函数 (void mainWhile(SOCKET Sock)), 在该函数中, 我们进行一个 while 循环, 在每次循环中都读取客户端用户输入的指令, 根据指令的不同调用不同的函数。

3, 指令为 list 时分离出功能 (void List(SOCKET Sock)), 在该函数中, 实现接收服务器传回的数据, 打印出所有文件名的功能。

4, 指令为 down 时分离出功能 (void Down(SOCKET Sock)), 在该函数中, 实现与服务器通信发出希望下载的文件名, 接收服务器传回的数据, 打印文件内容并调用 createfile 函数的功能。

5, 指令为 quit 时分离出功能 (void Quit(SOCKET Sock)), 在该函数中, 实现退出连接, 退出系统的功能。

6, 将用于判断指定文件名是否存在于目录的功能分离出函数 (BOOL IsExist(string filename))。

7, 将在客户端文件夹内创建文件并写入数据的功能分离出函数 (void createfile(string filename))。

4.3 核心代码 (有必要的注释)

服务器端:

```
//int main()
//功能说明: 程序 main 函数, 执行全部过程
//参数说明: 无传入参数, 无返回值
int main() {
    SOCKET Sock = createServerSocket();
```

```
    if (Sock == NULL) {
        printf("error createSocket");
        return 0;
    }
    initializeArr();
    SOCKET clifd = mainThread(Sock);
    if (clifd == NULL) {
        printf("error mainThread");
        return 0;
    }
    closesocket(clifd);
    closesocket(Sock);
    if (stopServerMain() == -1) {
        printf("error stopMain");
        return 0;
    }
    return 0;
}

//SOCKET createServerSocket()
//功能说明：用于创建服务器的套接字
//参数说明：无传入参数，返回 SOCKET 类型变量，即服务器的套接字

SOCKET createServerSocket() {
    //使用 Socket 的程序在使用 Socket 之前必须调用 WSStartup
    函数。以后应用程序就可以调用所请求的 Socket 库中的其它 Socket
    函数了，然后绑定找到的 Socket 库到该应用程序中。该函数执行成
    功后返回 0。
    WSADATA wsaData;
    if (0 != WSStartup(MAKEWORD(2, 2), &wsaData)) {
        printf("WSStartup failed code %d",
WSAGetLastError());
        return NULL;
    }
    //创建套接字
    SOCKET Sock = socket(AF_INET, SOCK_STREAM,
IPPROTO_TCP);
    if (Sock == INVALID_SOCKET) { //失败了
        printf("error socket");
    }
}
```

```

        return NULL;
    }
    //绑定套接字
    sockaddr_in sockAddr;
    memset(&sockAddr, 0, sizeof(sockAddr)); //每个字节都用
0 填充
    sockAddr.sin_family = AF_INET;
    //inet_pton() 将点分十进制的 IP 地址转换成二进制的
    inet_pton(AF_INET, "127.0.0.1", &sockAddr.sin_addr);
//具体的 IP 地址
    //htons() 是网络字节序与主机字节序之间转换的函数。用生
活中的例子来说，有一串数字 12345678 现在我们是左往右读的，
以前的人是从右往左读的。当你要给以前的人读的话就要把这串数据
写成 87654321 。htons() 就是类似要完成这个转换的功能
    sockAddr.sin_port = htons(PORT); //端口
    if (SOCKET_ERROR == bind(Sock, (sockaddr*)&sockAddr,
sizeof(sockAddr))) {
        printf("bind");
        return NULL;
    }
    listen(Sock, 20); //开始监听
    printf("waiting client connect...\n");
    return Sock;
}
//SOCKET mainThread(SOCKET Sock)
//功能说明：主线程中执行的操作，其中可能会创建子线程，进
入 clientChildThread 函数
//参数说明：Sock 是服务器的套接字，返回与子线程中与客户
端连接的套接字
SOCKET mainThread(SOCKET Sock) {
    SOCKET clifd = NULL;
    int addrsz = sizeof(sockaddr_in);
    HANDLE pThread = NULL;
    while (true) { //循环，用于多线程
        struct SockInfo* pinfo = NULL;
        for (int i = 0; i < BUFSIZ; i++) { //为了找到空闲的
线程资源
            if (infos[i].socket == -1) {

```

```
        pinfo = &infos[i];
        break;
    }
}
if (pinfo == NULL) { //BUFSIZ 已经全部用完
    printf("error FullArr");
    return NULL;
}
else {
    //如果服务器应用层需要用到客户端的 IP 和端口号,
    可以给 accept 指定第二个参数 addr, 以获取 TCP 链接时的客户端 ip
    和端口号; 如果服务器应用层不需要, 则写 NULL 即可
    clifd = accept(Sock, (struct
sockaddr*)&pinfo->addr, &addrsz);
    pinfo->socket = clifd;
    if (INVALID_SOCKET == clifd) {
        printf("error accept");
        return NULL;
    }
    else {
        /*
        //CreateThread
        //lpSa: 线程句柄的安全性, 比如子进程是否可以
        继承这个线程句柄, 一般设置为 NULL
        //cbStack: 线程栈大小, 一般取 0 表示默认大小
        //lpStartAddr: 线程入口函数
        //lpvThreadParam: 线程入口函数的参数
        //fdwCreate: 控制线程创建的标志, 一般为 0, 表
        示线程立即启动。也可以选择可以挂起, 使用 CREATE_SUSPENDED,
        之后在代码中使用 ResumeThread 启动。
        //lpIDThread: 线程的 ID 值, 接收线程返回的 ID
        */
        pThread = CreateThread(NULL, 0,
clientChildThread, (LPVOID)pinfo, 0, NULL);
        if (pThread == NULL) {
            printf("error CreateThread");
            break;
        }
    }
}
```

```

        CloseHandle(pThread);
    }
}

return clifd;
}

//DWORD WINAPI clientChildThread(LPVOID ipParameter)
//功能说明：子线程函数，所有创建的子线程都要执行的功能
//参数说明：线程函数的参数是任意类型，进入内部强转即可，
返回类型中 WINAPI 是一个宏，所代表的符号是__stdcall，函数名前
加上这个符号表示这个函数的调用约定是标准调用约定，windows
API 函数采用这种调用约定。
DWORD WINAPI clientChildThread(LPVOID ipParameter) {
    struct SockInfo* pinfo = (struct SockInfo*) ipParameter;
    char buf[20] = { 0 };
    //inet_ntop 将一个十进制网络字节序转换为点分十进制 IP
    格式的字符串。
    inet_ntop(AF_INET, &pinfo->addr.sin_addr, buf,
sizeof(buf));
    //ntohs 将一个 16 位数由网络字节顺序转换为主机字节顺序
    cout << "客户端 IP: " << buf << " 端口: " <<
ntohs(pinfo->addr.sin_port) << " 的客户端连接成功" << endl;
    while (true) {
        memset(recvbuf, 0, sizeof(recvbuf));
        if (0 < recv(pinfo->socket, recvbuf, BUFSIZ, 0)) {//
成功了
            printf("端口为 %d 的客户端发来的请求: %s\n",
ntohs(pinfo->addr.sin_port), recvbuf);
        }
        else {
            closesocket(pinfo->socket);
            pinfo->socket = -1;
            break;
        }
    }
    //分辨命令
    if (strcmp(recvbuf, "list") == 0) {
        if (List(pinfo) == -1) {
            break;
        }
    }
}

```

```
    }
}
else if(strcmp(recvbuf, "down") == 0) {
    if (Down(pinfo) == -1) {
        break;
    }
}
else if (strcmp(recvbuf, "quit") == 0) {
    Quit(pinfo);
    break;
}
}
return NULL;
}
```

客户端:

```
//int main()
//功能说明: 程序 main 函数, 执行全部过程
//参数说明: 无传入参数, 无返回值
int main() {
    SOCKET Sock = createClientSocket();
    if (Sock == NULL) {
        printf("error createClientSocket");
        return 0;
    }
    mainWhile(Sock);
    if (stopClientMain() == -1) {
        printf("error stopClientMain");
        return 0;
    }
    return 0;
}

//SOCKET createClientSocket()
//功能说明: 用于创建与服务器通信的套接字
//参数说明: 无传入参数, 返回 SOCKET 类型变量, 即与服务器
建立连接的套接字
SOCKET createClientSocket() {
    //使用 Socket 的程序在使用 Socket 之前必须调用 WSASStartup
函数。以后应用程序就可以调用所请求的 Socket 库中的其
```

//它 Socket 函数了,然后绑定找到的 Socket 库到该应用程序中。该函数执行成功后返回 0。

```
WSADATA wsaData;
if (0 != WSStartup(MAKEWORD(2, 2), &wsaData)) {
    printf("WSStartup failed code %d",
WSAGetLastError());
    return NULL;
}
//创建套接字
SOCKET Sock = socket(AF_INET, SOCK_STREAM,
IPPROTO_TCP);
if (Sock == INVALID_SOCKET) { //失败了
    printf("error socket");
    return NULL;
}
//连接套接字
sockaddr_in sockAddr;
sockAddr.sin_family = AF_INET;
//inet_addr()将点分十进制的 IP 地址转换成二进制的
inet_pton(AF_INET, "127.0.0.1", &sockAddr.sin_addr);
//具体的 IP 地址
//sockAddr.sin_addr.s_addr = ADDR_ANY; //本机任意网卡
任意 IP 地址
//htons()是网络字节序与主机字节序之间转换的函数。用生
活中的例子来说,有一串数字 12345678 现在我们是左往右读的,
以前的人是从右往左读的。当你要给以前的人读的话就要把这串数据
写成 87654321。htons()就是类似要完成这个转换的功能
sockAddr.sin_port = htons(PORT); //端口
if (INVALID_SOCKET == connect(Sock,
(sockaddr*)&sockAddr, sizeof(sockAddr))) { //连接失败
    printf("error connect");
    return NULL;
}
return Sock;
}
//void mainWhile(SOCKET Sock)
//功能说明:用于客户端与服务器之间主要循环过程
//参数说明:Sock 是与服务器建立连接的套接字,无返回值
```



```
void mainWhile(SOCKET Sock) {
    //连接成功后，首先打印出该系统界面
    printf("欢迎进入FTP 综合应用系统！\n");
    printf("你可以使用的命令有：\n");
    printf("list\t<展示服务器当前所在目录的全部文件（夹）\n");
    printf("down\t<从服务器下载文件，参数为文件名>\n");
    printf("quit\t<当前客户端退出系统，无需参数>\n");
    //主循环，进行客户端操作
    while (true) {
        printf("send>");
        scanf_s("%s", cmd, CMDSIZ);
        switch (cmd[0]) {
            case 'l':
                List(Sock);
                break;
            case 'd':
                Down(Sock);
                break;
            case 'q':
                printf("本客户端退出系统。");
                Quit(Sock);
                break;
            default:
                printf("不存在的命令，请输入有效的命令。");
                break;
        }
        if (cmd[0] == 'q') {
            break;
        }
    }
    return;
}

//void createfile(string filename)
//功能说明：在客户端文件夹内创建文件并写入数据
//参数说明：filename 是希望创建的文件名，无返回值
void createfile(string filename) {
    string filepath =
```

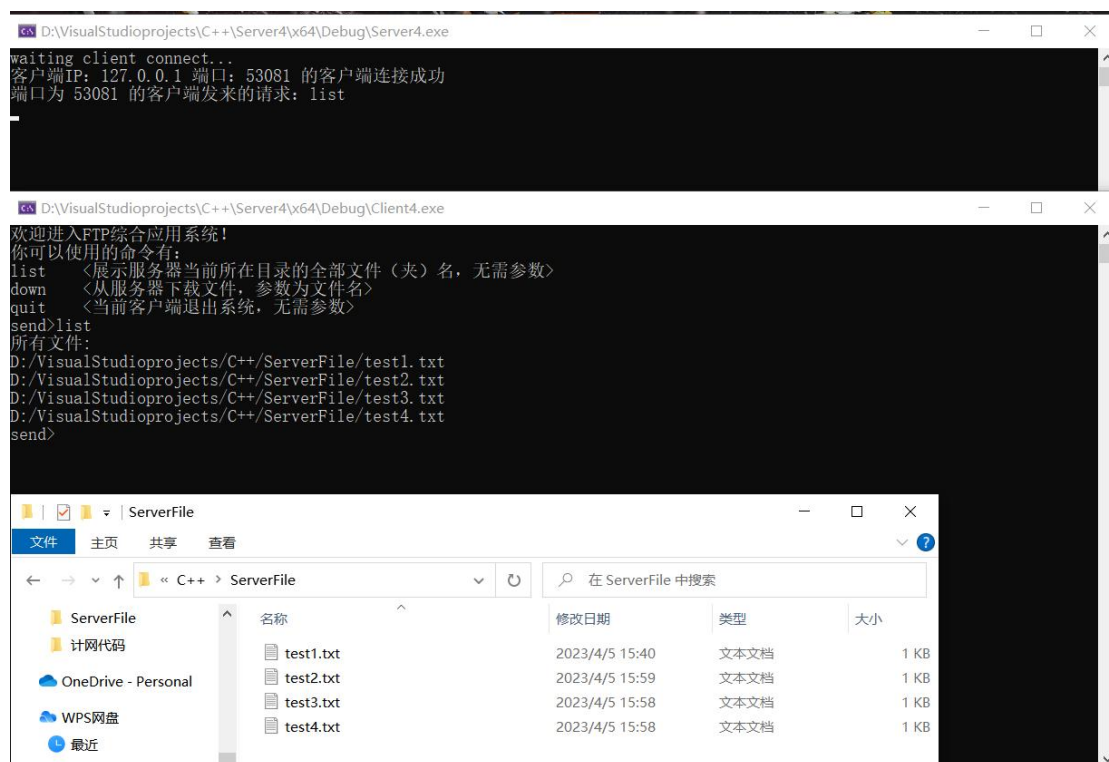
```
"D:/VisualStudioprojects/C++/ClientFile/" + filename;
    ofstream location_out;
    //以写入和在文件末尾添加的方式打开.txt 文件，没有的话
    就创建该文件。
    location_out.open(filepath, std::ios::out |
std::ios::app);
    if (!location_out.is_open()) {
        return;
    }
    location_out << recvbuf << endl;
    location_out.close();
}
```

4.4 测试方法、测试数据与测试结果

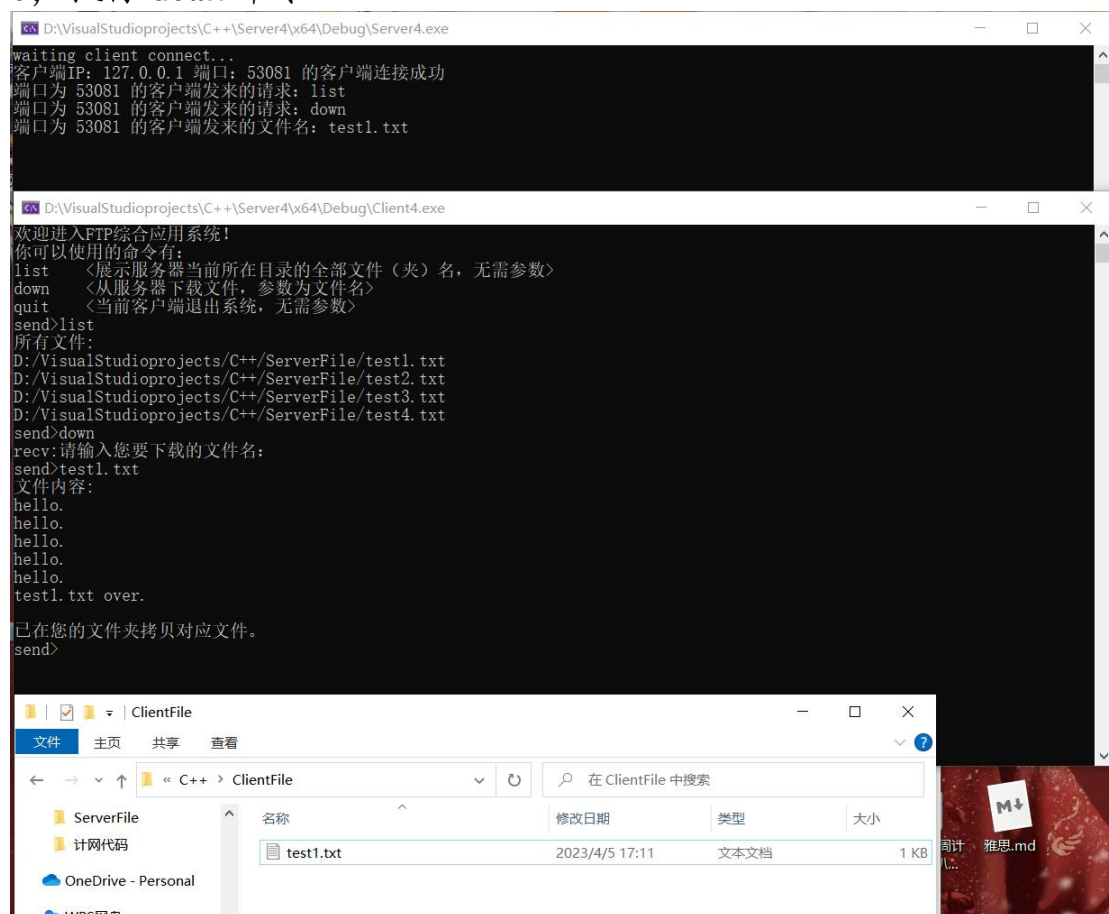
1, 首先我们测验程序主要功能是否正确, 打开服务器端, 客户端:



2, 执行 list 命令:



3, 执行 down 命令



4, 执行错误的 down 命令:

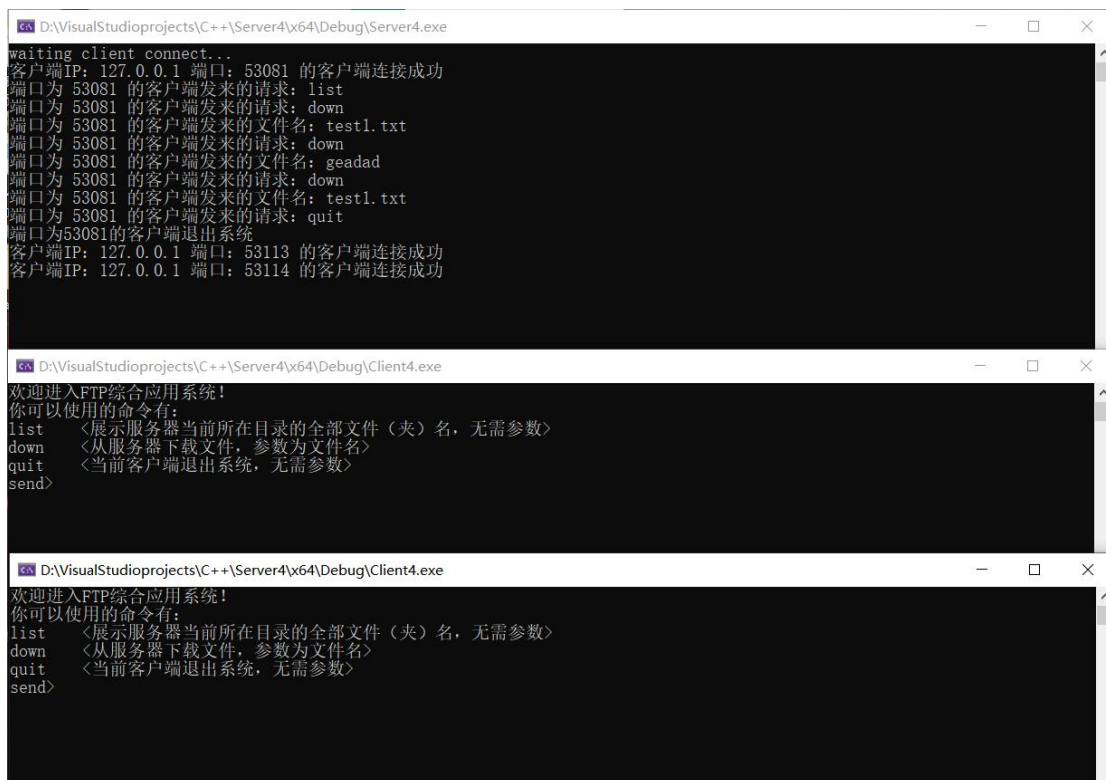
```
send>down
recv:请输入您要下载的文件名:
send>geadad
recv:文件不存在
send>down
recv:请输入您要下载的文件名:
send>test1.txt
文件内容:
hello.
hello.
hello.
hello.
hello.
test1.txt over.

在您的文件夹已有该文件，不会创建新的文件。
send>
```

5, 执行 quit 命令:

```
D:\VisualStudio\projects\C++\Server4\Debug\Server4.exe
waiting client connect...
客户端IP: 127.0.0.1 端口: 53081 的客户端连接成功
端口为 53081 的客户端发来的请求: list
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: test1.txt
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: geadad
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: test1.txt
端口为 53081 的客户端发来的请求: quit
端口为53081的客户端退出系统
```

6, 多线程使用, 同时打开两个客户端:

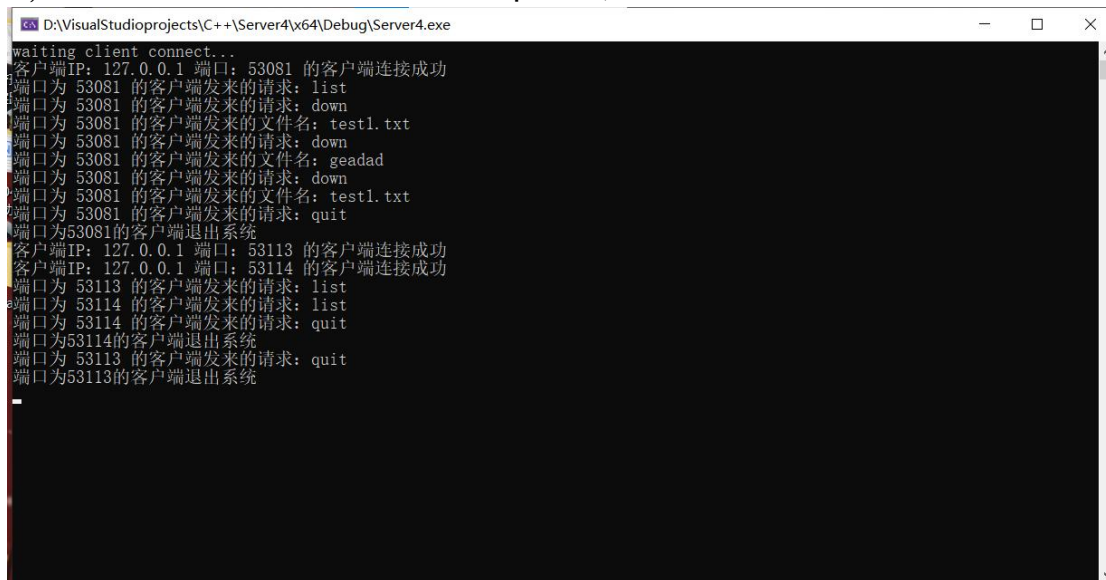


```
D:\VisualStudioprojects\C++\Server4\x64\Debug\Server4.exe
waiting client connect...
客户端IP: 127.0.0.1 端口: 53081 的客户端连接成功
端口为 53081 的客户端发来的请求: list
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: test1.txt
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: geadad
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: test1.txt
端口为 53081 的客户端发来的请求: quit
端口为53081的客户端退出系统
客户端IP: 127.0.0.1 端口: 53113 的客户端连接成功
客户端IP: 127.0.0.1 端口: 53114 的客户端连接成功

D:\VisualStudioprojects\C++\Server4\x64\Debug\Client4.exe
欢迎进入FTP综合应用系统!
你可以使用的命令有:
list    <展示服务器当前所在目录的全部文件(夹)名, 无需参数>
down    <从服务器下载文件, 参数为文件名>
quit    <当前客户端退出系统, 无需参数>
send>

D:\VisualStudioprojects\C++\Server4\x64\Debug\Client4.exe
欢迎进入FTP综合应用系统!
你可以使用的命令有:
list    <展示服务器当前所在目录的全部文件(夹)名, 无需参数>
down    <从服务器下载文件, 参数为文件名>
quit    <当前客户端退出系统, 无需参数>
send>
```

7, 两客户端分别使用 list 和 quit 命令:



```
D:\VisualStudioprojects\C++\Server4\x64\Debug\Server4.exe
waiting client connect...
客户端IP: 127.0.0.1 端口: 53081 的客户端连接成功
端口为 53081 的客户端发来的请求: list
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: test1.txt
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: geadad
端口为 53081 的客户端发来的请求: down
端口为 53081 的客户端发来的文件名: test1.txt
端口为 53081 的客户端发来的请求: quit
端口为53081的客户端退出系统
客户端IP: 127.0.0.1 端口: 53113 的客户端连接成功
客户端IP: 127.0.0.1 端口: 53114 的客户端连接成功
端口为 53113 的客户端发来的请求: list
端口为 53114 的客户端发来的请求: list
端口为 53114 的客户端发来的请求: quit
端口为53114的客户端退出系统
端口为 53113 的客户端发来的请求: quit
端口为53113的客户端退出系统
```

4.5 程序的使用说明

当使用时, 先打开服务器端, 然后打开客户端, 打开客户端之后根据指令不同跳转不同功能: 如果输入“list”, 则客户端接收到服务器端传来的所有文件名; 如果输入“down”, 然后请再次输入文件名, 此时如果文件名是错误的, 则打印错误信息, 如果是正确的, 则会接收到来自服务器端传来的文件内容, 如果文件未接收过, 则接收文件, 否则打印错误信息; 如果输入“quit”, 则客户端退出系统。

4.6 总结

该程序运行效果简洁高效, 可以对错误输入进行抓取, 并打印错

误信息，健壮性较好，并且程序运行效果符合实验题目要求，满足完整性。

遇到的问题：

1，如何实现并发套接字效果？本实验使用的是多线程实现并发效果，通过定义全局变量 `infos` 来限定线程数目。

2，客户端服务器之间如何实现文件传输？本实验通过两者多次 `send`, `recv` 获取指定文件名，在服务器端查找对应文件，并将文件内容通过 `send` 传给客户端，客户端通过函数创建新文件并写入指定内容。

特色说明：

本程序简洁高效的完成了实验内容，通过客户端服务期间指令的传输来调用不同函数不同过程来实现不同功能，同时对异常的处理恰到好处。本程序实现了多线程的并发套接字，实现了多个客户端同时访问服务器端的功能。