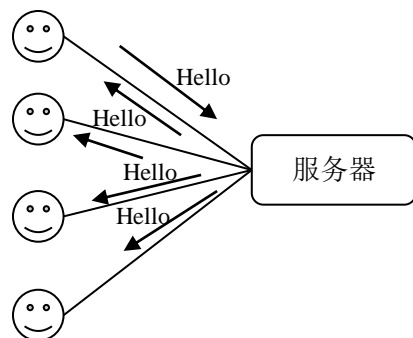


【实验题目】多人聊天编程实验

【实验目的】掌握套接字的多线程编程方法。

【实验介绍】利用客户/服务器(Client/Server 或 CS)模式实现一个多人聊天(群聊)程序。其功能是每个客户发送给服务器的消息都会传送给所有的客户端。



【实验环境】

采用 Windows VC++控制台编程，建立新项目的方法：选择菜单“文件/新建项目/Win32 控制台应用程序/空项目”，然后增加 CPP 空文件，拷贝源程序。集成环境使用 VS2012 或更高版本。

【参考资料】

- <https://docs.microsoft.com/en-us/windows/desktop/WinSock/getting-started-with-winsock> (套接字)
- <https://www.cnblogs.com/hgwang/p/6074038.html> (套接字)
- <https://www.jb51.net/article/37410.htm> (字符串)
- <https://docs.microsoft.com/en-us/cpp/c-runtime-library/stream-i-o?view=vs-2017> (字符串)
- <https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/crt-alphabetical-function-reference?view=vs-2017#s> (字符串)
- <http://www.runoob.com/cprogramming/> (字符串)

【注意事项】

依步骤完成以下实验内容，**尽量多完成一些步骤**。把出现的问题、解决方法和一些思考和体会写在**实验体会**部分；对典型的运行情况的客户和服务器控制台进行截屏并放在**实验结果**部分；截屏用按键(Ctrl+Alt+PrintScreen)单独截取控制台窗口。截屏应尽量 windows 绘图程序缩小尺寸(能看清就行)后粘入。

端口号采用 50500

字符串可以采用函数 `scanf` 或 `gets` 输入。

【参考资料】

- 1、 例程 “_beginthreadex” (创建线程)
- 2、 例程 “TCPServer 和 TCPClient” (传送服务器时间)
- 3、 课件 “套接字并发编程.PDF”
- 4、 Chat 实验的课件

【实验内容】

先阅读课件“套接字并发编程.PDF”。重点是读懂课件中“chat 并发编程(服务器)”和“chat 并发编程(客户端)”的流程图。然后，完成下面步骤(截屏要同时显示服务器和至少两个客户端)：

- (1)编写多人聊天程序，要求客户端和服务器都采用多线程方式进行编程。每个客户端都采用 TCP 协议连接服务器并保持连接。服务器同时与所有客户端建立和保持连接。每个客户端输入的消息都会通过服务器转发给所有客户。

我的思路:

服务器程序中, 定义一个名为 ThreadInfo 的类 (见下方粘贴的源代码, 或项目目录中的 GroupChat_Server/server.h 文件), 其中包含了索引、线程句柄、服务的套接字、客户端地址结构体。创建一个 ThreadInfo 数组 threadinfo, 大小是 MAXSOCKS。服务器开始监听 (listen) 后, 每有一个新的连接 (accept), 就为新的套接字创建一个子线程, 然后把这个新的子线程从 threadinfo 数组中找到一个线程句柄为 NULL 的位置填进去。这个新的子线程负责服务一个客户端, 功能是接收该客户端发来的消息并转发给已经连接的、在 threadinfo 数组中有效的所有客户端 (sendToAll)。服务器的主线程不负任何数据收发操作。

客户端程序中, 有一个主线程和一个子线程。主线程负责接受用户输入并发送给服务器 (send), 子线程负责接收服务器传来的消息 (recv) 并将其打印到屏幕上。在客户端推出后, 要结束子线程并关闭套接字 (closesocket)。

下面附上客户端和服务器的源代码。

客户端程序:

```
#include <stdio.h>
#include <stdlib.h>
#include <WinSock2.h>
#include <string.h>
#include <process.h>
#pragma comment(lib, "ws2_32.lib") // 使用 winsock 2.0 库

const int BUFLen = 2000; // 缓冲区大小

unsigned __stdcall recvThread(void *p);

int main(int argc, char *argv[])
{
    printf("[Group Chat Client]\n群聊客户端正在连接服务器...");

    const char *host = "127.0.0.1"; // server IP to connect
    u_short port = 50500;           // server port to connect
    struct sockaddr_in sin;          // an Internet endpoint address
    char buf[BUFLen + 1];           // buffer for one line of text
    SOCKET sock;                    // socket descriptor
    int cc;                         // recv character count

    WSADATA wsadata;
    WSAStartup(MAKEWORD(2, 0), &wsadata); // 加载 winsock 库

    sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); // 创建套接字, 参数: 因特网协议簇(family), 流套接字, TCP 协议
    memset(&sin, 0, sizeof(sin)); // 从 &sin 开始的长度为 sizeof(sin) 的内存清 0
    sin.sin_family = AF_INET;      // 因特网地址簇
    sin.sin_addr.s_addr = inet_addr(host); // 服务器 IP 地址(32 位)
    sin.sin_port = htons(port);    // 服务器端口号
    if (connect(sock, (struct sockaddr *)&sin, sizeof(sin)) == 0) { // 连接到服务器
        printf("连接成功! \n\n");
        printf("*****\n");
    }
    else {
        printf("失败: %d\n", GetLastError());
        printf("[-] 请检查网络连接, 并确认服务器已启动。按任意键关闭客户端。 \n");
        getchar();
        exit(1);
    }

    HANDLE recv_thread = (HANDLE)_beginthreadex(NULL, 0, &recvThread, (void *)&sock, 0, NULL);
    while (1)
    {
        fgets(buf, BUFLen, stdin);
        buf[strlen(buf) - 1] = '\0'; // 把换行符替换为空字符
        int sendlen = send(sock, buf, strlen(buf) + 1, 0);
    }
}
```

```

        closesocket(sock); // 关闭监听套接字
        WSACleanup();      // 卸载 winsock library

        printf("按回车键继续...");
        getchar();

        return 0;
    }

unsigned __stdcall recvThread(void *p)
{
    char buf[BUFLEN + 1]; // 接收缓冲区

    SOCKET sock = *(SOCKET *)p;
    while (1)
    {
        printf(">>> "); // 提示符
        int recvlen = recv(sock, buf, BUFLen, 0);
        printf("\r"); // 回到行首, 覆盖掉之前显示的提示符
        if (recvlen == SOCKET_ERROR)
        {
            printf("[-] Error: %ld.\n", GetLastError());
            break;
        }
        else if (recvlen == 0)
        {
            printf("[-] Server closed!");
            break;
        }
        else if (recvlen > 0)
        {
            printf("[+] 收到消息: \n");
            buf[recvlen] = '\0';
            printf("%s\n", buf);
        }
        printf("-----\n");
    }

    return 0;
}

```

服务器端程序:

```

#include <stdio.h>
#include <stdlib.h>
#include <WinSock2.h>
#include <time.h>
#include <process.h>
#include <string>
#include "conio.h"
#pragma comment(lib, "ws2_32.lib") // 使用 winsock 2.2 library

const int MAXSOCKS = 5; // 最大并发连接数
const size_t BUFLEN = 2000; // 缓冲区大小

struct ThreadInfo {
    int thread_index; // 线程被分配至的数组下标
    HANDLE handle;    // 句柄, NULL 代表尚未被分配
    SOCKET ssock;     // 和客户端连接的从套接字

    ThreadInfo() : handle(NULL) {} // 构造函数
};

ThreadInfo threadinfo[MAXSOCKS];
unsigned __stdcall serveThread(void* p); // 服务一个客户端的线程
void sendToAll(const char* msg); // 向所有客户端发送字符串

int main()
{
    printf("[Group Chat Server]\n 正在启动群聊服务器...");
}

```

```

struct sockaddr_in sin; // an Internet endpoint address
struct sockaddr_in fsin; // the from address of a client
u_short port = 50500;

WSADATA wsadata;
WSAStartup(MAKEWORD(2, 0), &wsadata); // 加载winsock library
SOCKET msock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); // 监听套接字
memset(&sin, 0, sizeof(sin)); // 从&sin 开始的长度为sizeof(sin)的内存清0
sin.sin_family = AF_INET; // 因特网地址簇(INET-Internet)
sin.sin_addr.s_addr = INADDR_ANY; // 监听所有(接口的)IP 地址
sin.sin_port = htons(port); // 监听的端口号
bind(msock, (struct sockaddr *)&sin, sizeof(sin)); // 绑定监听的IP 地址和
端口号
listen(msock, 5); // 等待建立连接的队列长度为5

time_t now = time(NULL);
printf("成功.\n 启动时间: %s\n", ctime(&now));
printf("=====\\n");

while (!_kbhit())
{
    // 检测是否有按键
    int alen = sizeof(struct sockaddr); // 取到地址结构的长度
    SOCKET new_ssock = accept(msock, (struct sockaddr*)&fsin, &alen); //
如果有新的连接请求, 返回连接套接字, 否则, 被阻塞。fsin 包含客户端IP 地址和端口号

    int ssocks_full = 1; // 是否达到最大并发数的标志位
    for (int i = 0; i < MAXSOCKS; i++) {
        if (threadinfo[i].handle == NULL) { // 找到数组中的一个可用位置
            threadinfo[i].thread_index = i;
            threadinfo[i].handle = (HANDLE)_beginthreadex(NULL, 0,
&serveThread, (void*)&threadinfo[i], 0, NULL);
            threadinfo[i].ssock = new_ssock;
            ssocks_full = 0; // 未达到最大并发数, 清除标志位
            CloseHandle(threadinfo[i].handle); // 关闭线程句柄, 但并没有结
束线程

            break;
        }
    }
    if (ssocks_full) { // 已经达到最大并发数, 无法接纳更多连接
        const char* sorry_msg = "对不起, 群聊人数已满, 您暂时不能加入.";
        send(new_ssock, sorry_msg, strlen(sorry_msg), 0);
    }
}
closesocket(msock); // 关闭监听套接字
WSACleanup(); // 卸载winsock library

return 0;
}

unsigned __stdcall serveThread(void* p)
{
    ThreadInfo mythread = *(ThreadInfo*)p;
    char buf[BUFLen + 1]; // 线程缓冲区

    while (1) {
        int recvlen = recv(mythread.ssock, buf, BUFLen, 0); // 接收信息
        if (recvlen == SOCKET_ERROR)
        {
            printf("[ - ] Error: %ld.\\n", GetLastError());
            printf("-----\\n");
            break;
        }
        else if (recvlen == 0)
        {
            printf("[ - ] 客户端正常退出.\\n");
            printf("-----\\n");
            break;
        }
        else {
            buf[recvlen] = '\\0'; // 保证以空字符结尾
            printf("收到消息: %s\\n", buf);
        }
    }
}

```

```

        printf("-----\n");

        sendToAll(buf);
    }
}
//Call CloseHandle?
threadinfo[mythread.thread_index].handle = NULL; // 从数组中移除该线程
closesocket(mythread.ssock);
return 0;
}

void sendToAll(const char* msg)
{
    for (int i = 0; i < MAXSOCKS; i++) {
        if (threadinfo[i].handle != NULL) {
            send(threadinfo[i].ssock, msg, strlen(msg), 0);
        }
    }
}
}

```

运行截屏：

服务器：

```

[Group Chat Server]
正在启动群聊服务器...成功。
启动时间: Sun Mar 17 18:35:38 2019

=====
收到消息: Hello! I'm Client 1.
=====
收到消息: This is Client 2~~~
=====
收到消息: Hi guys, I am Client 3!
=====

```

客户端（3个）：

客户端1

客户端2

客户端3

(2) 服务器程序转发某个客户端发来的消息时都在消息前面加上该客户端的 IP 地址和端口号以及服务器的当前时间。要求服务器程序把转发的消息也显示出来。

我的思路:

客户端不需要改动, 只需要在服务器中添加一些代码。原来服务器只是将客户端发来的消息原封不动地转发给所有客户端, 现在要求在转发消息之前在消息中添加客户端的地址信息和当前时间信息等, 因此, 只需要在调用 `sendToAll` 函数之前处理一下字符串即可。

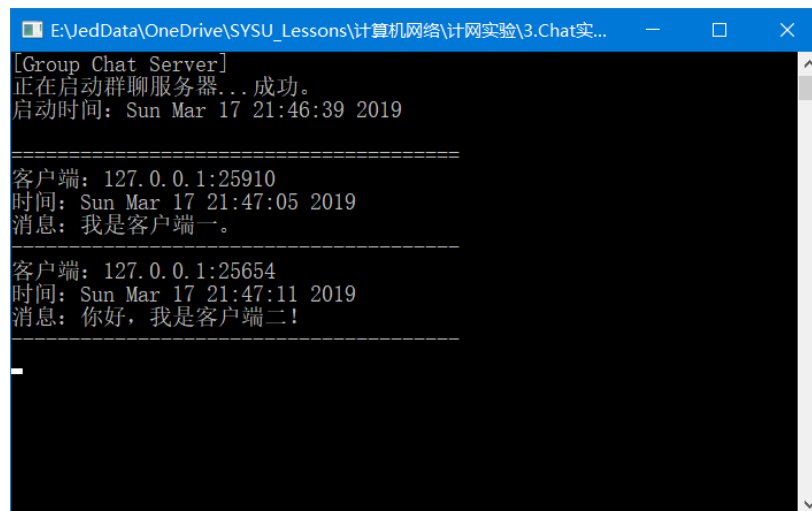
我新定义了一个函数 `makeReplyMsg` 来处理字符串。服务器程序中, 只需要在 `SendToAll` 之前先调用 `makeReplyMsg`。

服务器程序(修改部分):

```
void makeReplyMsg(char* msg, struct sockaddr_in fsin) {
    using namespace std;
    ostringstream os;
    os << "客户端: " << (int)fsin.sin_addr.S_un.S_un_b.s_b1 << '.' <<
(int)fsin.sin_addr.S_un.S_un_b.s_b2
    << '.' << (int)fsin.sin_addr.S_un.S_un_b.s_b3 << '.' <<
(int)fsin.sin_addr.S_un.S_un_b.s_b4
    << ':' << fsin.sin_port << endl;
    time_t now = time(NULL);
    os << "时间: " << ctime(&now) << flush;
    os << "消息: " << msg << endl;
    strncpy(msg, os.str().c_str(), BUFLen);
}
```

运行截屏:

服务器:



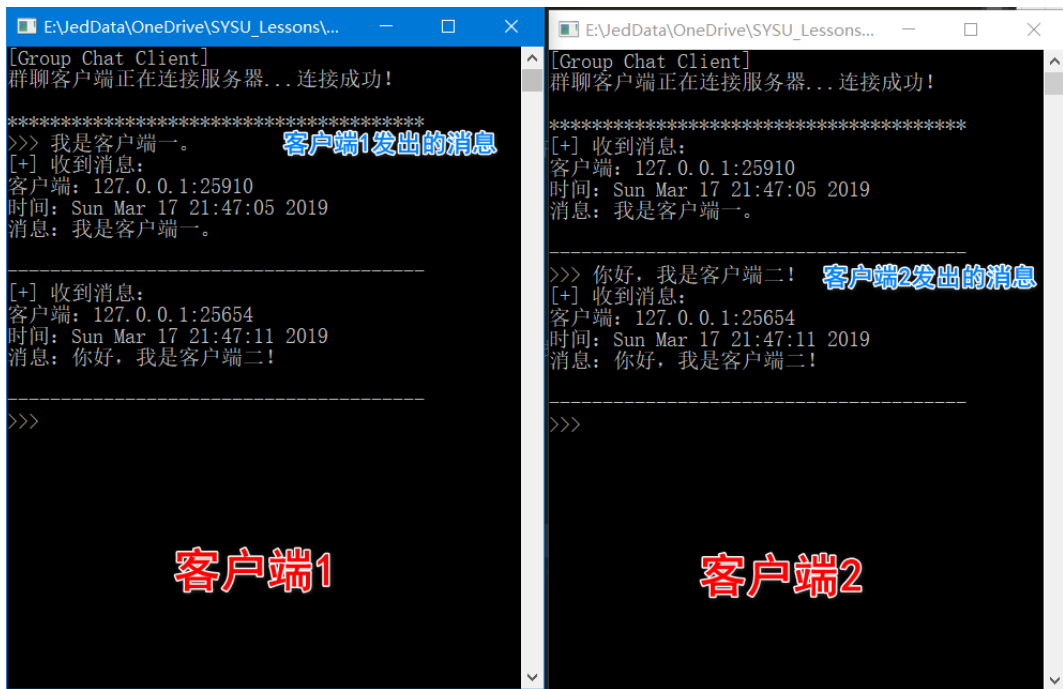
```
E:\JedData\OneDrive\SYSU_Lessons\计算机网络\计网实验\3.Chat实...
[Group Chat Server]
正在启动群聊服务器...成功。
启动时间: Sun Mar 17 21:46:39 2019

=====
客户端: 127.0.0.1:25910
时间: Sun Mar 17 21:47:05 2019
消息: 我是客户端一。

=====
客户端: 127.0.0.1:25654
时间: Sun Mar 17 21:47:11 2019
消息: 你好, 我是客户端二!

=====
```

客户端 (2 个):



(3) 新客户刚连接时服务器端把“enter”消息（包含客户端 IP 地址和端口号）发送给所有客户端。

服务器程序(修改部分):

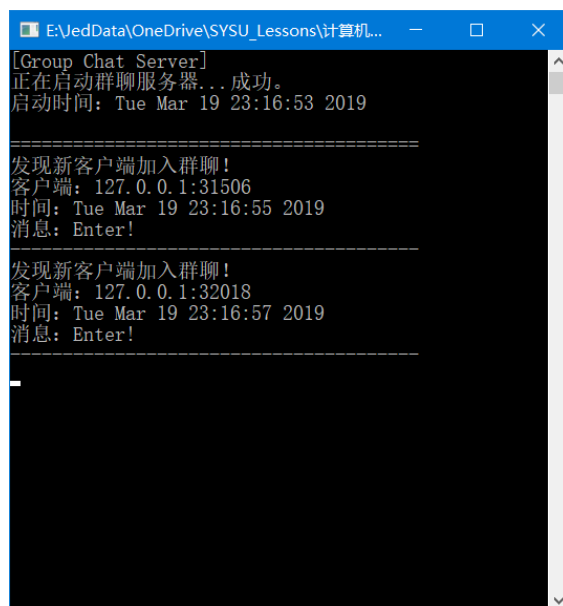
思路:

当有新客户端连接到服务器之后,服务器将新建线程,该新线程由 `serveThread` 函数管理。因此,要实现新客户连接时服务器显示并发送“Enter”消息,只需要在 `serveThread` 函数的开始处(`while` 循环之前)添加代码即可。注意,下面这段代码是在 `serveThread` 函数定义内的,而不是在 `main` 函数中。

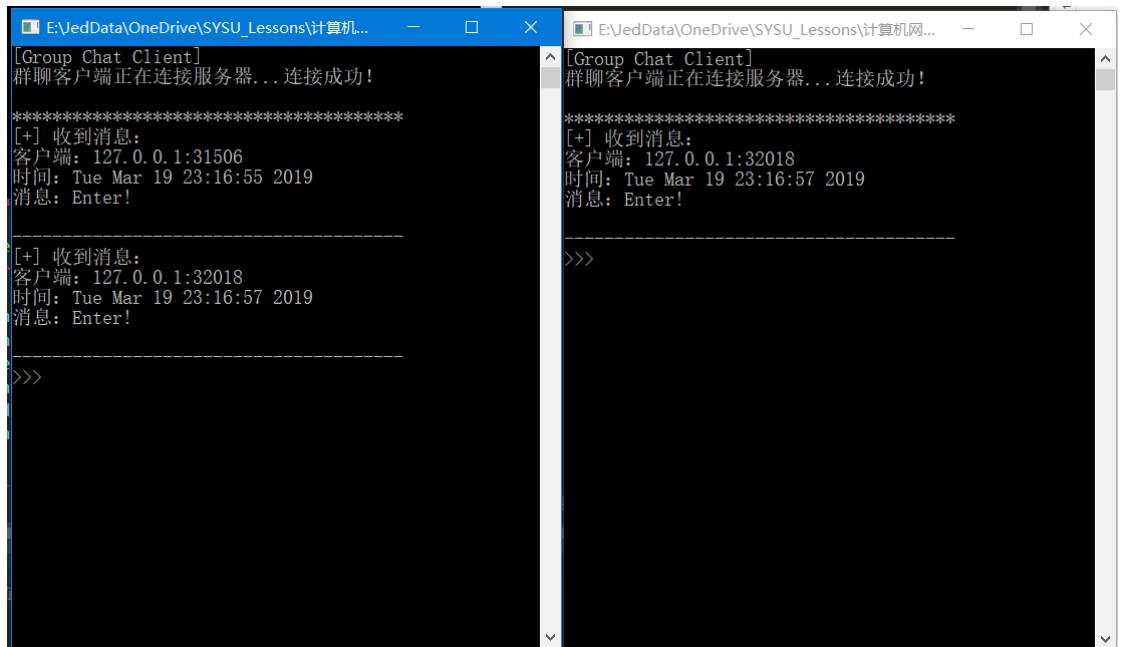
```
printf("发现新客户加入群聊! \n");
strncpy(buf, "Enter!", BUFLen);
makeReplyMsg(buf, mythread.fsin);
sendToAll(buf);
```

运行截屏:

服务器:



客户端 (2 个):



- (4) 客户端输入 exit 时退出客户端程序（正常退出），或者客户端直接关闭窗口退出（异常退出），服务器都会把该客户 leave 的消息广播给所有客户。

我的思路：

当服务器中的一个线程正在被 `recv` 函数阻塞时，客户端正常退出或异常退出，则 `recv` 将返回 `SOCKET_ERROR`，且错误码为 `10054`。因此增加如以下代码所示的判断条件即可。

服务器程序(修改部分)：

```
if (recvlen == SOCKET_ERROR)
{
    if (GetLastError() == 10054) {
        strncpy(buf, "Leave!", BUFLen);
        makeReplyMsg(buf, mythread.fsin);
        printf("%s", buf);
        sendToAll(buf);
    }
    else{
        printf("[-] Error: %ld.\n", GetLastError());
    }
    printf("-----\n");
    break;
}
```

运行截屏：

服务器：


```
E:\JedData\OneDrive\SYSU_Lessons\计算机...
[Group Chat Server]
正在启动群聊服务器...成功。
启动时间: Tue Mar 19 23:16:53 2019

=====
发现新客户端加入群聊!
客户端: 127.0.0.1:31506
时间: Tue Mar 19 23:16:55 2019
消息: Enter!

-----
发现新客户端加入群聊!
客户端: 127.0.0.1:32018
时间: Tue Mar 19 23:16:57 2019
消息: Enter!

-----
客户端: 127.0.0.1:31506
时间: Tue Mar 19 23:18:18 2019
消息: Leave!

-----
-

```

客户端（本来打开了两个，关掉了一个，还剩一个）：

```
E:\JedData\OneDrive\SYSU_Lessons\计算机网...
[Group Chat Client]
群聊客户端正在连接服务器...连接成功!

*****
[+] 收到消息:
客户端: 127.0.0.1:32018
时间: Tue Mar 19 23:16:57 2019
消息: Enter!

-----
[+] 收到消息:
客户端: 127.0.0.1:31506
时间: Tue Mar 19 23:18:18 2019
消息: Leave!

-----
>>>

```

(5) 运行客户端程序测试与老师的服务器程序的连接（172.18.187.9:50500）。

说明：

如下图所示，我的客户端可以和老师的服务器正常连接，我的校园网 IP 地址是 172.18.147.199，每次发送消息都是通过本地 4124 端口发送的（仅限当次测试）。显然，老师的服务器在客户端发过去的消息后面都加上了“>>”提示符，而由于我的客户端本身就会显示分割线和“>>>”提示符，因此看起来像是重复了一样，实际上没有错误，一切都在意料之中。运行截屏（客户端）：

```
E:\JedData\OneDrive\SYSU_Lessons\计算机网络\计网实验\3.Chat实...
[Group Chat Client]
群聊客户端正在连接服务器... 连接成功!

*****
[+] 收到消息:
ip: 172.18.147.199 port: 4124
time: Sun Mar 17 18:45:17 2019
message: Enter!

>>

>>> Hi, this is Zhang Yixin (17341203)!
[+] 收到消息:
ip: 172.18.147.199 port: 4124
time: Sun Mar 17 18:45:36 2019
message: Hi, this is Zhang Yixin (17341203)!

>>

>>> This is a test message!
[+] 收到消息:
ip: 172.18.147.199 port: 4124
time: Sun Mar 17 18:46:01 2019
message: This is a test message!

>>

>>> 测试一下中文。
[+] 收到消息:
ip: 172.18.147.199 port: 4124
time: Sun Mar 17 18:46:17 2019
message: 测试一下中文。

>>

>>>
```

(6) 与同学的程序进行相互测试，一个人可以与多人测试，截屏选择其中一个。

测试①:

作为服务器运行截屏:

```
E:\JedData\OneDrive\SYSU_Lessons\计算机网络\计网实验\3.Chat实验\GroupChat_Server\Debug\...
[Group Chat Server]
正在启动群聊服务器... 成功。
启动时间: Sun Mar 17 22:01:57 2019

=====
发现新客户端加入群聊!
客户端: 172.26.8.73:14689
时间: Sun Mar 17 22:02:18 2019
消息: ccc

=====
```

作为客户端运行截屏:

```
E:\JedData\OneDrive\YSU_Lessons\计算机网络\计网实验\3.Chat实验\GroupChat_Client\Debug\GroupChat_Client.exe
[Group Chat Client]
群聊客户端正在连接服务器... 连接成功!

*****
[+] 收到消息:
客户端地址: 172.18.147.199客户端端口号: 2615 enter

-----
>>> 张怡昕, 17341203
[+] 收到消息:
张怡昕, 17341203时间: Sun Mar 17 22:01:14 2019
客户端地址: 172.18.147.199客户端端口号: 2615
-----
>>> -
```

测试②:

作为服务器运行截屏:

```
E:\JedData\OneDrive\YSU_Lessons\计算机网络\计网实验\3.Chat实验\GroupChat_Server\Debug\GroupCh...
[Group Chat Server]
正在启动群聊服务器... 成功。
启动时间: Fri Mar 22 20:55:48 2019

=====
发现新客户端加入群聊!
客户端: 172.26.8.153:43762
时间: Fri Mar 22 20:55:50 2019
消息: Enter!

-----
客户端: 172.26.8.153:43762
时间: Fri Mar 22 20:55:55 2019
消息: Zhuhaonan

-----
发现新客户端加入群聊!
客户端: 172.18.147.199:55567
时间: Fri Mar 22 20:55:56 2019
消息: Enter!

-----
发现新客户端加入群聊!
客户端: 172.26.8.153:44530
时间: Fri Mar 22 20:56:00 2019
消息: Enter!

-----
客户端: 172.26.8.153:44530
时间: Fri Mar 22 20:56:02 2019
消息: 17341221

-----
```

作为客户端运行截屏:

```
E:\JedData\OneDrive\SYSU_Lessons\计算机网络\计网实验\3.Chat实验\GroupChat_Client\Debug\...
[Group Chat Client]
群聊客户端正在连接服务器...连接成功!

*****
[+] 收到消息:
Fri Mar 22 21:14:31 2019
IP: 172.18.147.199
Port: 45841
Enter!

-----

>>> Jed here! 17341203
[+] 收到消息:
Fri Mar 22 21:14:39 2019
IP: 172.18.147.199
Port: 45841

-----

[+] 收到消息:
Fri Mar 22 21:14:50 2019
IP: 127.0.0.1
Port: 29427

-----

>>>
```

测试③（3个人一起互测）:

作为客户端运行截屏:

```
E:\JedData\OneDrive\SYSU_Lessons\计算机网络\计网实验\3.Chat实验\Group...
[+] 收到消息:
<IP: 192.168.43.90 PORT:39473>
TIME: Fri Mar 22 22:03:05 2019
Message: xuzhicneg
>>

-----

[+] 收到消息:
<IP: 192.168.43.90 PORT:39473>
TIME: Fri Mar 22 22:03:14 2019
Message: zcheng
>>

-----

[+] 收到消息:
<IP: 192.168.43.90 PORT:39473>
TIME: Fri Mar 22 22:03:18 2019
Message: xuzhichen4g
>>

-----

[+] 收到消息:
<IP: 192.168.43.90 PORT:39473>
TIME: Fri Mar 22 22:03:25 2019
Message: xuzhicheng
>>

>>> _
```

作为服务器运行截屏:

```
E:\JedData\OneDrive\SYSU_Lessons\计算机网络\计网实...
客户端: 192.168.43.100:6179
时间: Fri Mar 22 22:26:15 2019
消息: Enter!

-----
发现新客户端加入群聊!
客户端: 192.168.43.90:22303
时间: Fri Mar 22 22:26:52 2019
消息: Enter!

-----
客户端: 192.168.43.178:35589
时间: Fri Mar 22 22:27:01 2019
消息: okokok

-----
客户端: 192.168.43.100:6179
时间: Fri Mar 22 22:27:05 2019
消息: hhello

-----
客户端: 192.168.43.100:6179
时间: Fri Mar 22 22:27:07 2019
消息: nb

-----
客户端: 192.168.43.90:22303
时间: Fri Mar 22 22:27:12 2019
消息: kkkk

-----
客户端: 192.168.43.90:22303
时间: Fri Mar 22 22:27:15 2019
消息: mmmm

-----
```

【完成情况】

是否完成了这些步骤? (√完成 ×未做或未完成)

(1) [√] (2) [√] (3) [√] (4) [√] (5) [√] (6) [√]

【实验体会】

1. **线程、并行与参数传递。**本次实验中，关于 SOCKET 编程的部分与上一个实验（实验 2 - Echo 实验）几乎相同，没有任何新的函数出现。本次实验新增的内容是多线程编程，主要涉及到的函数是 `_beginthreadex`、`CloseHandle` 等，重点是要理解进程、主线程、子线程的概念及其之间的关系。在编程操作上，难点是掌握 `_beginthreadex` 函数的参数传递，对程序员来说，该函数最重要的两个参数分别是第 3 个——函数名参数，以及第 4 个——传递给新线程的参数列表。需仔细阅读文档，并使用简单的程序进行测试尝试，方能掌握函数的用法。

2. **值传递与引用传递。**编程中我遇到了一个问题，现象如下：启动服务器后，启动 1 号客户端，经测试可以正常与服务器之间通信，再启动 2 号客户端，它也可以正常和服务器之间通信，然而此时再次尝试使用 1 号客户端发送消息时，发现 1 号客户端只能发送消息，而服务器无法收到，也无法转发消息给所有客户端。

经过简单的调试便发现了问题：`_beginthreadex` 传参时，传递的是指针，当 2 号客户端连接服务器后，相应的 socket 变为 2 号客户端的，于是 1 号客户端的线程中服务的 socket 的值也变成了 2 号——相当于 1 号客户端的 socket 丢失了，原先的线程无法再服务 1 号客户端。经过我的修改，在 `serveThread` 函数中先将参数拷贝一份，然后用被拷贝的参数的 socket 与客户端通信，这样修改后，每个客户端都可以与服务器正常通信了。

3. **CloseHandle 放在何处？**把整个程序的框架基本完成后，我发现似乎没有合适的地方调用 `CloseHandle` 这一函数。之前，我错误地认为这个函数的作用是关闭线程，即对某一线程句柄调用 `CloseHandle` 后，该线程将立即被强制结束。受这一错误理解的影响，我在编程中遇到了一个矛盾之处，就是——客户端输入 `exit` 正常退出或点击关闭按钮异常退出后，服务器必须断开连接并结束这个线程，在 `threadinfo` 数组中腾出位置来供新客户端连接用，然而由于主线程不负责与客户端通信，因此结束线程的任务必须放在线程函数（即 `serveThread`）中，这样就造成了“在线程内部结束自己”这种荒谬的逻辑，导致结束线程后的关闭 socket、卸载 Winsock Library 流程不能完成。

在思考、请教、查资料后，我知道了 `CloseHandle` 函数的作用并不是结束线程。当线程被创建时，认为它的内核状态为 2，对该线程的句柄调用 `CloseHandle` 之后，它的状态减为 1，但此时

县城仍然可以继续执行，被关闭的仅仅是句柄——也就是说不能再用该句柄来对线程进行操作了。线程真正被关闭是在线程函数返回后，此时县城内核状态减为 0，操作系统回收资源，删除线程。在我的程序中，子线程被创建后便不再需要线程句柄，所以考虑创建线程后立即关闭线程句柄。经测试，我的程序可以正常地运行。

4. **多线程导致的显示问题。**在客户端程序中，主线程负责接受用户输入并向服务器发送消息，子线程负责接收服务器发来的消息并打印在屏幕上。这样就会导致一个问题，当客户端主线程正在等待用户输入的时候，突然来了一条消息，那么出现的消息将会立即出现在光标后面，之后用户在输入时就没有提示符了。我的解决方案是，使用 `\r` 转义字符，在显示消息前将光标定位到行首，这样，服务器发来的消息就可以正常地从行首开始显示了。
5. **未解决的显示问题。**如果用户在客户端中输入了一些字符，但是尚未按下回车发送，这时服务器发来消息后，客户端中用户输入的字符（在缓冲区中）将被部分覆盖，不过这些字符并未从缓冲区中删除——用户按下回车后，之前输入的字符仍然会被发送。我有一个解决的思路，不过受限于某些原因未能完成：当客户端收到服务器发来的消息后，先把键盘缓冲区的内容缓存起来，然后将光标定位到行首并显示服务器发来的消息，最后再将缓存的内容还原到键盘缓冲区去。