**海南大学计算机与网络空间安全学院**

**实 验 报 告**

|  |  |
| --- | --- |
| **评定成绩** | **指导教师** |
|  |  |

**实验课程：** 网络协议与分析

**实验名称：**基于TCP的客户端与服务器编程-网络聊天

1. 实验目的
   1. 掌握网络应用程序的开发方法；
   2. 掌握Client/ Server结构软件的设计与开发方法
   3. 掌握Socket机制的工作原理
2. 实验任务

使用Win32 Socket 函数实现多人在线聊天程序：能相互发送与接受文本消息。

1. 实验环境

Win10+visual studio2019

1. 实验内容

服务器端代码

#include <winsock2.h>

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#define MAX\_CLNT 256

#define BUF\_SIZE 100

#pragma comment(lib,"WS2\_32.lib")

void error\_handling(const char\* msg); /\*错误处理函数\*/

DWORD WINAPI ThreadProc(LPVOID lpParam); /\*线程执行函数\*/

void send\_msg(char\* msg, int len); /\*消息发送函数\*/

HANDLE g\_hEvent; /\*事件内核对象\*/

int clnt\_cnt = 0; //统计套接字

int clnt\_socks[MAX\_CLNT]; //管理套接字

HANDLE hThread[MAX\_CLNT]; //管理线程

int main()

{

// 初始化WS2\_32.dll

WSADATA wsaData;

WORD sockVersion = MAKEWORD(2, 2);

WSAStartup(sockVersion, &wsaData); //请求了一个2.2版本的socket

// 创建一个自动重置的（auto-reset events），受信的（signaled）事件内核对象

g\_hEvent = CreateEvent(NULL, FALSE, TRUE, NULL);

// 创建套节字

SOCKET serv\_sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (serv\_sock == INVALID\_SOCKET)

error\_handling("Failed socket()");

// 填充sockaddr\_in结构

sockaddr\_in sin;

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(8888); //8888端口

sin.sin\_addr.S\_un.S\_addr = INADDR\_ANY; //本地地址

// 绑定这个套节字到一个本地地址

if (bind(serv\_sock, (LPSOCKADDR)&sin, sizeof(sin)) == SOCKET\_ERROR)

error\_handling("Failed bind()");

// 进入监听模式

if (listen(serv\_sock, MAX\_CLNT) == SOCKET\_ERROR)

error\_handling("Failed listen()");

printf("Start listen:\n");

// 循环接受客户的连接请求

sockaddr\_in remoteAddr;

int nAddrLen = sizeof(remoteAddr);

DWORD dwThreadId; /\*线程ID\*/

SOCKET clnt\_sock;

while (TRUE)

{

printf("等待新连接\n");

// 接受一个新连接

clnt\_sock = accept(serv\_sock, (SOCKADDR\*)&remoteAddr, &nAddrLen);

if (clnt\_sock == INVALID\_SOCKET)

{

printf("Failed accept()");

continue;

}

/\*等待内核事件对象状态受信\*/

WaitForSingleObject(g\_hEvent, INFINITE);

hThread[clnt\_cnt] = CreateThread(

NULL, // 默认安全属性

NULL, // 默认堆栈大小

ThreadProc, // 线程入口地址（执行线程的函数）

(void\*)&clnt\_sock, // 传给函数的参数

0, // 指定线程立即运行

&dwThreadId); // 返回线程的ID号

clnt\_socks[clnt\_cnt++] = clnt\_sock;

SetEvent(g\_hEvent); /\*设置受信\*/

printf(" 接受到一个连接：%s 执行线程ID：%d\r\n", inet\_ntoa(remoteAddr.sin\_addr), dwThreadId);

}

WaitForMultipleObjects(clnt\_cnt, hThread, true, INFINITE);

for (int i = 0; i < clnt\_cnt; i++)

{

CloseHandle(hThread[i]);

}

// 关闭监听套节字

closesocket(serv\_sock);

// 释放WS2\_32库

WSACleanup();

return 0;

}

void error\_handling(const char\* msg)

{

printf("%s\n", msg);

WSACleanup();

exit(1);

}

DWORD WINAPI ThreadProc(LPVOID lpParam)

{

int clnt\_sock = \*((int\*)lpParam);

int str\_len = 0, i;

char msg[BUF\_SIZE];

while ((str\_len = recv(clnt\_sock, msg, sizeof(msg), 0)) != -1)

{

send\_msg(msg, str\_len);

printf("群发送成功\n");

}

printf("客户端退出:%d\n", GetCurrentThreadId());

/\*等待内核事件对象状态受信\*/

WaitForSingleObject(g\_hEvent, INFINITE);

for (i = 0; i < clnt\_cnt; i++)

{

if (clnt\_sock == clnt\_socks[i])

{

while (i++ < clnt\_cnt - 1)

clnt\_socks[i] = clnt\_socks[i + 1];

break;

}

}

clnt\_cnt--;

SetEvent(g\_hEvent); /\*设置受信\*/

// 关闭同客户端的连接

closesocket(clnt\_sock);

return NULL;

}

void send\_msg(char\* msg, int len)

{

int i;

/\*等待内核事件对象状态受信\*/

WaitForSingleObject(g\_hEvent, INFINITE);

for (i = 0; i < clnt\_cnt; i++)

send(clnt\_socks[i], msg, len, 0);

SetEvent(g\_hEvent); /\*设置受信\*/

}

客户端代码

#include <winsock2.h>

#include <stdio.h>

#include <windows.h>

#pragma comment(lib,"WS2\_32.lib")

#define BUF\_SIZE 256

#define NAME\_SIZE 30

DWORD WINAPI send\_msg(LPVOID lpParam);

DWORD WINAPI recv\_msg(LPVOID lpParam);

void error\_handling(const char\* msg);

char name[NAME\_SIZE] = "[DEFAULT]";

char msg[BUF\_SIZE];

int main()

{

HANDLE hThread[2];

DWORD dwThreadId;

// 初始化WS2\_32.dll

WSADATA wsaData;

WORD sockVersion = MAKEWORD(2, 2);

WSAStartup(sockVersion, &wsaData);

/\*设置登录用户名\*/

printf("Input your Chat Name:");

scanf("%s", name);

getchar();

// 创建套节字

SOCKET sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (sock == INVALID\_SOCKET)

error\_handling("Failed socket()");

// 填写远程地址信息

sockaddr\_in servAddr;

servAddr.sin\_family = AF\_INET;

servAddr.sin\_port = htons(8888);

// 如果你的计算机没有联网，直接使用本地地址127.0.0.1

servAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

if (connect(sock, (sockaddr\*)&servAddr, sizeof(servAddr)) == -1)

error\_handling("Failed connect()");

printf("connect success\n");

hThread[0] = CreateThread(

NULL, // 默认安全属性

NULL, // 默认堆栈大小

send\_msg, // 线程入口地址（执行线程的函数）

&sock, // 传给函数的参数

0, // 指定线程立即运行

&dwThreadId); // 返回线程的ID号

hThread[1] = CreateThread(

NULL, // 默认安全属性

NULL, // 默认堆栈大小

recv\_msg, // 线程入口地址（执行线程的函数）

&sock, // 传给函数的参数

0, // 指定线程立即运行

&dwThreadId); // 返回线程的ID号

// 等待线程运行结束

WaitForMultipleObjects(2, hThread, true, INFINITE);

CloseHandle(hThread[0]);

CloseHandle(hThread[1]);

printf(" Thread Over,Enter anything to over.\n");

getchar();

// 关闭套节字

closesocket(sock);

// 释放WS2\_32库

WSACleanup();

return 0;

}

DWORD WINAPI send\_msg(LPVOID lpParam)

{

int sock = \*((int\*)lpParam);

char name\_msg[NAME\_SIZE + BUF\_SIZE];

while (1)

{

fgets(msg, BUF\_SIZE, stdin);

if (!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))

{

closesocket(sock);

exit(0);

}

sprintf(name\_msg, "[%s]: %s", name, msg);

int nRecv = send(sock, name\_msg, strlen(name\_msg), 0);

}

return NULL;

}

DWORD WINAPI recv\_msg(LPVOID lpParam)

{

int sock = \*((int\*)lpParam);

char name\_msg[NAME\_SIZE + BUF\_SIZE];

int str\_len;

while (1)

{

str\_len = recv(sock, name\_msg, NAME\_SIZE + BUF\_SIZE - 1, 0);

if (str\_len == -1)

return -1;

name\_msg[str\_len] = 0;

fputs(name\_msg, stdout);

}

return NULL;

}

void error\_handling(const char\* msg)

{

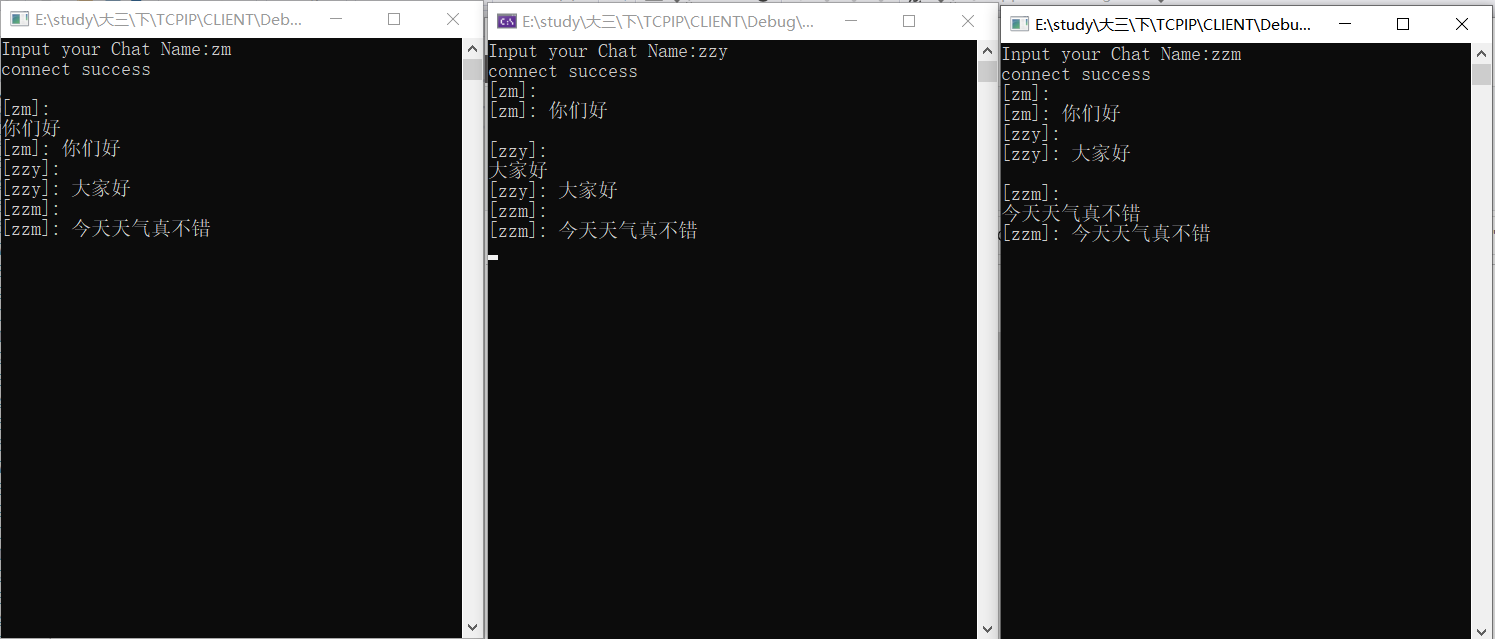
printf("%s\n", msg);

WSACleanup();

exit(1);

}

运行结果



1. 实验总结

通过本次实验，我掌握了基于TCP的socket网络编程以及多线程编程和线程的同步。Winsock的通信步骤为WSAStartup 进行初始化、socket 创建套接字、bind 绑定、 listen 监听、 connect 连接、accept 接收请求、 send/recv 发送或接收数据、closesocket 关闭 socket、WSACleanup 最终关闭。多线程是为了提高系统的运行效率。Win32 API下的多线程编程也就是两个函数的应用CreateThread以及WaitForSingleObject。使用多线程的过程中需要注意的问题是如何防止两个或两个以上的线程同时访问同一个数据，以免破坏数据的完整性。本实验是基于事件内核对象实现的线程同步，事件内核对象是一种抽象的对象，有受信和未授信两种状态，通过等待WaitForSingleObject实现线程同步。