数据通信作业

姓名：刘浩文　　学号：517021911065　　日期：2020/5/11

一、 实验名称及内容

名称：利用 *Winsock* 完成基于 **UDP** 协议的 **P2P** 聊天程序

内容：利用 *Winsock* 完成基于 **UDP** 协议的 **P2P** 聊天程序，用户可以通过该聊天程序给其他用户发信息，通过 *IP* 地址和端口号识别用户。

二、实验过程和结果

环境

物理主机系统：*macOS Catalina 10.15.4*

虚拟机系统：*Windows 10 专业版 x64*

计算机名：*691B*

虚拟机软件：*Parallels Desktop 15 for Mac Pro Edition, version 15.1.4 (47270)*

编程环境(IDE)：*Visual Studio 2019*

程序设计

1. 调用 *WSAStartup* 函数，初始化 *winsock*
2. 调用 *socket()* 函数创建一个 **Socket**（UDP，DGRAM）
3. 调用 *bind()* 函数和 **socket** 绑定
4. 调用 *_beginthread()* 函数创建一个消息发送线程
    1. 调用 *sendto()* 函数发送消息

    2. 如果发送 “**quit**” 字符串则关闭本线程 *_endthread()*
5. 调用 *_beginthread()* 函数创建一个消息接收线程
    1. 调用 *recvfrom()* 函数接收消息
    2. 如果收到 “**quit**” 字符串则关闭本线程 *_endthread()*
6. 如果两个线程均已终止，则调用 *closesocket()* 和 *WSACleanup()* 函数关闭 **socket** ，结束程序

程序主体说明

数据结构

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include<iostream>
#include<sstream>
#include<string>
#include <process.h>
using namespace std;

#pragma comment(lib, "Ws2_32.lib")

struct clientinfo          // 用来之后向线程中传递参数
{
  SOCKADDR_IN addrClt;
  SOCKET sockServer;
};

int      Count;            // 计数线程数，为0时程序结束
```

函数

```
u_short ss2n(string s)     // 将字符串转为数字，用来将argv指向的字符串类型的端口号转为短整型
{
  stringstream ss;
  u_short u;
  ss << s;
  ss >> u;
  return u;
}
```

```
void sendmsg(void* client)  // 发送消息线程函数
{
  char sendBuf[2048];
  int len = sizeof(SOCKADDR);
  SOCKADDR_IN addrClt;
  SOCKET sockServer;
```

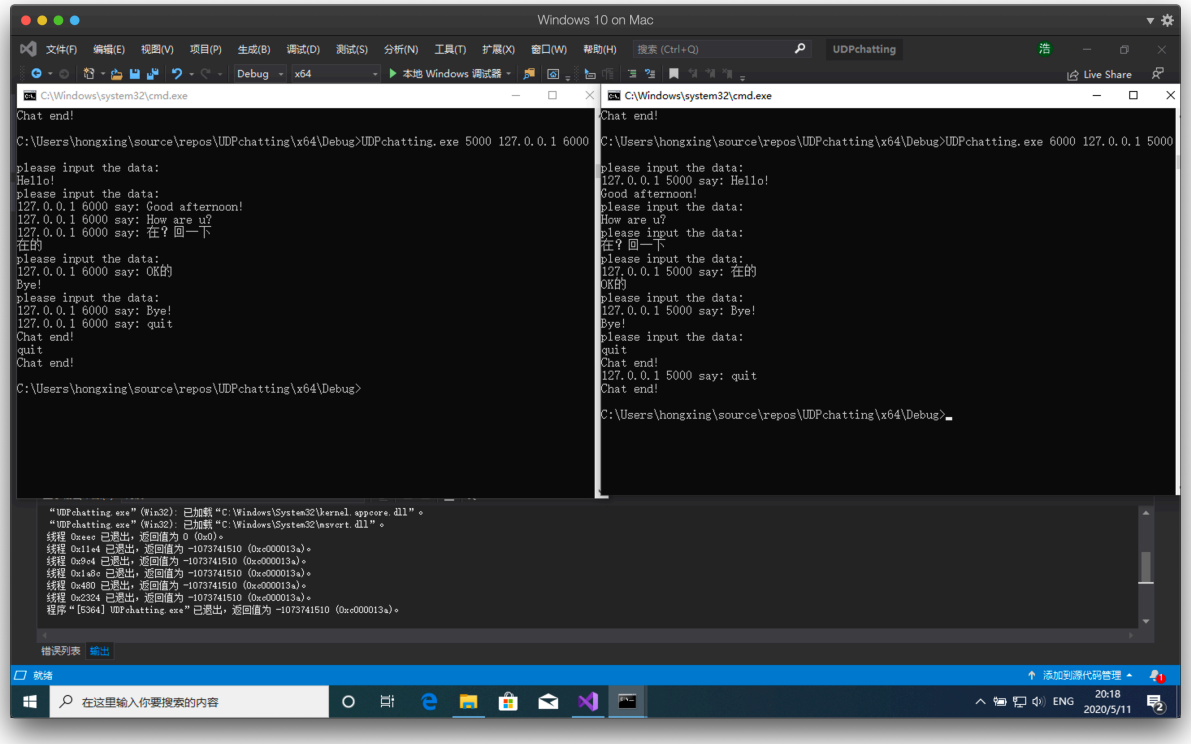
```
  addrClt = ((clientinfo*)client)->addrClt;       // 接受参数传递的通信对方信息
  sockServer = ((clientinfo*)client)->sockServer; // 接受参数传递的socket
  while (1)
  {
    cout << "please input the data: " << endl;
    gets_s(sendBuf);
    // 发送消息
    sendto(sockServer, sendBuf, strlen(sendBuf) + 1, 0, (SOCKADDR*)&addrClt, len);
    if (strcmp(sendBuf, "quit") == 0) {   // 检测是否要结束发送线程
      cout << "Chat end!" << endl;
      break;
    }
  }
  Count--;      // 线程结束，线程数减一
  _endthread();
}

void receivemsg(void* client) // 接收消息线程函数
{
  char recvBuf[2048];
  int len = sizeof(SOCKADDR);
  SOCKADDR_IN addrClt;
  SOCKET sockServer;
  addrClt = ((clientinfo*)client)->addrClt;       // 接受参数传递的通信对方信息
  sockServer = ((clientinfo*)client)->sockServer; // 接受参数传递的socket

  while (1)
  {
    recvfrom(sockServer, recvBuf, strlen(recvBuf) + 1, 0, (SOCKADDR*)&addrClt, &len);
    char str[INET_ADDRSTRLEN];
    char* ptr = (char*)inet_ntop(AF_INET, &addrClt.sin_addr, str, sizeof(str));
    // 接收消息
    cout << ptr << " " << addrClt.sin_port << " say: " << recvBuf << endl;
    if (strcmp(recvBuf, "quit") == 0) {   // 检测是否要结束接收线程
      cout << "Chat end!" << endl;
      break;
    }
  }
  Count--;      // 线程结束，线程数减一
  _endthread();
}
```

```
void main(int argc, char** argv)
{
```

```
  WORD wVersionRequested;
  WSADATA wsaData;
  int err;
  wVersionRequested = MAKEWORD(2, 2);

  err = WSAStartup(wVersionRequested, &wsaData);  // 初始化winsock
  if (err != 0) {
    return;
  }
  if (LOBYTE(wsaData.wVersion) != 2 ||
    HIBYTE(wsaData.wVersion) != 2) {
    WSACleanup();
    return;
  }

  if (argv[1] == NULL)  // 解决编译报错的问题：由于编译时没有传递指针数组argv导致报错内存冲突
  {
    argv[1] = (char*)"10000";
    argv[2] = (char*)"127.0.0.1";
    argv[3] = (char*)"10001";
  }

  // 创建 UDP socket
  SOCKET sockServer = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

  SOCKADDR_IN addrClt;    // 通信对方地址信息
  inet_pton(AF_INET, argv[2], &addrClt.sin_addr);
  addrClt.sin_family = AF_INET;
  addrClt.sin_port = ss2n(argv[3]);

  SOCKADDR_IN addrSrv;    // 自己的地址信息
  addrSrv.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
  addrSrv.sin_family = AF_INET;
  addrSrv.sin_port = ss2n(argv[1]);

  bind(sockServer, (SOCKADDR*)&addrSrv, sizeof(SOCKADDR));  // 绑定socket

  clientinfo clienti;   // 线程传参预备
  clienti.addrClt = addrClt;
  clienti.sockServer = sockServer;

  int len = sizeof(SOCKADDR);
  Count = 2;
  if (_beginthread(sendmsg, 8192, (void*)&clienti) < 0)   // 创建消息发送线程
  {
    printf("ERROR - Unable to create thread \n");
    exit(1);
  }
```

```
51
52    if (_beginthread(receivemsg, 8192, (void*)&clienti) < 0)     // 创建消息接收线程
53    {
54      printf("ERROR - Unable to create thread \n");
55      exit(1);
56    }
57
58    while (Count);
59
60    closesocket(sockServer);     // 关闭socket，关闭程序
61    WSACleanup();
62  }
```

实验结果



如图，左边的用户 **IP** 为 127.0.0.1，端口为 5000，它要和 **IP** 为 127.0.0.1，端口为 6000 的用户通信；右边的用户 **IP** 为 127.0.0.1，端口为 6000，它要和 **IP** 为 127.0.0.1，端口为 5000 的用户通信。双方任意一方先发起通信都可以。接下来就可以进行通信，双方都可收发消息。由于发送消息与接收消息是两个线程，所以**发送与接收消息互不干扰**。发送者发出消息后，另一方能立即收到并打印出消息发送者标示与接收到的消息。输入 “**quit**” 并发送后，发送者的发送线程即关闭，但仍能接收对方发的消息；对方的接收线程也关闭，但仍能向发起 **quit** 者发送消息。直到对方也发送了 “**quit**”，原发送方的接收线程与原对方的发送线程才会关闭，通信正式结束，双方打出 “**Chat end!**“。

三、问题与思考

多线程

由于这是一个 **P2P** 的聊天程序，接收消息与发送消息必须要用两个不同的线程，否则会相互干扰，比如发消息时不能收消息，等待接收消息时不能发消息。由于用惯了 *Python* 简单明了的线程函数，不熟悉 *C++* 的线程编程，因此搞清楚参数传递关系，参数类型就花了很多时间，最后用一个结构体实现传递多个参数，使用多个强制类型转换来配合传递数据类型。