

## 【实验题目】Echo 实验

【实验目的】掌握套节字的基本使用方法。

### 【实验说明】

- ♦ 把源程序和可执行文件放在相应的**上交源码**目录中。
- ♦ **截屏**用按键(Ctrl+Alt+PrintScreen)截取当前窗口

### 【参考资料】

- ♦ <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）

### 【实验环境】

- ♦ Windows + VS 2012
- ♦ 对于 VS2015 和 VS2017 默认使用**安全周期检查**，如果不关闭 VS 的安全周期检查，很多字符串函数都不能用。
- ♦ Linux + gcc
- ♦ Windows + gcc

### 【实验内容】

先尝试运行文件夹“TCP”中的程序：先运行 Server 程序(打开 TCPServer.sln，然后执行)再运行 Client 程序(打开 TCPClient.sln，然后执行)。这两个程序的功能是客户端从服务器获取当前时间。

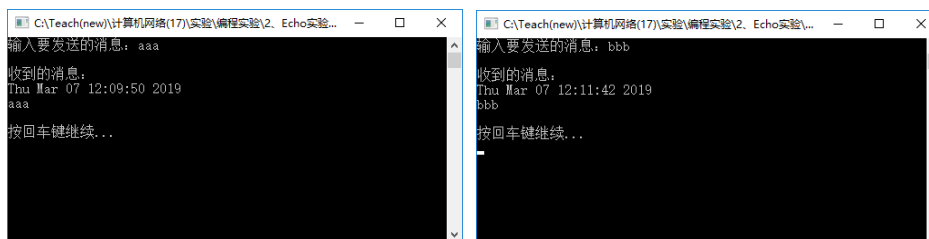
## (1)编写 TCP Echo 程序

### ▪ 实验要求：

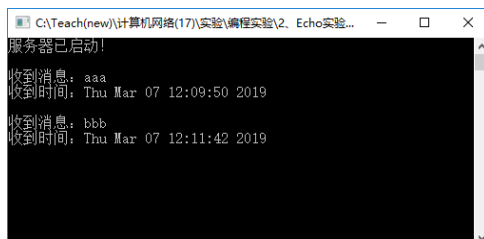
**服务器**把客户端发送来的任何消息都返回给客户端，返回的消息前面要加上服务器的当前时间。  
**客户端**把返回的消息显示出来。客户端每输入一条消息就建立 TCP 连接，并把消息发送给服务器，在收到服务器回应后关闭连接。

### ▪ 参考运行截屏：

客户端（两次运行）



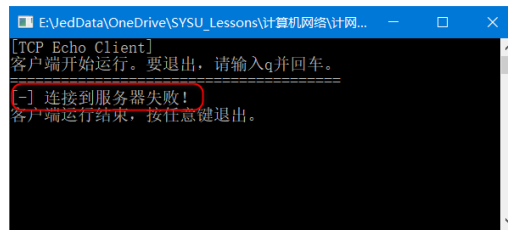
服务器：



### ▪ 只运行客户端程序而不运行服务器程序会出现什么错误，截屏并说明原因。

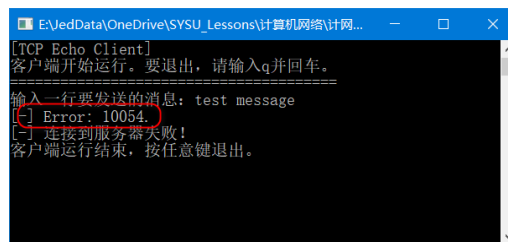
我的程序充分考虑了这种情况，并有相应的处理措施。这里分为两种情况分别说明：

- 情况一：在服务器没有运行的时候启动客户端



如上图所示，客户端提示“连接到服务器失败”，接着客户端退出。这是因为客户端在执行 `connect` 函数时就无法连接到服务器，连接失败时 `connect` 函数返回-1。

- 情况二：客户端启动时服务器是运行的，之后才关闭。即服务器在客户端发送数据时处于关闭状态



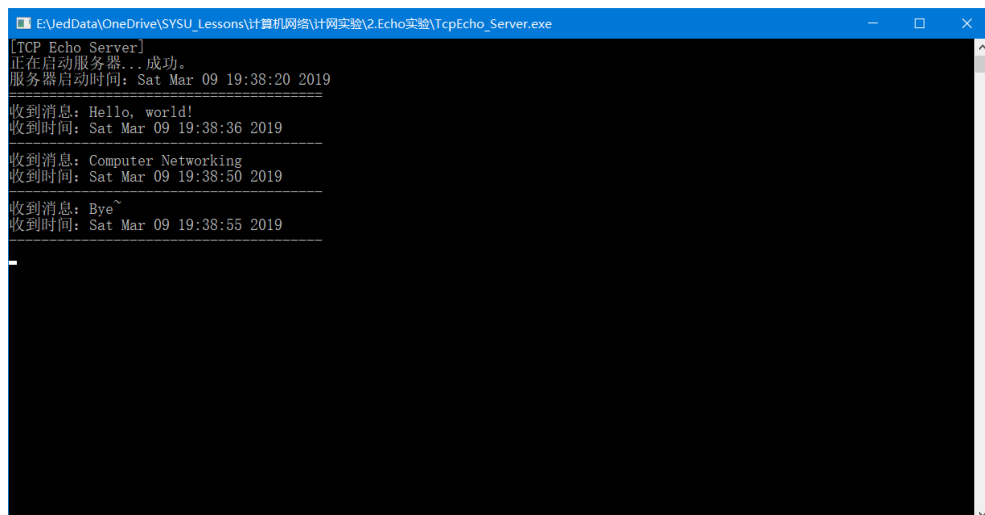
如上图所示，用户输入并回车要发送的消息后，客户端显示错误提示，错误代码为 10054，查阅错误代码表得知该错误表示“Connection reset by peer”，当服务器关闭了连接时会导致此错误。

#### ▪ 服务器如何可以退出循环？

服务器运行的过程中，如果有按键按下，那么服务器将在完成一次完整的数据收发之后退出循环。

这是因为服务器的循环条件为 `!_kbhit()`，这是一个**非阻塞**的函数，若有键盘输入则返回非 0 值，否则返回 0。函数体内的 `accept` 函数是**阻塞**的，若连接队列 `msock` 为空，它将等待到一个新的连接建立才会继续执行 `accept` 之后的语句。因此，期间若有键盘输入，必须会执行完当前函数体后才会退出循环。

#### ▪ 截屏（ctrl+alt+PrintScreen）服务器和客户端的运行结果（注明服务器和客户端）：



▲ TCP 服务器运行截图

```
E:\JedData\OneDrive\SYSU_Lessons\计算机网络\计划实验\2.Echo实验\TcpEcho_Client.exe
[ TCP Echo Client ]
客户端开始运行。要退出，请输入q并回车。
=====
输入一行要发送的消息: Hello, world!
[+] 收到的消息:
Sat Mar 09 19:38:36 2019
Hello, world!
=====
输入一行要发送的消息: Computer Networking
[+] 收到的消息:
Sat Mar 09 19:38:50 2019
Computer Networking
=====
输入一行要发送的消息: Bye~
[+] 收到的消息:
Sat Mar 09 19:38:55 2019
Bye~
=====
输入一行要发送的消息: q
客户端运行结束，按任意键退出。
```

▲ TCP 客户端运行截图（共发送了 3 条数据）

▪ 服务器的全部源代码（或自选主要代码）:

```
#include <stdio.h>
#include <time.h>
#include <winsock2.h>
#include <conio.h>
#pragma comment(lib, "ws2_32.lib") // 使用 winsock 2.2 Library
#define BUFLen 2000 // 缓冲区大小

void makeNewMsg(char *msg, char *timestr);

int main()
{
    printf("[TCP Echo Server]\n 正在启动服务器...");

    struct sockaddr_in clsin; // the from address of a client
    struct sockaddr_in sin; // an Internet endpoint address
    SOCKET msock, ssock; // master & slave sockets
    u_short port = 54321; // server port

    WSADATA wsadata;
    WSASStartup(MAKEWORD(2, 0), &wsadata); // 加载 Winsock Library, 使用 2.0 版本

    msock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); // 创建主 socket
    memset(&sin, 0, sizeof(sin)); // 清零
    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 (网络序)
    bind(msock, (struct sockaddr *)&sin, sizeof(sin)); // 绑定监听的 IP 地址和端口号
    listen(msock, 5); // 建立长度为 5 的连接请求队列,
    并把到来的连接请求加入队列等待处理

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

    char buf[BUFLen + 1]; // 建立缓冲区

    while (!_kbhit()) // 检测是否有按键, 如
    { // 果没有则进入循环体执行
        int alen = sizeof(struct sockaddr); // 取到地址结构的长度
        ssock = accept(msock, (struct sockaddr *)&clsin, &alen); // 从 socket, 阻塞地
        从连接请求队列中创建连接

        int recvlen = recv(ssock, buf, BUFLen, 0); // 接收信息
        buf[recvlen] = '\0'; // 保证以空字符结尾
        now = time(NULL);
        char *timestr = ctime(&now);
        printf("收到消息: %s\n", buf);
        printf("收到时间: %s", timestr);
        printf("-----\n");
    }
}
```

```

        makeNewMsg(buf, timestr);           // 构建要发送给客户端的字符串
        send(ssock, buf, strlen(buf) + 1, 0); // 发送信息
        shutdown(ssock, SD_SEND);           // 禁止收发数据
        closesocket(ssock);                 // 关闭从 socket
    }

    closesocket(msock); // 关闭主 socket
    WSACleanup();       // 卸载 winsock Library
    printf("服务器已关闭, 按任意键退出.\n");
    getchar();
    return 0;
}

void makeNewMsg(char *msg, char *timestr)
{
    char tempbuf[BUFLEN + 1];
    strncpy(tempbuf, timestr, BUFLEN);
    strncat(tempbuf, msg, BUFLEN);
    strncpy(msg, tempbuf, BUFLEN);
}

```

▪ 客户端的全部源代码（或自选主要代码）：

```

#include <stdio.h>
#include <winsock2.h>
#define BUFLEN 2000           // 缓冲区大小
#pragma comment(lib, "ws2_32.lib") // 使用 Winsock 2.0 Library

int connectAndSend();

int main()
{
    printf("[TCP Echo Client]\n 客户端开始运行。要退出, 请输入 q 并回车.\n");
    printf("===== \n");

    while (connectAndSend());

    printf("客户端运行结束, 按任意键退出.\n");
    getchar();
    return 0;
}

/* ***** connectAndSend() 函数说明 *****
 * 该函数执行完整的建立连接和断开连接过程,
 * 写在一个函数中的好处是可以仅运行一次程序
 * 就可以发送多条信息。
 *
 * 本函数的返回值: 若用户输入了 q 或遇到错误
 * 则返回 0; 其他情况均返回 1。
 */
int connectAndSend()
{
    char *host = "127.0.0.1";
    u_short serverport = 54321;
    struct sockaddr_in sin;
    char buf[BUFLEN + 1];
    SOCKET sock;

    WSADATA wsadata;
    WSStartup(MAKEWORD(2, 0), &wsadata);

    sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = inet_addr(host);
    sin.sin_port = htons(serverport);
    int ret = connect(sock, (struct sockaddr *)&sin, sizeof(sin));
    if (ret == -1)
    {
        printf("[ - ] 连接到服务器失败! \n");
        closesocket(sock);
        WSACleanup();
        return 0;
    }

    printf("输入一行要发送的消息: ");
    fgets(buf, BUFLEN, stdin);
}

```

```

buf[strlen(buf) - 1] = '\0'; // 把换行符替换为空字符
if (buf[1] == '\0' && (buf[0] == 'q' || buf[0] == 'Q'))
{
    closesocket(sock);
    WSACleanup();
    return 0;
} // 用户退出

send(sock, buf, strlen(buf) + 1, 0); // 发送信息，信息末尾不带换行符
int recvlen = recv(sock, buf, BUFLen, 0); // 接收信息
if (recvlen == SOCKET_ERROR)
{
    printf("[ - ] Error: %ld.\n", GetLastError());
}
else if (recvlen == 0)
{
    printf("[ - ] Server closed!");
}
else if (recvlen > 0)
{
    printf("[ + ] 收到的消息: \n");
    buf[recvlen] = '\0';
    printf("%s\n", buf);
    printf("-----\n");
}
closesocket(sock);
WSACleanup();
return 1;
}

```

## (2) 编写 TCP Echo 增强程序

### ■ 实验要求:

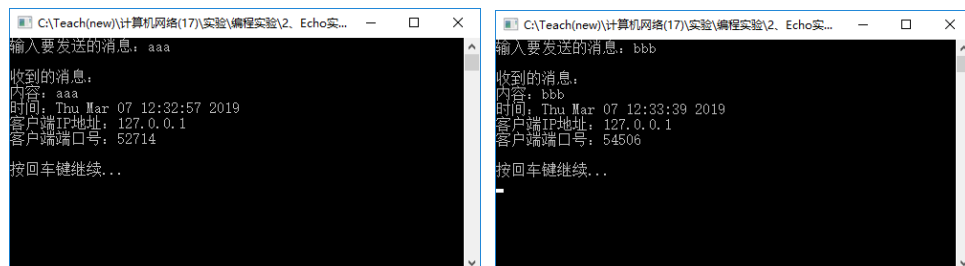
在(1)的基础上，**服务器**在收到客户端的消息时显示服务器的当前时间、客户端的 IP 地址、客户端的端口号和客户端发来的信息，并把它们一并返回给客户端。

**客户端**在发送消息后把服务器发回给它的消息显示出来。\*客户端程序与(1)同，不用修改

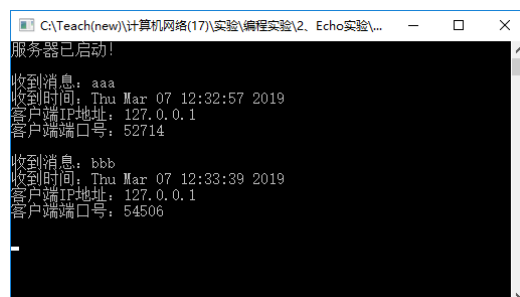
要求**服务器**直接从 accept() 的参数 fsin 中得到客户端的 IP 地址和端口号。注：服务器获取 IP 地址后要求直接使用 s\_un\_b 的四个分量得到 IP 地址，不能使用函数 inet\_ntoa() 转换 IP 地址。

### ■ 参考运行截屏:

客户端（两次运行）



服务器



### ■ 截屏服务器和客户端的运行结果(注明服务器和客户端):

```
E:\JedData\OneDrive\SVSU_Lessons\计算机网络\计网实验\2.Echo实验\TcpEcho_Server_Enhanced.exe
[TCP Echo Server Enhanced]
正在启动服务器...成功。
服务器启动时间: Sun Mar 10 19:45:43 2019
=====
收到消息: AAAAA
收到时间: Sun Mar 10 19:45:59 2019
客户端IP地址: 127.0.0.1
客户端端口号: 18985
=====
收到消息: 12345
收到时间: Sun Mar 10 19:46:02 2019
客户端IP地址: 127.0.0.1
客户端端口号: 19241
=====
收到消息: SOCKET is fun!
收到时间: Sun Mar 10 19:46:07 2019
客户端IP地址: 127.0.0.1
客户端端口号: 19497
=====
```

▲ TCP 增强型服务器运行截图

```
E:\JedData\OneDrive\SVSU_Lessons\计算机网络\计网实验\2.Echo实验\TcpEcho_Client.exe
[TCP Echo Client]
客户端开始运行。要退出, 请输入q并回车。
=====
输入一行要发送的消息: AAAAA
[+] 收到的消息:
内容: AAAAA
时间: Sun Mar 10 19:45:59 2019
客户端IP地址: 127.0.0.1
客户端端口号: 18985
=====
输入一行要发送的消息: 12345
[+] 收到的消息:
内容: 12345
时间: Sun Mar 10 19:46:02 2019
客户端IP地址: 127.0.0.1
客户端端口号: 19241
=====
输入一行要发送的消息: SOCKET is fun!
[+] 收到的消息:
内容: SOCKET is fun!
时间: Sun Mar 10 19:46:07 2019
客户端IP地址: 127.0.0.1
客户端端口号: 19497
=====
输入一行要发送的消息:
```

▲ TCP 客户端运行截图（共发送了 3 条数据）

- 服务器的全部源代码（或自选主要代码）:

```
#include <stdio.h>
#include <time.h>
#include <winsock2.h>
#include <conio.h>
#include <string>
#include <sstream>

#pragma comment(lib, "ws2_32.lib") // 使用 winsock 2.2 Library
#define BUFLen 2000 // 缓冲区大小

void makeNewMsg_enhanced(char *msg, char *timestr, struct sockaddr_in clsin);

int main()
{
    printf("[TCP Echo Server Enhanced]\n 正在启动服务器...");

    struct sockaddr_in clsin; // the from address of a client
    struct sockaddr_in sin; // an Internet endpoint address
    SOCKET msock, ssock; // master & slave sockets
    u_short port = 54321; // server port

    WSADATA wsadata;
    WSASStartup(MAKEWORD(2, 0), &wsadata); // 加载 Winsock Library, 使用 2.0 版本

    msock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); // 创建主 socket
    memset(&sin, 0, sizeof(sin)); // 清零
    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(网络序)
bind(msock, (struct sockaddr *)&sin, sizeof(sin)); // 绑定监听的 IP 地址和端口号
listen(msock, 5); // 建立长度为5的连接请求队列, 并
// 把到来的连接请求加入队列等待处理

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

char buf[BUFLen + 1]; // 建立缓冲区

while (!kbhit())
{
    // 检测是否有按键, 如果没有则进入循环体执行
    int alen = sizeof(struct sockaddr); // 取到地址结构的长度
    ssock = accept(msock, (struct sockaddr *)&clsin, &alen); // 从 socket, 阻塞地从连接请求队列中创建连接

    int recvlen = recv(ssock, buf, BUFLen, 0); // 接收信息
    buf[recvlen] = '\0'; // 保证以空字符结尾
    now = time(NULL);
    char *timestr = ctime(&now);
    printf("收到消息: %s\n", buf);
    printf("收到时间: %s", timestr);
    printf("客户端 IP 地址: %u.%u.%u.%u\n", clsin.sin_addr.S_un.S_un_b.s_b1,
clsin.sin_addr.S_un.S_un_b.s_b2, clsin.sin_addr.S_un.S_un_b.s_b3,
clsin.sin_addr.S_un.S_un_b.s_b4);
    printf("客户端端口号: %hu\n", clsin.sin_port);
    printf("-----\n");

    makeNewMsg_enhanced(buf, timestr, clsin); // 构建要发送给客户端的字符串
    send(ssock, buf, strlen(buf) + 1, 0); // 发送信息
    shutdown(ssock, SD_SEND); // 禁止收发数据
    closesocket(ssock); // 关闭从 socket
}

closesocket(msock); // 关闭主 socket
WSACleanup(); // 卸载 winsock Library
printf("服务器已关闭, 按任意键退出.\n");
getchar();
return 0;
}

/* 构造要发给客户端的字符串, 用到了 C++ 特性 */
void makeNewMsg_enhanced(char *msg, char *timestr, struct sockaddr_in clsin)
{
    using std::endl;
    std::ostringstream os;
    os << "内容: " << msg << endl;
    os << "时间: " << timestr << std::flush;
    os << "客户端 IP 地址: " << (int)clsin.sin_addr.S_un.S_un_b.s_b1 << '.' <<
(int)clsin.sin_addr.S_un.S_un_b.s_b2
    << '.' << (int)clsin.sin_addr.S_un.S_un_b.s_b3 << '.' <<
(int)clsin.sin_addr.S_un.S_un_b.s_b4 << endl;
    os << "客户端端口号: " << clsin.sin_port << std::flush;
    strncpy(msg, os.str().c_str(), BUFLen);
}

```

### (3) 编写 UDP Echo 增强程序

#### ■ 实验要求:

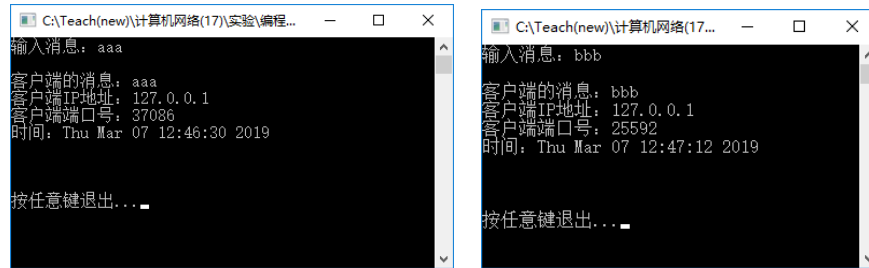
修改 UDP 例程, 完成 Echo 功能, 即当客户端发来消息时, 服务器显示出服务器的当前时间、客户端的 IP、客户端的端口号和客户发来的信息, 并把它一并发回给客户端, 客户端然后把它们显示出来。

服务器可以直接从 `recvfrom()` 的参数 `from` 中得到客户端的 IP 地址和端口号, 并且服务器用 `sendto()` 发回给客户端消息时可以直接用该参数 `from` 作为参数 `toAddr`。可以使用 `inet_ntoa()` 转换客户端 IP 地址。

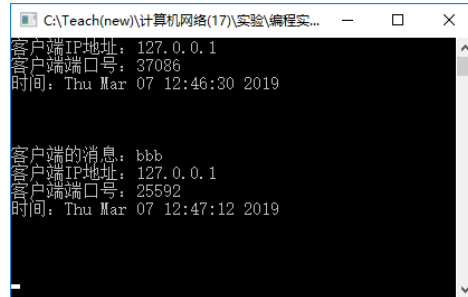
客户端程序的 `recvfrom()` 可以直接使用原来 `sendto` 使用的 `sock`。该 `sock` 已经绑定了客户端的 IP 地址和端口号, 客户端可以直接用来接收数据。

▪ 参考运行截屏：

客户端（两次运行）



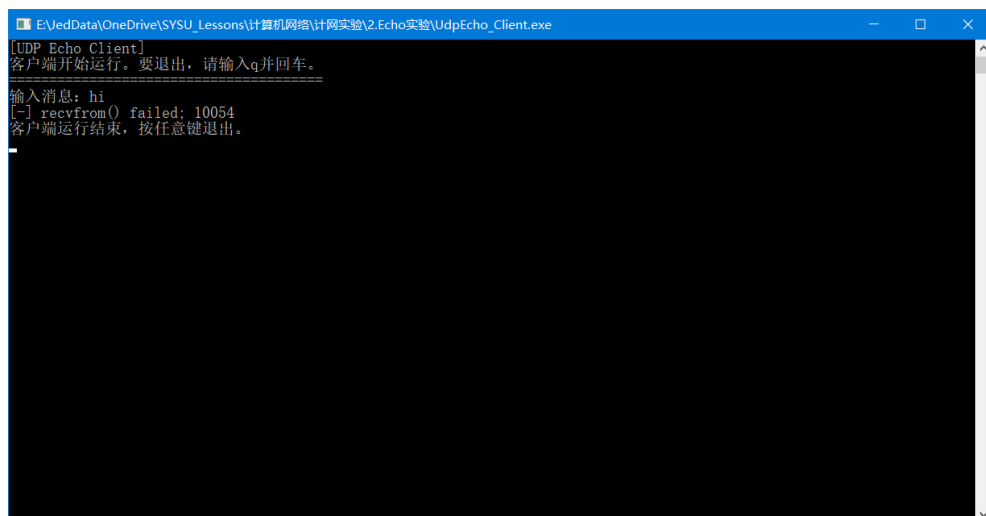
服务器：



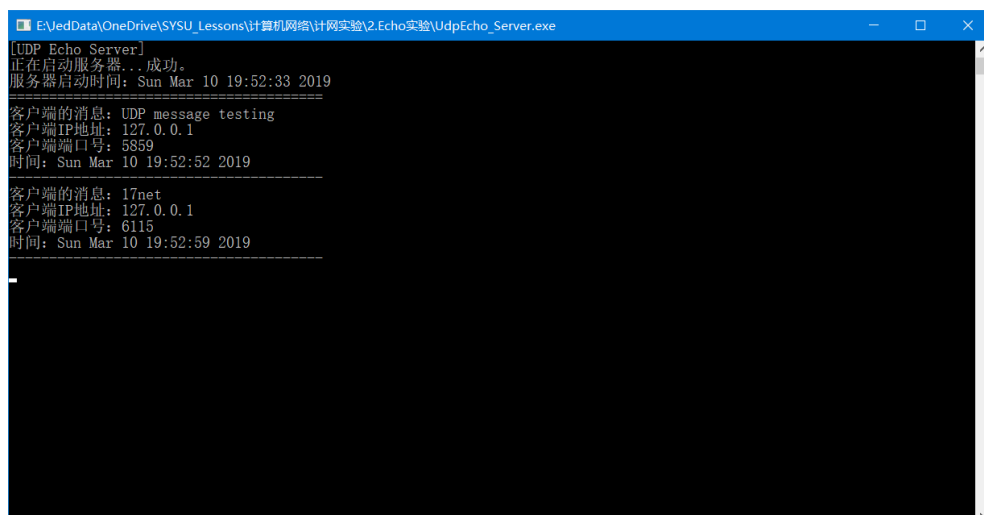
▪ 只运行客户端程序而不运行服务器程序会出现什么错误，截屏并说明原因。

与 TCP 不同，UDP 发送消息前不需要建立连接。

若服务器没有运行，仅仅运行客户端并发送消息，会导致代码为 10054 的错误。该错误与 TCP 中第二种情况相同，代表“Connection reset by peer”。错误截图如下：

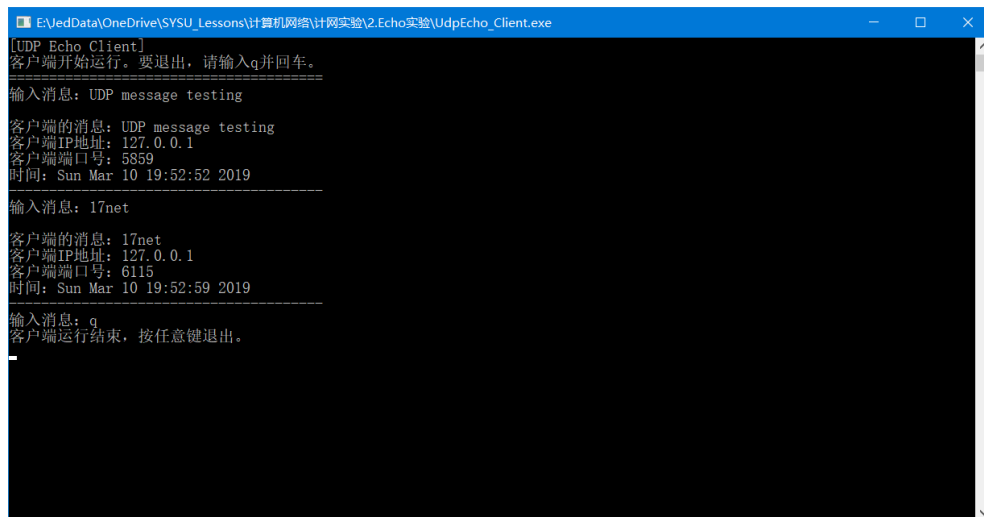


▪ 截屏服务器和客户端的运行结果（注明客户端和服务端）：





### ▲ UDP 增强型服务器运行截图



```
[UDP Echo Client]
客户端开始运行。要退出，请输入q并回车。
=====
输入消息: UDP message testing
客户端的消息: UDP message testing
客户端IP地址: 127.0.0.1
客户端端口号: 5859
时间: Sun Mar 10 19:52:52 2019
-----
输入消息: 17net
客户端的消息: 17net
客户端IP地址: 127.0.0.1
客户端端口号: 6115
时间: Sun Mar 10 19:52:59 2019
-----
输入消息: q
客户端运行结束，按任意键退出。
```

### ▲ UDP 客户端运行截图（发送了两条消息）

#### ■ 服务器的全部源代码（或自选主要代码）:

```
#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <string.h>
#include <time.h>
#include <conio.h>
#include <sstream>

#pragma comment(lib, "ws2_32.lib") // 加载Winsock 2.2 Library
#define BUFLen 2000 // 缓冲区大小

void makeNewMsg(char *msg, char *timestr, struct sockaddr_in from);
int main()
{
    printf("[UDP Echo Server]\n 正在启动服务器...");

    u_short port = 54322; // server port to connect
    struct sockaddr_in sin; // an Internet endpoint address
    struct sockaddr_in from; // sender address
    int fromsize = sizeof(from);
    SOCKET sock; // socket descriptor

    WSADATA wsadata;
    WSASStartup(MAKEWORD(2, 2), &wsadata); // 加载winsock Library

    sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP); // 创建UDP socket
    memset(&sin, 0, sizeof(sin)); // 清零
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(port);
    bind(sock, (struct sockaddr *)&sin, sizeof(sin)); // 绑定本地端口号（和本地IP地址）

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

    char buf[BUFLen + 1]; // 建立缓冲区

    while (!_kbhit())
    {
        int recvlen = recvfrom(sock, buf, BUFLen, 0, (SOCKADDR *)&from, &fromsize);
        if (recvlen == SOCKET_ERROR)
        {
            printf("[+] recvfrom() failed; %d\n", WSAGetLastError());
            break;
        }
        else if (recvlen == 0)
        {
            break;
        }
    }
}
```

```

        else
        {
            time_t now = time(NULL);
            char *timestr = ctime(&now);
            buf[recvlen] = '\0'; // 保证以空字符结尾
            printf("客户端的消息: %s\n", buf);
            printf("客户端 IP 地址: %u.%u.%u.%u\n", from.sin_addr.S_un.S_un_b.s_b1,
from.sin_addr.S_un.S_un_b.s_b2, from.sin_addr.S_un.S_un_b.s_b3,
from.sin_addr.S_un.S_un_b.s_b4);
            printf("客户端端口号: %hu\n", from.sin_port);
            printf("时间: %s", timestr);
            printf("-----\n");

            makeNewMsg(buf, timestr, from);
            sendto(sock, buf, BUFLen, 0, (SOCKADDR *)&from, sizeof(from));
        }
    }
    closesocket(sock);
    WSACleanup();

    printf("服务器已关闭, 按任意键退出.\n");
    getchar();
    return 0;
}

/* 构造要发给客户端的字符串, 用到了C++特性 */
void makeNewMsg(char *msg, char *timestr, struct sockaddr_in from)
{
    using std::endl;
    std::ostringstream os;
    os << "客户端的消息: " << msg << endl;
    os << "客户端 IP 地址: " << (int)from.sin_addr.S_un.S_un_b.s_b1 << '.' <<
(int)from.sin_addr.S_un.S_un_b.s_b2
    << '.' << (int)from.sin_addr.S_un.S_un_b.s_b3 << '.' <<
(int)from.sin_addr.S_un.S_un_b.s_b4 << endl;
    os << "客户端端口号: " << from.sin_port << endl;
    os << "时间: " << timestr << std::flush;
    strncpy(msg, os.str().c_str(), BUFLen);
}

```

▪ 客户端的全部源代码（或自选主要代码）：

```

#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <string.h>
#include <time.h>

#define BUFLen 2000 // 缓冲区大小
#pragma comment(lib, "ws2_32.lib") // 加载 winsock 2.2 Llibrary

int connectAndSendto();
int main()
{
    printf("[UDP Echo Client]\n 客户端开始运行。要退出, 请输入 q 并回车.\n");
    printf("=====\n");

    while (connectAndSendto());

    printf("客户端运行结束, 按任意键退出.\n");
    getchar();
    return 0;
}

int connectAndSendto()
{
    char *host = "127.0.0.1";
    u_short serverport = 54322;
    struct sockaddr_in toAddr;
    SOCKET sock;

    WSADATA wsadata;
    WSAStartup(MAKEWORD(2, 2), &wsadata);

    sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
    memset(&toAddr, 0, sizeof(toAddr));
    toAddr.sin_family = AF_INET;

```

```

toAddr.sin_port = htons(serverport);
toAddr.sin_addr.s_addr = inet_addr(host);

char buf[BUFLEN + 1]; // 建立缓冲区

printf("输入消息: ");
fgets(buf, BUFLEN, stdin);
buf[strlen(buf) - 1] = '\0'; // 将换行符替换为空字符
if (buf[1] == '\0' && (buf[0] == 'q' || buf[0] == 'Q'))
{
    closesocket(sock);
    WSACleanup();
    return 0;
}
int sendlen = sendto(sock, buf, BUFLEN, 0, (SOCKADDR *)&toAddr, sizeof(toAddr));
if (sendlen == SOCKET_ERROR)
{
    printf("[-] 发送失败, 错误号: %d", WSAGetLastError());
}
else
{
    int tosize = sizeof(toAddr);
    int recvlen = recvfrom(sock, buf, BUFLEN, 0, (SOCKADDR *)&toAddr, &tosize);
    if (recvlen == SOCKET_ERROR)
    {
        printf("[-] recvfrom() failed; %d\n", WSAGetLastError());
        return 0;
    }
    else if (recvlen == 0)
    {
        return 0;
    }
    else
    {
        printf("\n%s", buf);
        printf("-----\n");
    }
}

closesocket(sock);
WSACleanup();
return 1;
}

```

#### 【实验体会】

1. **字符串处理。**在完成 TCP 增强型服务器时,需要进行大量的字符串处理。C 语言对于 char 数组保存的 C 风格字符串的处理能力相当弱,尤其是拷贝、拼接这种处理,十分麻烦。因此我使用了一些 C++ 的特性,包括 string 类和 stringstream 流,在传送时再将其转换为 C 风格字符串,大大降低了字符串操作的难度。
2. **对 SOCKET 的理解。**通过本次实验,我从零开始学习了 SOCKET 的概念,以及 C 中的 SOCKET 编程接口,并尝试动手使用 SOCKET 接口发送 TCP 或 UDP 数据。同时,通过阅读文档,我对 SOCKET 接口中的函数、结构体(如 struct sockaddr\_in)有了更深刻的理解。
3. **阻塞与非阻塞。**在 TCP 服务器的代码中,用到了两个函数。一是 kbhit(), 这是一个非阻塞的函数,用来检测是否有键盘输入。它与 getch() 不同,前者非阻塞,后者阻塞。非阻塞的意思是,执行到该函数时,若不满足条件,那么函数不会阻挡程序的执行,若满足条件,则返回相应的值。另一个函数是 accept(), 它是 SOCKET 接口的一部分。accept 函数是一个阻塞的函数,如果在连接请求队列中有连接请求,则接受连接请求并建立连接,返回该连接的套接字,否则,本语句被阻塞直到队列非空。理解了以上概念,可以更好地把握函数执行的流程。
4. **-lws2\_32。**在 Windows 的 gcc 编译器下,必须在执行链接时使用参数 "-lws2\_32",否则会无法编译。