实验报告

院 系 数据科学与计算机学院 班 级 18 级超算方向 学 号 18340161 姓 名 田蕊

【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）
- https://www.runoob.com/cprogramming/c-function-sscanf.html (sscanf)
- sprintf 可以用于合并多个字符串和整数等：

```
char  buffer[50];
char *test = "Hello world!";
sprintf(buffer,"The length of %s is %d",test,strlen(test));/*赋予数值*/
```

【实验环境】

- **172.18.187.251 为校园网内的服务器地址，需要先用 VPN 连入校园网才能访问。**
- Windows + VS 2012 http://172.18.187.251/netdisk/default.aspx?vm=18net
- 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

【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。

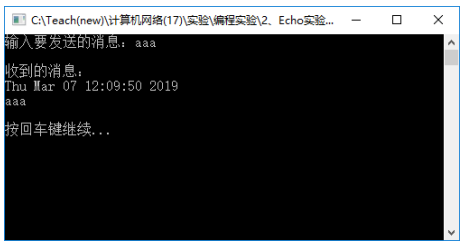
(1)编写 TCP Echo 程序

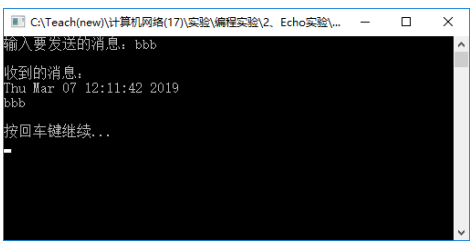
- **实验要求：**

  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。

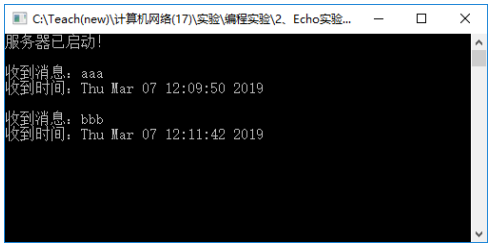
- **参考运行截屏：**

  客户端（两次运行）

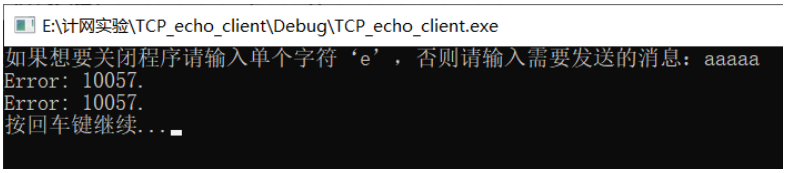




  服务器：



- **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。**



截图结果如上所示，根据查阅的资料，Error：10057 表示的错误是由于套接字没有连接并且没有提供地址，发送或接收数据的请求没有被接受。因为在这次运行实验中，只运行客户端程序而不运行服务器程序，所以在运行到 connect 函数时发送连接请求失败，返回 SOCKET_ERROR，调用函数 WASGetLastError 取得错误代码。屏幕上将会显示错误代码。

- **服务器如何可以退出循环？**

根据服务器代码

```
while (!_kbhit()) {
    alen = sizeof(struct sockaddr);
    ssock = accept(msock, (struct sockaddr *)&fsin, &alen);

    cc = recv(ssock, buf, BUFLEN, 0);
    if (!strcmp(buf, "Exit")) {
        (void)closesocket(ssock);
        break;
    }
    (void)time(&now);
    pts = ctime(&now);
    if (cc == SOCKET_ERROR) {
        printf("Error: %d.\n", GetLastError());
    }
    else if (cc == 0) {
        printf("Server closed!", buf);
    }
    else if (cc > 0) {
        buf[cc] = '\0';
        printf("收到消息：%s\n", buf);
        printf("收到时间：%s\n\n", pts);
        int dd = send(ssock, pts, strlen(pts), 0);
        dd = send(ssock, buf, cc, 0);
    }
    (void)closesocket(ssock);
}
```

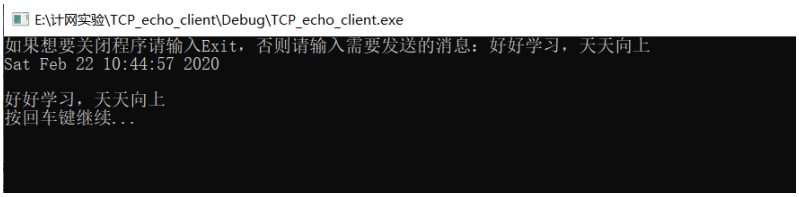
按照正常的逻辑，循环控制变量是括号中的_kbhit()函数，也就是说，只要随意按一个按键，就不满足循环条件，循环就会退出。但是在实际地操作过程中，我们发现这并不能奏效，因为在第二个红框中我们可以看到，连接队列中没有请求时，程序将会阻塞在这条语句，并不会向下执行，一旦队列非空才会继续向下进行。所以如果想要靠按下任意按键得方式结束循环，就必须卡在程序刚好运行到循环判断语句时按下任意按键，然而和电脑比手速我就没有赢过，所以我们这里采用一个小技巧。在客户端部分我们加入一段提示语句“如果想要关闭程序请输入 Exit”，所以当用户端输入 Exit 时，连接队列非空，字符将会被传送到 buf 中，这时我们做出一个判断，如果接收到的字符串与“Exit”一致，则结束循环断开连接。

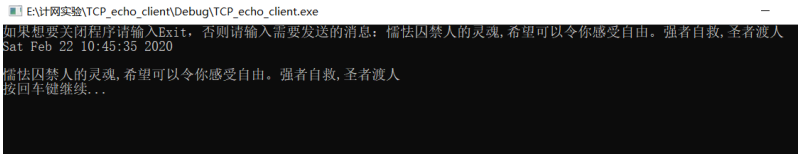
中山大學 SUN YAT-SEN UNIVERSITY

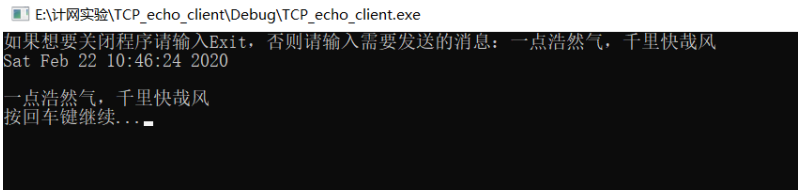
实验报告

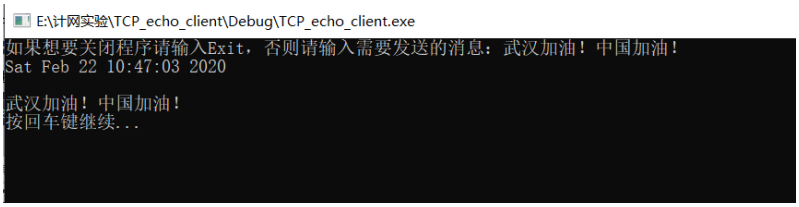
- **截屏（ctrl+alt+PrintScreen）服务器和客户端的运行结果（注明客户端和服务器）：**

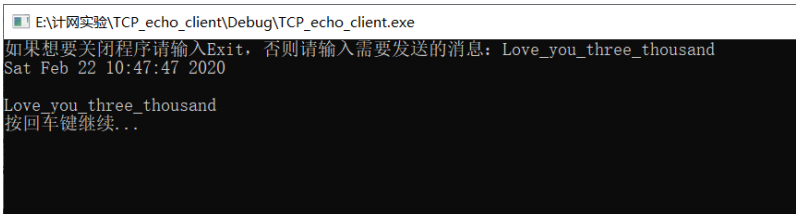
客户端：

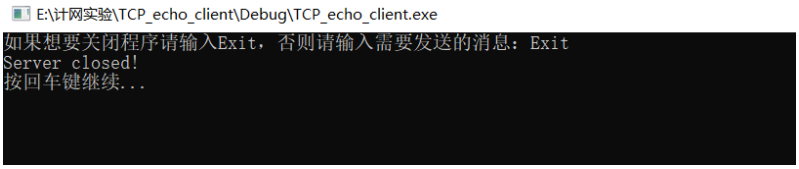




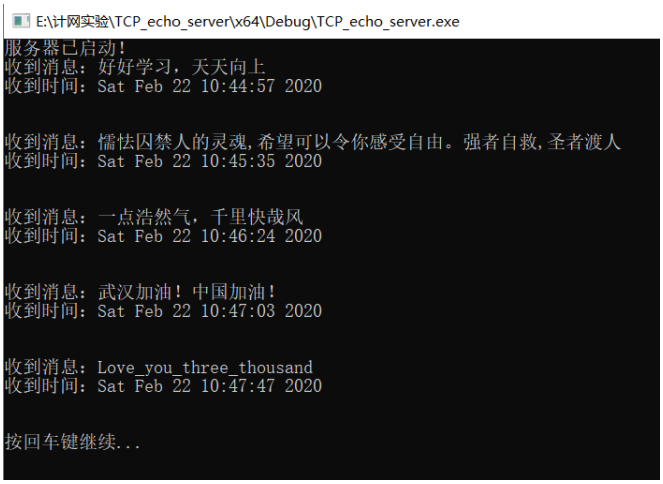








服务器：



▪ 服务器的全部源代码（或自选主要代码）：

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

#define WSVERS  MAKEWORD(2, 0)
#define _CRT_SECURE_NO_WARNINGS
#define BUFLEN 2000
#pragma comment(lib,"ws2_32.lib")  //使用winsock 2.2 library

void main(int argc, char *argv[]) {
    struct  sockaddr_in fsin;       /* the from address of a client   */
    SOCKET  msock, ssock;           /* master & slave sockets         */
    WSADATA wsadata;
    char    *service = "50500";
    struct  sockaddr_in sin;        /* an Internet endpoint address     */
    int     alen;                   /* from-address length              */
    char    *pts;                   /* pointer to time string           */
    time_t  now;                    /* current time                     */

    int cc;
    char buf[BUFLEN + 1];
    char buf_send[BUFLEN + 1];

    WSAStartup(WSVERS, &wsadata);                       // 加载winsock library。WSVERS指明请求使用的版本。
                                                        //wsadata返回系统实际支持的最高版本
    msock = 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
                                                        // 返回：要监听套接字的描述符或INVALID_SOCKET

    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((u_short)atoi(service));       // 监听的端口号。atoi--把ascii转化为int，
                                                        //htons--主机序到网络序(host to network，s-short 16位)
    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。
    printf("服务器已启动！\n");
    while (!_kbhit()) {
        alen = sizeof(struct sockaddr);                             // 取到地址结构的长度
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen); // 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。fsin包含客户端IP地址和端口号
        cc = recv(ssock, buf, BUFLEN, 0);                           // buf里面保存的是接收到的字符串

        (void)time(&now);
        pts = ctime(&now);
        if (cc == SOCKET_ERROR) {
            printf("Error: %d.\n", GetLastError());
        }
        else if (cc == 0) {
            printf("Server closed!", buf);
        }
        else if (cc > 0) {
            buf[cc] = '\0';
            if (!strcmp(buf, "Exit")) {
                (void)closesocket(ssock);
                break;
            }
            printf("收到消息：%s\n", buf);
            printf("收到时间：%s\n\n", pts);
            int dd = send(ssock, pts, strlen(pts), 0);
            dd = send(ssock, buf, cc, 0);
        }
        (void)closesocket(ssock);
    }
    (void)closesocket(msock);
    WSACleanup();
    printf("按回车键继续...");
    getchar();                                      // 等待任意按键
    getchar();
}
```

▪ 客户端的全部源代码（或自选主要代码）：

```c
#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <string.h>
 #define _CRT_SECURE_NO_WARNINGS
 #define BUFLEN      2000                // 缓冲区大小
 #define WSVERS      MAKEWORD(2, 0)      // 指明版本2.0
 #pragma comment(lib,"ws2_32.lib")       // 使用winsock 2.0 Llibrary


void main(int argc, char *argv[]) {
    char    *host = "127.0.0.1";        /* server IP to connect         */
    char    *service = "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;                             /* send character count         */
    int dd;                             /* recv character count         */
    int ee;
    char pts[BUFLEN + 1];
    char buf_r[BUFLEN + 1];

    WSADATA wsadata;
    WSAStartup(WSVERS, &wsadata);                       //加载winsock library。
                                                        //WSVERS为请求的版本，
                                                        //wsadata返回系统实际支持的最高版本
    sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);   //创建套接字

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = inet_addr(host);
    sin.sin_port = htons((u_short)atoi(service));
    int ret = connect(sock, (struct sockaddr *)&sin, sizeof(sin));

    printf("如果想要关闭程序请输入Exit，否则请输入需要发送的消息：");
    scanf("%s", buf);
    cc = send(sock, buf, strlen(buf), 0);               //发送字符串
    dd = recv(sock, pts, BUFLEN, 0);                    //收到字符串，首先受到的是时间
    ee = recv(sock, buf_r, BUFLEN, 0);
    if (dd == SOCKET_ERROR)                             // 出错。其后必须关闭套接字sock
        printf("Error: %d.\n", GetLastError());
    else if (dd == 0) {                                 // 对方正常关闭
        printf("Server closed!\n", pts);
    }
    else if (dd > 0) {
        pts[dd] = '\0';                         // ensure null-termination
        printf("%s\n", pts);                            // 显示所接收的字符串
    }

    if (ee == SOCKET_ERROR)                             // 出错。其后必须关闭套接字sock
        printf("Error: %d.\n", GetLastError());
    /*else if (ee == 0) {                               // 对方正常关闭
        printf("Server closed!", buf_r);
    }*/
    else if (ee > 0) {
        buf_r[ee] = '\0';                           // ensure null-termination
        printf("%s\n", buf_r);                          // 显示所接收的字符串
    }

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

    printf("按回车键继续...");
    getchar();                                          // 等待任意按键
    getchar();
}
```

(2)编写 TCP Echo 增强程序

- **实验要求：**

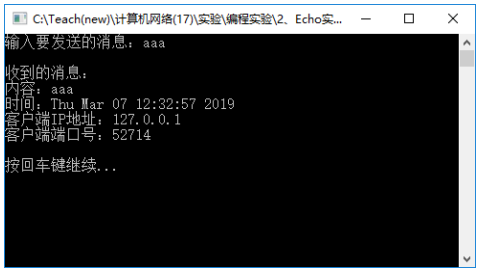
在(1)的基础上，**服务器**在收到客户端的消息时显示服务器的当前时间、客户端的 IP 地址、客户端的端口号和客户端发来的信息，并把它们一并返回给客户端。

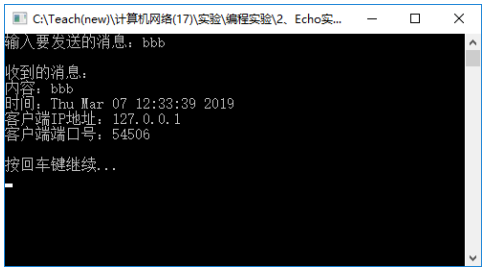
客户端在发送消息后把服务器发回给它的消息显示出来。*客户端程序与(1)同，不用修改

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。

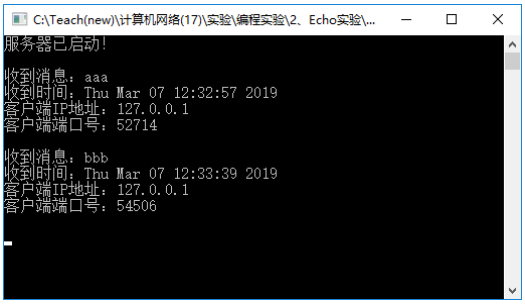
- **参考运行截屏：**

客户端（两次运行）



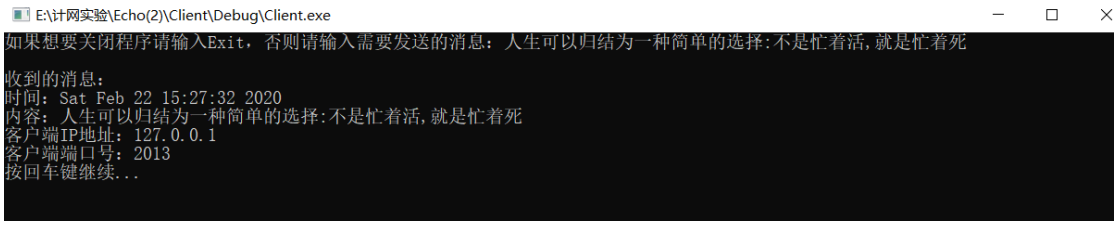


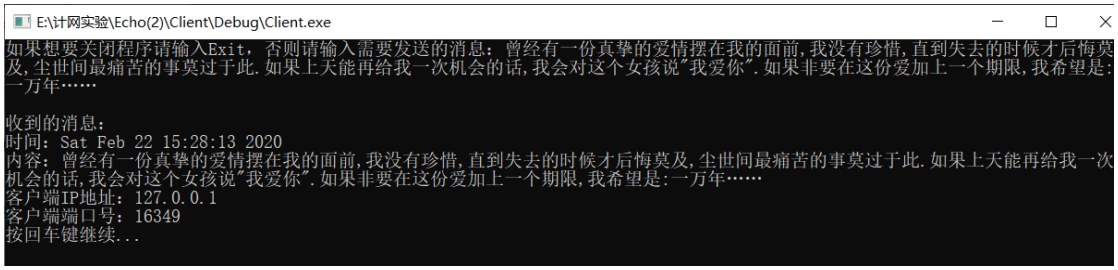
服务器



- **截屏服务器和客户端的运行结果(注明客户端和服务器)：**

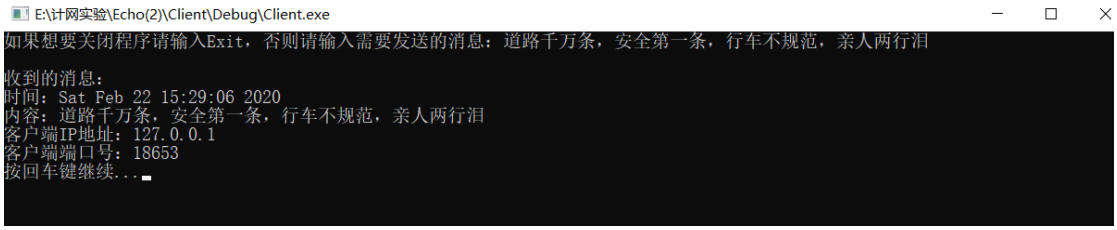
客户端：

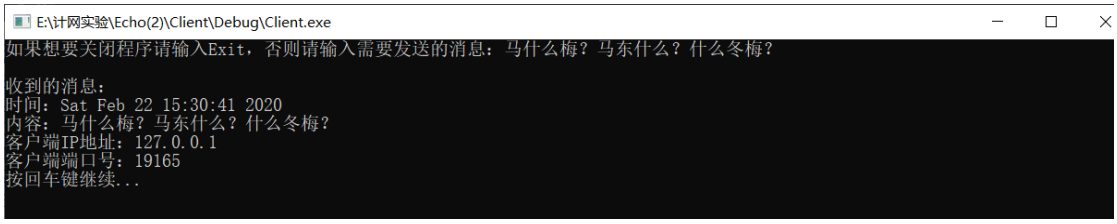


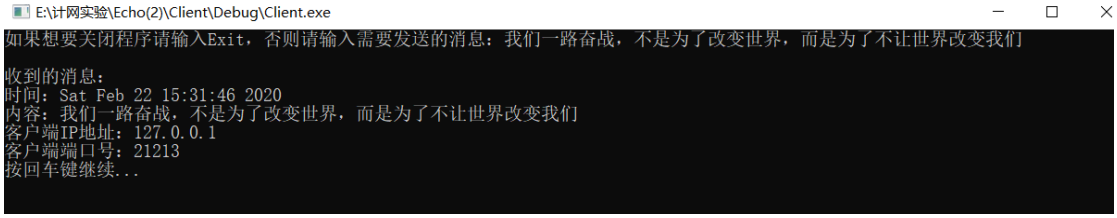


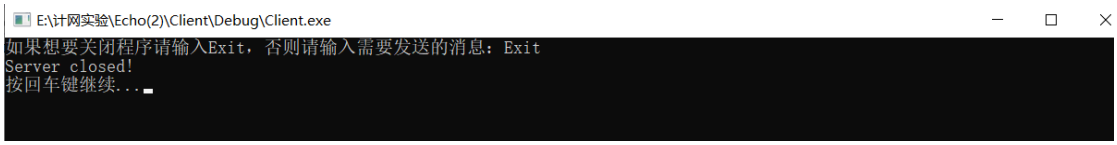
中山大學 SUN YAT-SEN UNIVERSITY

实验报告









服务端：

中山大學 SUN YAT-SEN UNIVERSITY

实验报告

- **服务器的全部源代码（或自选主要代码）：**

```c
#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <time.h>
#include <string.h>
#include <math.h>
#include "conio.h"

#define WSVERS  MAKEWORD(2, 0)
#define _CRT_SECURE_NO_WARNINGS
#define BUFLEN 2000
#pragma comment(lib,"ws2_32.lib")  //使用winsock 2.2 library

void main(int argc, char *argv[]) {
    struct  sockaddr_in fsin;       /* the from address of a client   */
    SOCKET  msock, ssock;           /* master & slave sockets         */
    WSADATA wsadata;
    char    *service = "50500";
    struct  sockaddr_in sin;        /* an Internet endpoint address     */
    int     alen;                   /* from-address length              */
    char    *pts;                   /* pointer to time string           */
    time_t  now;                    /* current time                     */

    int cc;
    char buf[BUFLEN + 1];
    char buf_PORT[BUFLEN + 1];
    unsigned short temp = 0;

    WSAStartup(WSVERS, &wsadata);                       // 加载winsock library。WSVERS指明请求使用的版本。
                                                        //wsadata返回系统实际支持的最高版本
    msock = 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
                                                        // 返回：要监听套接字的描述符或INVALID_SOCKET

    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((u_short)atoi(service));       // 监听的端口号。atoi—把ascii转化为int，
                                                        //htons—主机序到网络序(host to network，s-short 16位)
    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。
    printf("服务器已启动！\n");
    while (!_kbhit()) {
        alen = sizeof(struct sockaddr);                         // 取到地址结构的长度
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen); // 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。fsin包含客户端IP地址和端口号
        unsigned char a = fsin.sin_addr.S_un.S_un_b.s_b1;
        unsigned char b = fsin.sin_addr.S_un.S_un_b.s_b2;
        unsigned char c = fsin.sin_addr.S_un.S_un_b.s_b3;
        unsigned char d = fsin.sin_addr.S_un.S_un_b.s_b4;

        temp = fsin.sin_port;
        sprintf(buf_PORT, "%d", temp);

        cc = recv(ssock, buf, BUFLEN, 0);                       // buf里面保存的是接收到的字符串
        (void)time(&now);
        pts = ctime(&now);
        if (cc == SOCKET_ERROR) {
            printf("Error: %d.\n", GetLastError());
        }
        else if (cc == 0) {
            printf("Server closed!", buf);
        }
        else if (cc > 0) {
            buf[cc] = '\0';
            if (!strcmp(buf, "Exit")) {
                (void)closesocket(ssock);
                break;
            }
            printf("收到消息：%s\n", buf);
            printf("收到时间：%s", pts);
            printf("客户端IP地址：%d.%d.%d.%d\n", a, b, c, d);
            printf("客户端端口号：%d\n\n", fsin.sin_port);
            int dd = send(ssock, pts, strlen(pts), 0);
            dd = send(ssock, buf, cc, 0);
            dd = send(ssock, buf_PORT, strlen(buf_PORT), 0);

        }
        (void)closesocket(ssock);
    }
    (void)closesocket(msock);
    WSACleanup();
    printf("按回车键继续...");
    getchar();                                      // 等待任意按键
    getchar();
}
```

(3)编写 UDP Echo 增强程序

- **实验要求：**

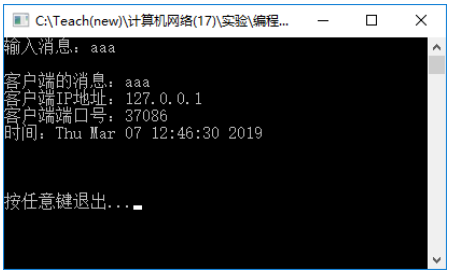
  修改 UDP 例程，完成 Echo 功能，即当客户端发来消息时，服务器显示出服务器的当前时间、客户端的 IP、客户端的端口号和客户发来的信息，并把它们一并发回给客户端，客户端然后把它们显示出来。

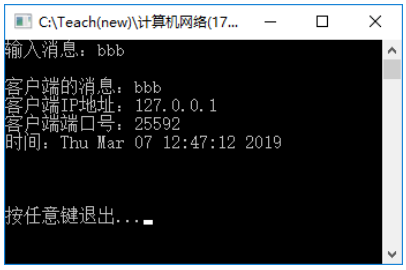
  服务器可以直接从 recvfrom()的参数 from 中得到客户端的 IP 地址和端口号，并且服务器用 sendto()发回给客户端消息时可以直接用该参数 from 作为参数 toAddr。可以使用 inet_ntoa()转换客户端 IP 地址。

  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。

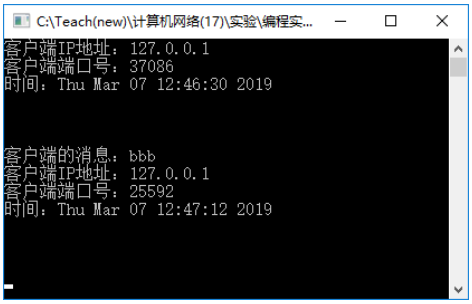
- **参考运行截屏：**

  客户端（两次运行）

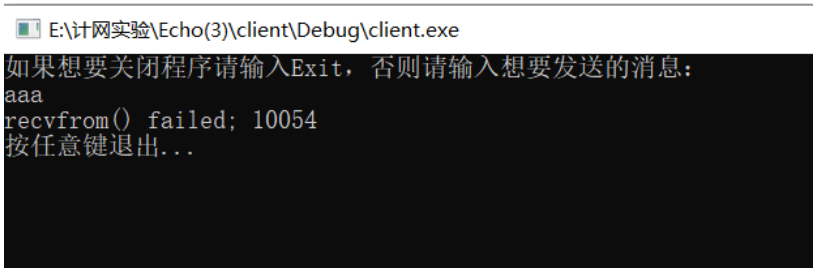




  服务器：



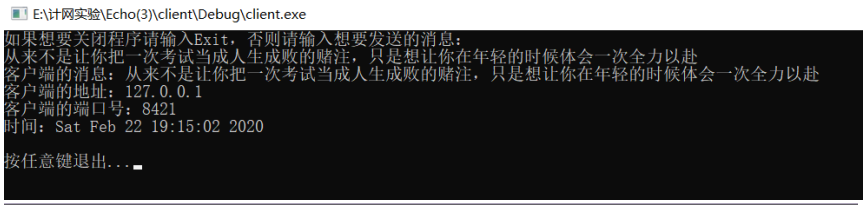
- **只运行客户端程序而不运行服务器程序会出现什么错误，截屏并说明原因。**

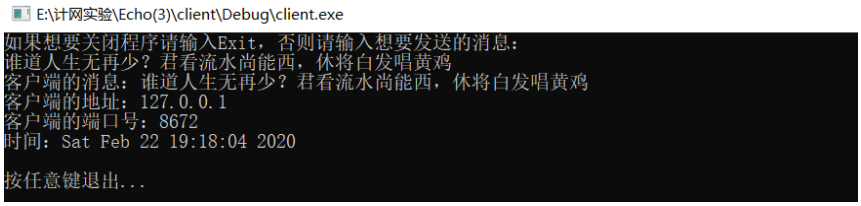


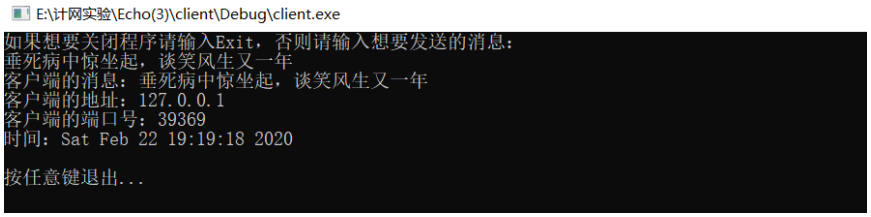
与 TCP 先建立连接再传输数据的模式不同，UDP 在由用户向服务端传输数据的时候是不需要提起建立连接的，给出 IP 地址后直接发送数据。而此时服务端是没有运行的，而客户端立即调用 recvfrom()函数试图接收服务端返回的数据，recvfrom 会返回-1.因为当 UDP Socket 在某次发送后收到一个不可到达的 ICMP 包时，这个错误将在下一个接收中返回，所以上面的套接字在下一次的接收中返回了 SOCKET_ERROR。

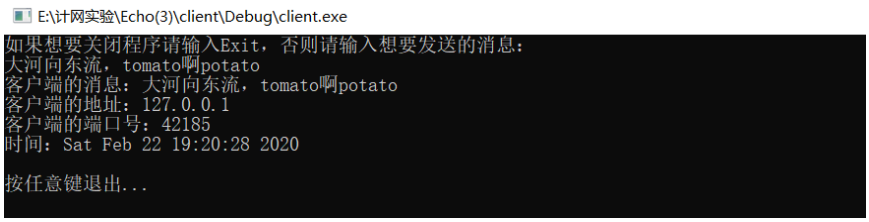
- **截屏服务器和客户端的运行结果（注明客户端和服务器）：**

客户端：









服务端：

- 服务器的全部源代码（或自选主要代码）：

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

#define _CRT_SECURE_NO_WARNINGS
#define BUFLEN      2000                // 缓冲区大小
#define WSVERS      MAKEWORD(2, 2)      // 指明版本2.2
#pragma comment(lib,"ws2_32.lib")       // 加载winsock 2.2 Llibrary

void main(int argc, char *argv[]) {
    char   *host = "127.0.0.1";     /* server IP Address to connect */
    char   *service = "50500";      /* server port to connect       */
    struct sockaddr_in sin;         /* an Internet endpoint address */
    struct sockaddr_in from;        /* sender address               */
    struct sockaddr_in toAddr;
    int    fromsize = sizeof(from);
    char*   pts;
    char   buf[BUFLEN + 1];         /* buffer for one line of text  */
    char   back[BUFLEN + 1];
    SOCKET  sock;                   /* socket descriptor            */
    int cc;                         /* recv character count         */
    int dd = 0,ee = 0;
    time_t now;

    WSADATA wsadata;
    WSAStartup(WSVERS, &wsadata);   /* 加载winsock library，WSVERS为请求版本,wsadata返回系统实际支持的最高版本。    */
    sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP); // 创建UDP套接字, 参数：因特网协议簇(family)，数据报套接字，UDP协议号，
                                                     //返回：要监听套接字的描述符或INVALID_SOCKET
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;                    // 绑定(监听)所有的接口。
    sin.sin_port = htons((u_short)atoi(service));        // 绑定指定接口。
                                                         //atoi--把ascii转化为int，
                                                         //htons -- 主机序(host)转化为网络序(network)， 为short类型。
    bind(sock, (struct sockaddr *)&sin, sizeof(sin));    // 绑定本地端口号（和本地IP地址）
    printf("服务器已启动！\n");
    while (!_kbhit()) {
        cc = recvfrom(sock, buf, BUFLEN, 0, (SOCKADDR *)&from, &fromsize);
        if (cc == SOCKET_ERROR) {
            printf("recvfrom() failed; %d\n", WSAGetLastError());
            break;
        }
        else if (cc == 0)
            break;
        else {
            buf[cc] = '\0';
            if (!strcmp(buf, "Exit")) {
                break;
            }
            printf("客户端的消息：%s\n", buf);

        }
        printf("客户端的地址：%s\n", inet_ntoa(from.sin_addr));
        printf("客户端的端口号：%d\n", from.sin_port);
        (void)time(&now);
        pts = ctime(&now);
        printf("时间：%s\n\n", pts);

         /*这里是把信息传送回去*/
        memset(&toAddr, 0, sizeof(toAddr));
        toAddr.sin_family = AF_INET;
        toAddr.sin_port = from.sin_port;
        toAddr.sin_addr = from.sin_addr;

        sprintf(back, "客户端的消息：%s\n客户端的地址：%s\n客户端的端口号：%d\n时间：%s",
            buf, inet_ntoa(from.sin_addr), from.sin_port, pts);

        ee = sendto(sock, back, sizeof(back), 0, (SOCKADDR *)&toAddr, sizeof(toAddr));

    }
    closesocket(sock);
    WSACleanup();                       /* 卸载某版本的DLL */

    printf("按任意键退出...");
    getchar();
}
```

中山大學 SUN YAT-SEN UNIVERSITY

实验报告

▪ 客户端的全部源代码（或自选主要代码）：

```c
#include <stdlib.h>
#include <stdio.h>
#include <winsock2.h>
#include <string.h>
 #define _CRT_SECURE_NO_WARNINGS
 #define BUFLEN      5000                // 缓冲区大小
 #define WSVERS      MAKEWORD(2, 2)      // 指明版本2.2
 #pragma comment(lib,"ws2_32.lib")       // 加载winsock 2.2 Llibrary

void main(int argc, char *argv[]) {
    char    *host = "127.0.0.1";        /* server IP to connect         */
    char    *service = "50500";         /* server port to connect       */
    struct sockaddr_in toAddr;          /* an Internet endpoint address */
    int     toAddr_size = sizeof(toAddr);
    char    buf[BUFLEN + 1];            /* buffer for one line of text  */
    SOCKET  sock;                       /* socket descriptor            */
    int cc,dd;                              /* recv character count         */
    char    pts[BUFLEN+1];                      /* pointer to time string       */
    char    recv[BUFLEN + 1];

    WSADATA wsadata;
    WSAStartup(WSVERS, &wsadata);       /* 启动某版本Socket的DLL        */

    sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);

    memset(&toAddr, 0, sizeof(toAddr));
    toAddr.sin_family = AF_INET;
    toAddr.sin_port = htons((u_short)atoi(service));    //atoi：把ascii转化为int.
                                                        // htons：主机序(host)转化为网络序(network)， s--short
    toAddr.sin_addr.s_addr = inet_addr(host);           //如果host为域名，
                                                        //需要先用函数gethostbyname把域名转化为IP地址

    printf("如果想要关闭程序请输入Exit，否则请输入想要发送的消息：\n");
    scanf("%s", buf);
    cc = sendto(sock, buf, strlen(buf), 0, (SOCKADDR *)&toAddr, sizeof(toAddr));   //送出信息
    if (cc == SOCKET_ERROR) {
        printf("发送失败，错误号：%d\n", WSAGetLastError());
    }

    if (strcmp(buf, "Exit")) {
        dd = recvfrom(sock, recv, BUFLEN, 0, (SOCKADDR *)&toAddr, &toAddr_size);
        if (dd == SOCKET_ERROR) {
            printf("recvfrom() failed; %d\n", WSAGetLastError());
        }
        else {
            recv[dd] = '\0';
            printf("%s\n", recv);
        }
    }
    closesocket(sock);
    WSACleanup();                       /* 卸载某版本的DLL */

    printf("按任意键退出...");
    getchar();
    getchar();
}
```

【完成情况】

是否完成以下步骤？（√完成 ×未做）

(1) [ ] (2) [ ] (3) [ ]

中山大學 SUN YAT-SEN UNIVERSITY

实验报告

【实验体会】

遇到的问题和解决方法：

1、 在(1)编写 TCP Echo 程序中，如何实现程序的退出部分遇到了问题。按照正常的逻辑，循环控制变量是括号中的_kbhit()函数，也就是说，只要随意按一个按键，就不满足循环条件，循环就会退出。但是在实际地操作过程中，我们发现这并不能奏效。因为在第二个红框中我们可以看到，连接队列中没有请求时，程序将会阻塞在这条语句，并不会向下执行，一旦队列非空才会继续向下进行。所以如果想要靠按下任意按键得方式结束循环，就必须卡在程序刚好运行到循环判断语句时按下任意按键，然而和电脑比手速我就没有赢过，所以我们这里采用一个小技巧。在客户端部分我们加入一段提示语句“如果想要关闭程序请输入 Exit”，所以当用户端输入 Exit 时，连接队列非空，字符将会被传送到 buf 中，这时我们做出一个判断，如果接收到的字符串与“Exit”一致，则结束循环断开连接。

```
while (!_kbhit()) {
    alen = sizeof(struct sockaddr);
    ssock = accept(msock, (struct sockaddr *)&fsin, &alen);

    cc = recv(ssock, buf, BUFLEN, 0);
    if (!strcmp(buf, "Exit")) {
        (void)closesocket(ssock);
        break;
    }
    (void)time(&now);
    pts = ctime(&now);
    if (cc == SOCKET_ERROR) {
        printf("Error: %d.\n", GetLastError());
    }
    else if (cc == 0) {
        printf("Server closed!", buf);
    }
    else if (cc > 0) {
        buf[cc] = '\0';
        printf("收到消息：%s\n", buf);
        printf("收到时间：%s\n\n", pts);
        int dd = send(ssock, pts, strlen(pts), 0);
        dd = send(ssock, buf, cc, 0);
    }
    (void)closesocket(ssock);
}
```

2、 在(2)编写 TCP Echo 增强程序时，为了实现 IP 地址的传送遇到了问题。开始一直希望能够直接在 client 中读取本机的 IP 地址和端口号，但是进行了许多尝试以及查阅了许多资料，都没有能够实现，最终还是采用将服务端获取得到的服务端的 IP 地址和端口号再传回用户端的方法。采用的方式是分条传输，因为 TCP 传输是稳定的，传输顺序不会发生错乱，所以可以保证程序的正确性。或许还是有办法不通过传输直接在客户端获取 IP 地址和端口号的方法，但我并没能想到，这还要靠以后一点一点学习。

3、 在(3)编写 UDP Echo 增强程序时，为了实现 IP 地址端口号还有时间信息的回传时，遇到了困难。开始希望采用和 TCP 一样的方式回传，但是由于 UDP 是不稳定的传输，所以不能保证回传的顺序正确，可能会发生信息的错位。所以最终采用的是将需要回传的数据保存为一个格式固定好的字符串，统一回传来解决传输的问题。但是即使如此还是遇到了问题，经过查阅发现是因为把所有回传的信息保存为一个字符串导致缓冲区空间不足，所以将缓冲区的空间由 2000 改为 5000，程序可以正常运行了。

思考：

在实验过程中，更多的是对 TCP 和 UDP 传输的过程的理解，在程序中最值观的看法是，TCP 需要两个 SOCKET,一个用来监听，一个用来建立连接；而 UDP 只需要一个 SOCKET.

从实验过程中我们可以看出来，UDP 相较于 TCP 的传输过程是非常简单的，UDP 沟通简单，不需要大量的数据结构。并且 UDP 不会建立连接，但会监听这个地方，设都可以给他传送数据，它也可以给任何人传送数据，甚至可以给多个人同时传送数据。但是 UDP 不会根据网络的情况进行拥塞控制，无论是否丢包，它该怎么发还是怎么发。一般 UDP 适用于需要的资源少，网络情况稳定的内网，或者对于丢包不敏感的应用。

而 TCP 相应的就会严谨很多，能够实现可靠的数据传输。TCP 服务器进程打开，准备接受客户进程的连接请求，此时服务器进入了监听状态。客户端向服务器发出连接请求，同时选择一个初始序列号，此时，TCP 客户端进程进入了同步已发送状态。TCP 客户进程收到确认后，还要向服务器给出确认后，还要向服务器给出确认。此时 TCP 连接建立，客户端进入已建立连接状态。当服务器收到客户端的确认后也进入已建立连接状态，双方可以开始通信。

实验体会：

本次实验是第一次进行计算机网络方面的实验，从最开始的懵懵懂懂到现在的初入门路，这其中有曲折也有乐趣。发现在进行计算机网络方面的实验时，接触到了很多之前完全没有见到过的结构体，类以及函数。这个是很有趣的一件事，一般来说虽然还没有学习过，但是看 C 语言或是 C++程序都能明白个大概，但是计算机网络部分的程序在没有接触之前完全看不懂，完全像是一门新的语言。所以以后要走的路还很长。另外，这次的三个小实验都是采用 C 语言来实现的，这也是一个小小的疑惑，是 C 语言对计算机网络的实验更加友好吗？还是有什么其他的原因？这以后还要一点一点摸索。

【交实验报告】

每位同学单独完成本实验内容并填写实验报告。

交作业地点：http://172.18.187.251/netdisk/default.aspx?vm=18net

编程实验

截止日期： 开学第二周的周五 23:00

上传文件： 学号_姓名_Echo 实验报告.doc

学号_姓名_Echo 实验源码.rar （源程序和可执行程序）