一、Windows篇

1.1 Socket基础

1.1.1 原理与概念

提供两种服务方式：面向连接、无连接

面向连接：每次数据传输都要进行建立连接、使用连接、关闭连接过程，传输数据过程中数据包**不需要指定目的IP和port。**TCP

无连接：每次数据传输不需要建立连接，但每个数据包**必须包含目的IP和port。**UDP

Socket分为三类：流式套接字、数据报式套接字、原始套接字。

流式套接字：面向连接、可靠数据传输；TCP

数据报式套接字：无连接，可能丢包，没有顺序；UDP

原始套接字：包含IP头部信息，可以发送和接收IP层数据包；ICMP

1.1.2 WinSock

WinSock包含两个版本：WinSock1和WinSock2

WinSock1.1:

```
#include <winsock.h>
#pragma comment (lib, "wsock32.lib")
```

WinSock2.2:

```
#include <winsock2.h>
#pragma comment (lib, "ws2_32.lib")
```

WSAStartup()

用于初始化Windows Sockets，只有在调用了WSAStartup()函数后才能使用其他Windows Sockets API。

使用：

```
WSADATA wsaData;
if(WSAStartup(MAKEWORD(2,2),&wsaData) != 0){
    printf("init error\n");
}
```

IP地址转换

网络字节顺序：

网络字节顺序采用大端方案（低地址存高字节）；

inet_addr():将点分IP地址转换为二进制表示的且网络字节顺序的IP地址；

inet_ntoa():将二进制表示的且网络字节顺序的IP地址转换为点分IP地址；

主机字节顺序:

掌握四个转换函数

htonl()

htons()

ntohl()

ntohs()

n:network h:host l:long(IP) s:short(port)

1.1.3 面向连接的Socket

注：函数调用失败几乎都返回-1

socket()

创建套接字

```
WSADATA wsaData;
if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
{
    MessageBox("Failed to load Winsock");
}
if ((m_listenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) ==
INVALID_SOCKET)
{
    MessageBox("Failed to Create SOCKET");
}
```

bind()

将本地地址与一个Socket绑定在一起，应用与未连接的Socket，在connect()、listen()之前调用，TCP，UDP均可。

```
if (bind(m_listenSocket, (LPSOCKADDR)&addrServ, sizeof(addrServ)) ==
SOCKET_ERROR)
{
    closesocket(m_listenSocket);
    MessageBox("Failed to Bind SOCKET");
}
```

listen()

将套接字设置为监听状态

```
if (listen(m_listenSocket, 5) == SOCKET_ERROR)//5:指定等待连接队列的最大长度
{
    MessageBox("Failed to Listen SOCKET");
}
```

accept()

在listen()之后，用来等待接受连接请求。

```
m_toClientSocket = accept(m_listenSocket, &addrOfClient, &addrofclientlen);
```

recv()

从已连接的Socket中接收数据。

```
length = recv(m_toClientSocket, (char*)buffer, BUFFER_SIZE, 0);
if (length == SOCKET_ERROR)
{
    MessageBox("recv failed !");
}
```

connect()

用于建立连接，对方Socket必须处于监听状态。

```
retVal = connect(m_toServerSocket, (LPSOCKADDR)&servAddr, sizeof(servAddr));
if (SOCKET_ERROR == retVal)
{
    closesocket(m_toServerSocket);
}
```

send()

从已连接的Socket上发送数据。

```
int retVal = ::send(m_toClientSocket, tbuf, sizeof(tbuf), 0);
if (SOCKET_ERROR == retVal)
{
    MessageBox("send failed !");
    closesocket(m_toClientSocket[i]);
}
```

closesocket()

关闭一个Socket，并释放其所占用的所有资源。

shutdown()

禁止指定Socket收发数据，但不会关闭Socket，也不会释放资源。

sockaddr_in

```
sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("192.168.0.1");
addr.sin_port = htons(9990);
```

1.1.4 面向非连接的Socket

```
WSADATA wsaData;
WSAStartup(MAKEWORD(2,2),&wsaData);
socket = socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
```

sendto()

```
ret = sendto(socket,sendBuf,sizeof(sendBuf),0,(struct
sockaddr_in)addr,sizeof(addr));
```

recvform()

```
ret = recvfrom(socket,recvBuf,sizeof(recvBuf),0,(struct
sockaddr_in)addr,sizeof(addr));
```

1.1.5 Socket选项

getsockopt()

获取socket属性

Tip:socket级别分为SOL_SOCKET和IPPROTO_TCP两个级别。

setsockopt()

当级别被设置为SOL_SOCKET时，可以通过setsockopt()设置socket属性

1.2 探测网络设备

获取信息的一般编程方法：

1.定义获取到的网络信息的结构体指针和用于保存获取到的网络信息的长度变量；

2.为指针分配空间（也可以直接等于0）；

3.调用获取网络信息的函数并将两个参数传进去，函数返回数据大小并存入第二个参数（长度变量）中；

4.重新为结构体指针分配返回数据大小的内存空间；

5.再次调用获取网络信息的函数，获取到全部网络信息，获取的信息在传入的结构体指针内；

6.通过结构体指针打印输出相应信息。

GetAdapterInfo() 获取适配器信息

GetNetworkParams() 获取主机名、域名和DNS

GetNumberOfInterfaces() 获取网络接口信息

GetIpAddrTable() 获取IP地址表

AddIPAddress() 添加IP地址；向指定适配器添加IP地址

DeleteIPAddress() 删除IP地址

1.3 select编程模式

1.定义变量，定义需要select监听的文件描述符，定义两个fd_set集合并初始化（FD_ZERO）；

2.初始化套接字，如果客户端程序就创建套接字、初始化服务器IP、调用connect连接；如果是服务器就创建套接字、bind绑定IP、listen监听；

3.向集合中添加需要监听的文件描述符（套接字）并拷贝集合，FD_SET();

客户端：标准输入和socket（键盘输入和接收服务器返回）；

服务器：监听socket和与客户端连接的socket（连接和收发数据）；

4.while循环执行select()函数，当前进程被阻塞，如果有fd被置位且返回值不等于-1，使用FD_ISSET()判断各个FD是否在返回集合中，以此确定哪个FD被置位，再执行对应操作；

5.最后重新拷贝集合，使传给select的集合始终为全部需要监听的fd。

1.4 Windows多线程编程

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  SIZE_T dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpParameter,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);

lpThreadAttributes：描述施行于这一新线程的security属性。NULL表示使用缺省值。
```

dwStatckSize ： 新线程拥有自几的堆栈，0表示使用缺省大小。
lpStartAddress： 新线程将开始的起始地址，这是一个函数指针（在C语言中函数名成即代表函数指针）。
lpParameter：此值将被传送到上述所指定的新线程函数去，作为参数（唯一的参数）。
dwCreationFlags：允许你产生一个暂时挂起的线程，默认情况是“立即开始执行”。
lpThreadId ：新线程的ID会被传回到这里。

基于多线程的客户端程序：

```
#include "pch.h"
#include "framework.h"
#include "MulThreadChatConClient.h"
#include <WINSOCK2.H>
#include "string.h"
#include "stdio.h"
#pragma comment(lib,"WS2_32.lib")
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
#pragma warning(disable : 4996)
#define BUF_SIZE    64
CWinApp theApp;
using namespace std;

SOCKET sHost;
DWORD  WINAPI MTClientRecvThread(LPVOID pParam)
{
    int ReadLen;
    char buf[BUF_SIZE];
    SOCKET sHost = (SOCKET)pParam;  //参数类型转换（必考），多半是传socket参数进来
    int   retVal;
    while (1)
    {
        ZeroMemory(buf, BUF_SIZE);
        if ((retVal = recv(sHost, buf, sizeof(buf), 0)) == SOCKET_ERROR)
        {
            int retError = WSAGetLastError();
            if (retError == 10054)
            {
                printf("错误：WSARecv() failed with error %d,服务器断开连接\n",
WSAGetLastError());
                break;
            }
            if (retError != ERROR_IO_PENDING)
            {
                printf("错误：WSARecv() failed with error %d\n",
WSAGetLastError());
                break;
            }
        }
        printf("Recv From Server: %s\n", buf);
        if (strcmp(buf, "quit") == 0)
        {
            printf("quit!\n");
            break;
        }
    }
    return 0;
```

```
}

int main()
{
    WSADATA     wsd;         // 用于初始化Windows Socket
    SOCKET      sHost;  // 与服务器进行通信的套接字
    SOCKADDR_IN servAddr;   // 服务器地址
    int         retVal; // 调用各种Socket函数的返回值
    if (WSAStartup(MAKEWORD(2, 2), &wsd) != 0)
    {
        printf("WSAStartup failed !\n");
        return 1;
    }
    sHost = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (INVALID_SOCKET == sHost)
    {
        printf("socket failed !, errorno\n", WSAGetLastError());
        WSACleanup();
        return -1;
    }
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.S_un.S_addr = inet_addr("192.168.124.2");
    servAddr.sin_port = htons(9990);
    int sServerAddlen = sizeof(servAddr);
    retVal = connect(sHost, (LPSOCKADDR)&servAddr, sizeof(servAddr));
    if (SOCKET_ERROR == retVal)
    {
        printf("connect failed !, errorno\n", WSAGetLastError());
        closesocket(sHost);
        WSACleanup();
        return -1;
    }
    DWORD dwThreadId;   //需要准备一个变量接收线程ID
    if (CreateThread(NULL, 0, MTClientRecvThread, (LPVOID)sHost, 0, &dwThreadId)
== NULL)//创建线程函数（必考）
    {
        printf("错误：CreateThread failed with error %d\n", GetLastError());
        return -1;
    }
    while (true)
    {
        char        buf[BUF_SIZE];
        ZeroMemory(buf, BUF_SIZE);
        printf("Please input a string to send: ");
        gets_s(buf,256);
        retVal = send(sHost, buf, strlen(buf), 0);
        if (SOCKET_ERROR == retVal)
        {
            printf("send failed !, errorno:%d\n", WSAGetLastError());
            closesocket(sHost);
            WSACleanup();
            return -1;
        }
        // 如果收到quit，则退出
        if (strcmp(buf, "quit") == 0)
        {
            printf("quit!\n");
            break;
```

```
        }
    }
    closesocket(sHost);
    WSACleanup();
    // 暂停，按任意键继续
    system("pause");
    return 0;
    return nRetCode;
}
```

1.5 基于WSAAsyncSelect的MFC窗口编程

WSAAsyncSelect()会自动将socket设置为非阻塞模式

自定义消息：

```
#define WM_MYMESSAGE WM_APP + 10        //名字自定义，值只需要WM_APP+任意一个数即可
```

消息映射：

```
BEGIN_MESSAGE_MAP(CSimpleChatClientDlg, CDialogEx)
    ON_MESSAGE(WM_MYMESSAGE, MyFunction)    //消息映射，宏里指定消息和函数即可
END_MESSAGE_MAP()
```

消息函数：

```
//hWnd:窗口句柄
//uMsg:当网络事件发生时的消息
//wParam: 表明发生网络事件的套接字。
//lParam:低字节表明已发生的网络事件。高字节包含错误代码。

LRESULT CALLBACK MyFunction(HWND hwnd,UINT uMsg,WPARAM wParam, LPARAM lParam)
{
    SOCKET socket = (SOCKET)wParam;
    int error = HIWORD(lParam);
    long event  = LOWORD(lParam);
    if(uMsg == WM_MYMESSAGE){   //自定义消息
        if(socket == listen_socket){
            if (event & FD_ACCEPT && error == 0)
            {
                //...新建套接字与客户端连接
            }
        }
        if(socket == toClientSocket){
            if(event & FD_READ && error == 0){
                //...处理客户端发送过来的请求
            }
        }
    }
}
```

WSAAsyncSelect:

```
int WSAAsyncSelect(
    SOCKET s,        //需要通知的套接字
    HWND hWnd,       //当网络事件发生时接收消息的窗口句柄
    u_int wMsg,      //当网络事件发生时窗口收到的消息,在此消息的响应函数内对网络事件进行处理
    long lEvent      //应用程序感兴趣的网络事件集合
);

//使用
if ((m_listenSocket = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET){
    MessageBox("Failed to Create SOCKET", "error message");
    return FALSE;
}
WSAAsyncSelect(m_listenSocket, m_hWnd, WM_MYMESSAGE, FD_ACCEPT | FD_CLOSE);
//bind、listen...
```

1.6 非阻塞模式

设置为非阻塞:

```
int iMode = 1;
ioctlsocket(socket,FIONBIO,&iMode);
```

编程实例:

```
main(){
    //...
    while(1){
        ret = recv(socket,buf,BUFSIZE,0);
        if(ret == -1){
            if(WSAGetLastError() == WSAEWOULDBLOCK){//接收缓冲区无数据
                sleep(1000);
                continue;
            }else{
                printf("recv error\n");
                break;
            }
        }
        //...
        //处理接收的数据
    }
}
```

二、Linux篇

2.1 gethostbyname()

```
struct hostent *gethostbyname(const char *name);
//传入值是域名或者主机名，返回一个hostent的结构。如果函数调用失败，将返回NULL。

const char *inet_ntop(int af, const void *src, char *dst, socklen_t cnt);
//这个函数，是将类型为af的网络地址结构src，转换成主机序的字符串形式，存放在长度为cnt的字符串中。返回指向dst的一个指针。如果函数调用错误，返回值是NULL。

struct hostent
{
    char    *h_name;
    char    **h_aliases;
    int     h_addrtype;
    int     h_length;
    char    **h_addr_list;
    #define h_addr h_addr_list[0]
};

//使用
struct hostent *hptr;
char str[32];
if((hptr = gethostbyname("xxx")) == NULL){
    printf("gethostbyname error");
    return 0;
}
printf("%s",inet_ntop(hptr->h_addrtype,hptr->h_addr,str,sizeof(str)));  //打印IP地址
```

2.2 多进程&信号处理

P64

多进程处理

信号安装、子进程回收

2.3 ping

P134

IP&ICMP结构体定义

2.4 traceroute

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
```

```c
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/udp.h>

#define BUFSIZE     1500
#define ICMP_HLEN   8
#define UDP_HLEN    8


uint16_t    sport, dport;           //源端端口和目的端端口
struct sockaddr_in  destaddr;       //目的端套接字地址结构
char *hostname;                     //目的端主机名
char sendbuf[BUFSIZE], recvbuf[BUFSIZE];    //发送缓冲区和接收缓冲区
int datalen;

void trace_icmp(void);
uint16_t in_cksum(uint16_t *addr, int len);
char * sock_ntop_host(const struct sockaddr *addr, socklen_t addrlen);
int sock_addr_cmp(const struct sockaddr *sa1, const struct sockaddr *sa2,
socklen_t salen);
uint16_t dport = 32768 + 666;

int main(int argc, char *argv[])
{
    char c;
    struct hostent *hostent;
    in_addr_t saddr;
    hostname = argv[1];
    // 处理用户输入
    if ((saddr = inet_addr(hostname)) == INADDR_NONE) {
        if ((hostent = gethostbyname(hostname)) == NULL) {
            printf("unknow host %s \n", hostname);
            exit(1);
        }
        memcpy(&saddr, hostent->h_addr, hostent->h_length );
    }
    destaddr.sin_family = AF_INET;
    destaddr.sin_addr.s_addr = saddr;

    trace_icmp();

    return 0;
}


uint16_t in_cksum(uint16_t *addr, int len)
{
        int nleft = len;
        uint32_t sum = 0;
        uint16_t *w = addr;
        uint16_t answer = 0;

        while (nleft > 1) {
                sum += *w++;
                nleft -= 2;
        }
        if (nleft == 1) {
                *(unsigned char *) (&answer) = *(unsigned char *) w;
```

```
                sum += answer;
        }
        sum = (sum >> 16) + (sum & 0xffff);
        sum += (sum >> 16);
        answer = ~sum;
        return answer;
}


void trace_icmp(void)
{
    int sockfd;
    struct sockaddr addr;
    socklen_t addrlen;
    double rtt;
    int icmpdatalen, seq, ttl, query, code, ndone;
    struct timeval tvsend, tvrecv;
    struct icmp *icmp;
    size_t len;
    static char str[64];
    struct ip   *ip1, *ip2;
    //struct icmp     *icmp, *icmp2;
    int iphlen1, iphlen2, icmplen, ret, n;
    struct sigaction act;

    //建立一个原始套接子发送/接受ICMP包
    if ((sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)) < 0) {
        printf("scoket error\n");
        exit(1);
    }
    setuid(getuid());

   /*
    //建立UDP socket发送UDP包
    if((send_sockfd = socket(AF_INET,SOCK_DGRAM,0)) < 0){
        printf("udp sock error/n");
        exit(1);
    }
*/
    printf("traceroute to %s (%s), 30 hops max (ICMP) \n",
        hostname,
        inet_ntoa(destaddr.sin_addr));

    icmpdatalen = 56;
    seq = 0;
    ndone = 0;
    for (ttl=1; ttl<=30 && ndone==0; ttl++) {
        setsockopt(sockfd, IPPROTO_IP, IP_TTL, &ttl, sizeof(int));//设置ttl
        printf("%3d", ttl);
        fflush(stdout);
        for (query=0; query<3; query++) {//默认请求3次
            ++seq;
            gettimeofday(&tvsend, NULL);
            // 构建ICMP回显请求消息
            icmp = (struct icmp *) sendbuf;
            icmp->icmp_type = ICMP_ECHO;//基于ICMP的是利用回显
            icmp->icmp_code = 0;
            icmp->icmp_id = htons(getpid());
```

```
            icmp->icmp_seq = htons(seq);
            memset(icmp->icmp_data, 0xa5, icmpdatalen);
            memcpy(icmp->icmp_data, &tvsend, sizeof(struct timeval));

            len = ICMP_HLEN + icmpdatalen;
            icmp->icmp_cksum = 0;   //要先置0
            icmp->icmp_cksum = in_cksum((u_short *) icmp, len);

            if (sendto(sockfd, sendbuf, len, 0, (struct sockaddr *) &destaddr,
sizeof(destaddr)) < 0) {
                printf("sendto error\n");
                exit(1);
            }
            // 处理回应结果
            while(1) {
                struct timeval tv = {3,0};
                setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char*)&tv,
sizeof(struct timeval));
                n = recvfrom(sockfd, recvbuf, sizeof(recvbuf), 0, (struct
sockaddr *)&addr, &addrlen);
                if (n < 0) {
                    if(errno == 11){
                        printf("\t*");
                        break;
                    }
                    if(errno == EINTR)  //处理慢系统调用
                        continue;
                    printf("recv error: %s\n",strerror(errno));
                    exit(1);
                }
                // 处理接收到的数据
                ip1 = (struct ip *) recvbuf;
                iphlen1 = ip1->ip_hl << 2;//IP头长度
                icmp = (struct icmp *) (recvbuf + iphlen1);//获取到ICMP字段
                if (icmp->icmp_type == 11       //ttl==0时返回的ICMP
type=11,code=0
                    && icmp->icmp_code == 0) {
                        gettimeofday(&tvrecv, NULL);    //需要取timeval结构体地址，
否则会出现段错误
                        //计算rtt
                        if ((tvrecv.tv_usec -= tvsend.tv_usec) < 0) {//借位
                                    tvrecv.tv_sec--;
                                    tvrecv.tv_usec += 1000000;
                        }
                        tvrecv.tv_sec -= tvsend.tv_sec;
                        rtt = tvrecv.tv_sec * 1000.0 + tvrecv.tv_usec / 1000.0;
                        printf("\t%.3f ms", rtt);
                        break;
                    }
                }else if (icmp->icmp_type == 0) {//回显返回，即找到目的主机
type=0
                        inet_ntop(AF_INET, ((struct sockaddr_in *)&addr)-
>sin_addr,str, sizeof(str));
                        printf("\t%s",str);
                        ++ndone;
                        break;
                }
```

```
                /*else if (icmp->icmp_type == 3 && icmp->icmp_code == 3) {//目的主机端口不可达，即找到目的主机           type=3,code=3
                        inet_ntop(AF_INET, ((struct sockaddr_in *)&addr)->sin_addr,str, sizeof(str));
                        printf("\t%s",str);
                        ++ndone;
                        break;
                }*/
            }
        }
        printf("\n");
    }
}
```

2.5 校验和函数

```
u16 checksum(u8 *buf,int len)
{
    u32 sum=0;
    u16 *cbuf;
    cbuf=(u16 *)buf;
    while(len>1)
    {
        sum+=*cbuf++;
        len-=2;
    }
    if(len)
        sum+=*(u8 *)cbuf;
    sum=(sum>>16)+(sum & 0xffff);
    sum+=(sum>>16);

    return ~sum;
}
```