计算机网络实验报告

警示

1. 实验报告如有雷同，雷同各方当次实验成绩均以 0 分计。
2. 当次小组成员成绩只计学号、姓名登录在下表中的。
3. 在规定时间内未上交实验报告的，不得以其他方式补交，当次成绩按 0 分计。
4. 实验报告文件以 PDF 格式提交。

院系	数据科学与计算机学院	班 级	15-1 班	组长	李佳
学号	15331151	15331150	15331143		
学生	李佳	李辉旭	黎皓斌		

编程实验

【实验内容】

（1）完成实验教程实例 3-2 的实验（考虑局域网、互联网两种实验环境），回答实验提出的问题及实验思考。（P103）。

（2）注意实验时简述设计思路。

（3）引起 UDP 丢包的可能原因是什么？

【实验名称】

基于 UDP 丢包统计程序设计

【实验目的】

选择一个操作系统环境（Linux 或者 Windows），编制 UDP/IP 通信程序，完成一定的通信功能。

【实验要求】

在发送 UDP 数据包时做一个循环，连续发送 100 个数据包；在接收端统计丢失的数据包。

实验时，请运行 Wireshark 软件，对通信的数据包进行跟踪分析。

【实验设计】

UDP（User Data Protocol，用户数据报协议）是与 TCP 相对应的协议。它是面向非连接的协议，就是在正式通信前不必与对方先建立连接，不管对方状态就直接发送。

首先了解双方的通信流程大致如下。

服务器端：

1、打开通信通道并告知本地主机，服务器端可在某一地址和端口接收客户端请求。

2、等待客户请求到达该端口。

3、接收到重复的服务请求，处理并发送应答信号。如果是并发的请求，那么就启动一个新进程。服务完成后，关闭此进程与客户的通信链路，并终止。

4、返回第二步，也就是继续等待新的请求。

5、关闭服务器。

客户端：

1、打开一个通信通道，并连接到服务器所在主机的特定端口。

2、向服务器端发送请求报文，等待并接受应答，然后可继续提出请求 。

3、通信结束，关闭通信通道并终止。

根据通信流程对于服务器端大概的过程设计有下面几个关键：

1、创建套接字，socket 函数，返回一个套接字的描述符,三个参数，一个参数 af 指定地址族，第二个参数指定套接字的类型也就是 UDP，第三个参数是与特定的地址家族相关的协议，如果指定为 0，那么他就根据地址格式和套接字的类别，自动设置一个合适的协议。如果该函数调用成功，他将返回一个新的 SOCKET 数据类型的套接字描述符。如果失败则返回一个 INVALID_SOCKET 错误信息通过 WSAGetLastID_SOCKET 函数返回。

2、绑定套接字函数 bind，接收三个参数，第一个指定要绑定的套接字，第二个参数指定该套接字的本地地址信息，是一个指向 sockaddr 结构的指针变量，由于地址结构是为所有地址家族准备的，这个结构可能随所拥有网络协议不同而不同。所以要用第三个参数指定该地址结构的长度，显然要事先定义 sockaddr 结构体。另外，因为实际要求的是内存区，所以对于不同的协议家族，用不同的结构来替换。常用的两个函数 inet_addr(),将点十进制的 IP 地址转换为适合分配给 s_addr 的 u_long 类型的数值。inet_ntoa()函数起相反的作用。

3、用返回的套接字和客户进行通信（sendto/recvfrom 函数）

4、返回，等待一个新的客户请求

5、关闭套接字（closesocket 函数）

客户端比较简洁的概括就是：

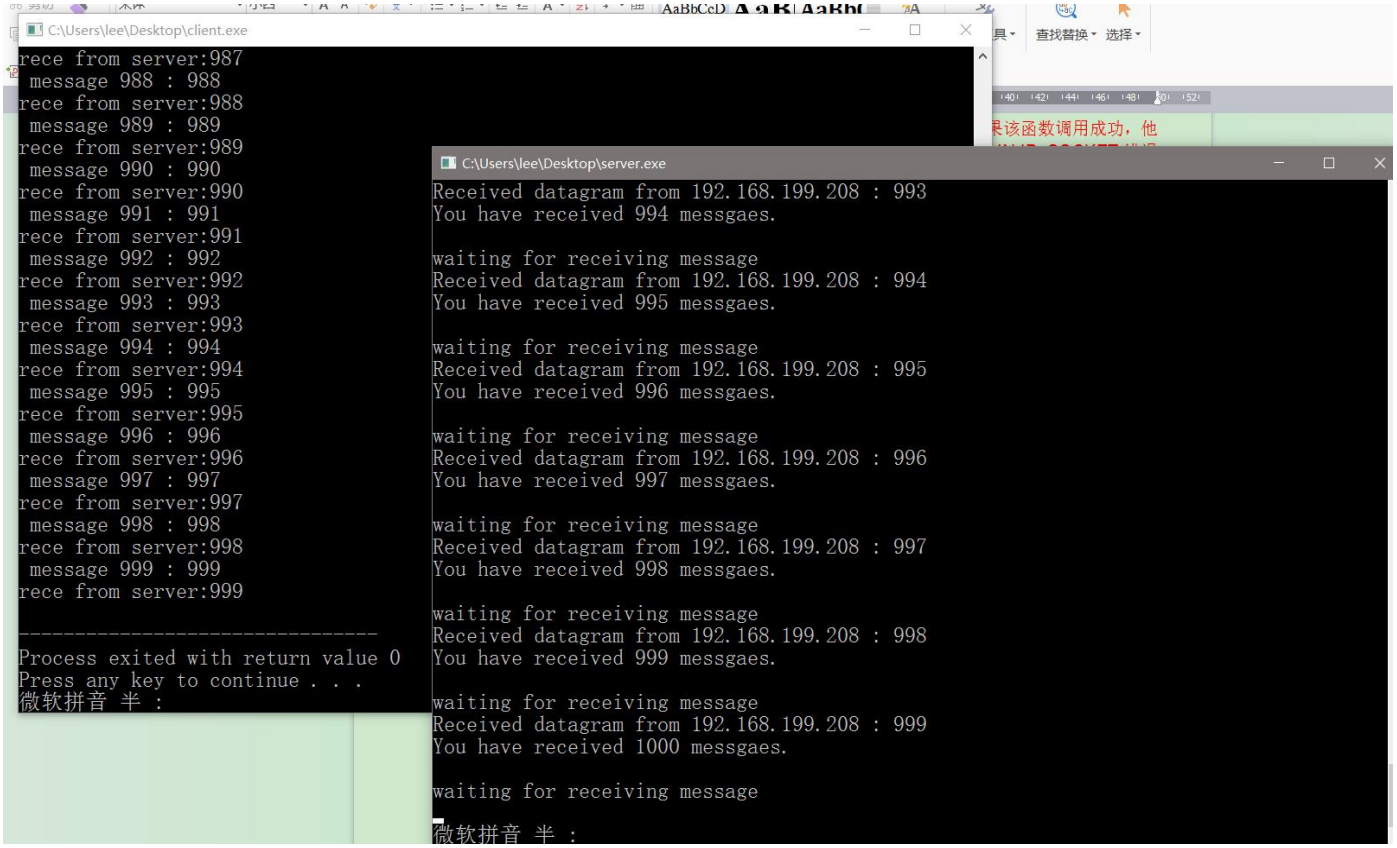
1、创建套接字

2、 和服务器进行通信（sendto/recvfrom 函数）

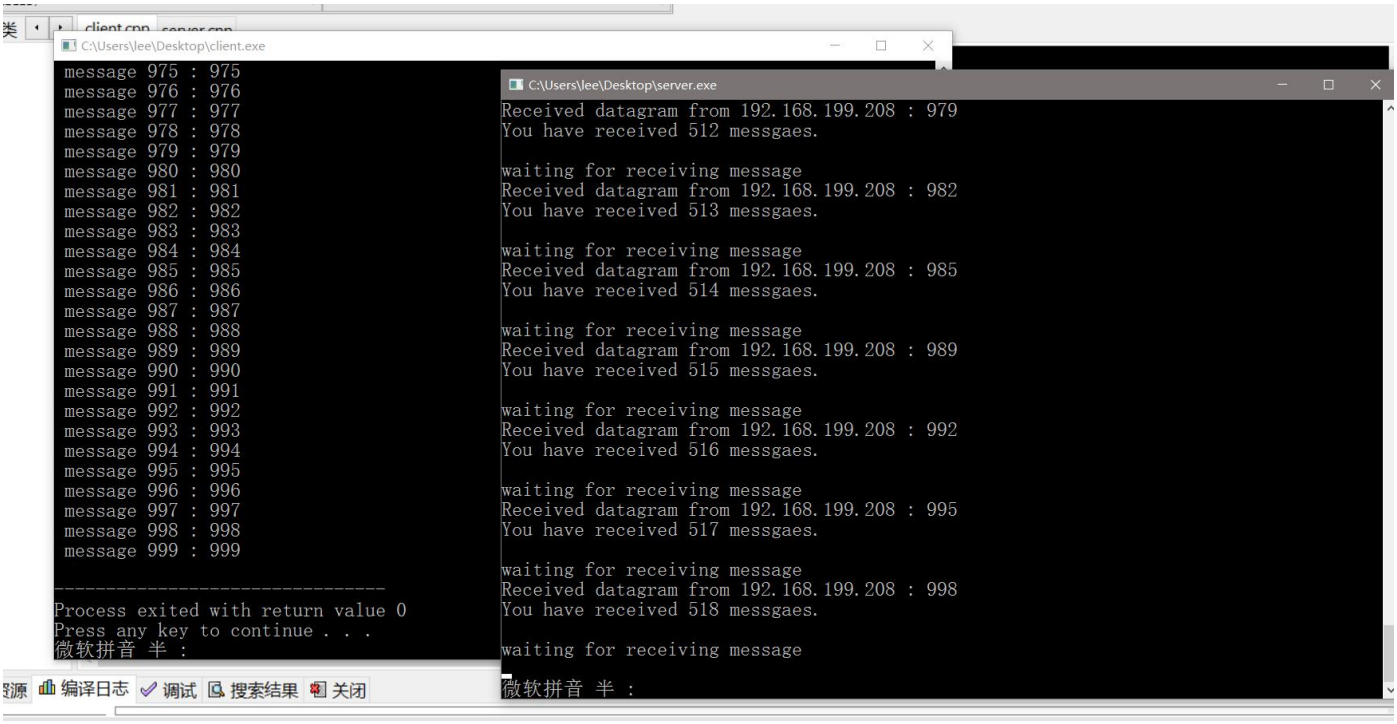
3、 关闭套接字

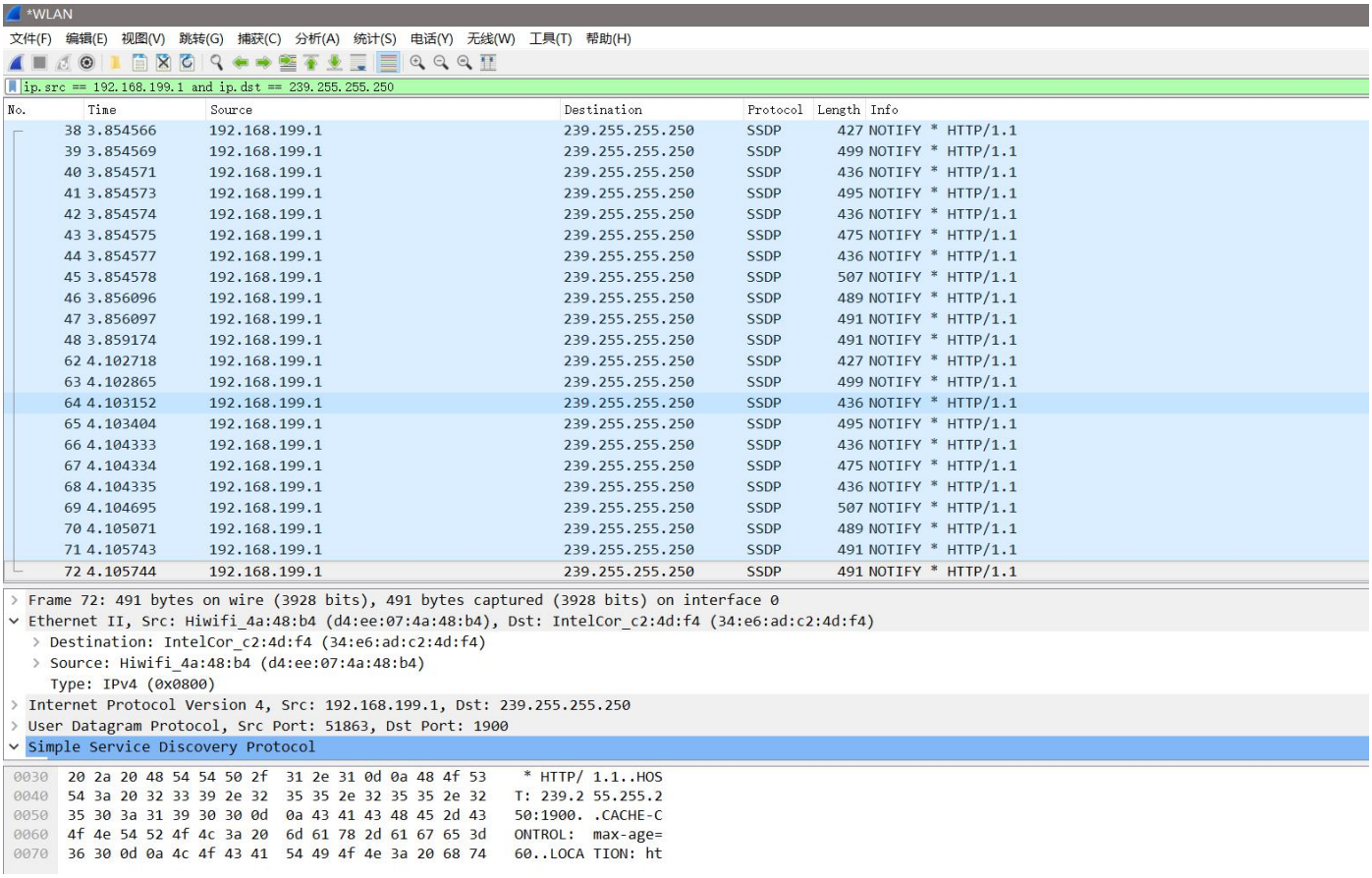
实验结果展示：

1.添加函数使得不丢包，1000 数据包成功发送并全部接收。



2.丢包情况，1000 数据包成功发送并接收 518 个包。

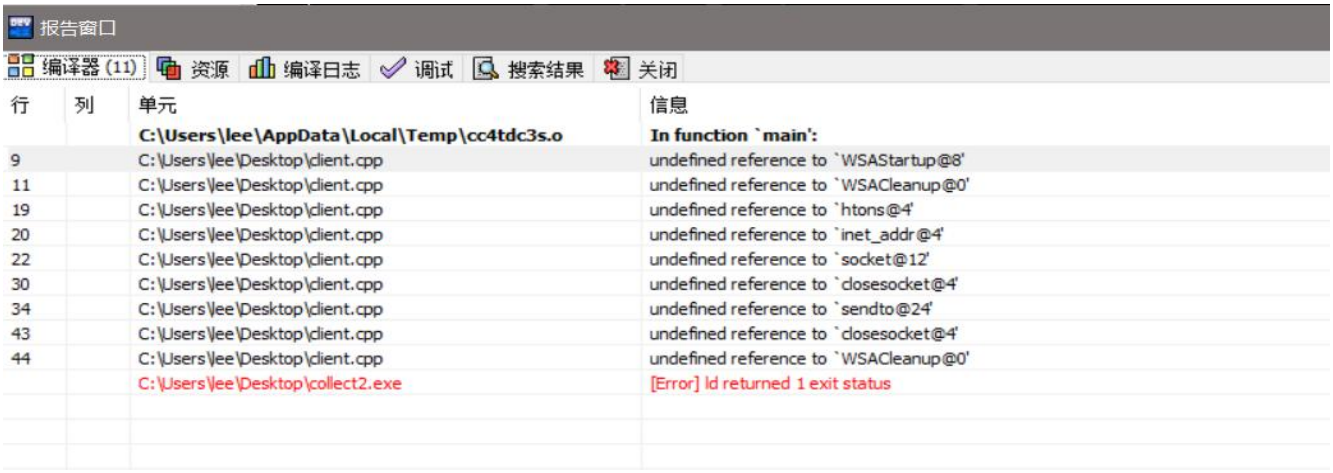




【实验思考】

（1）说明在实验过程中遇到的问题和解决方法。

①在 dev C++上编码完成后无法通过编译，报错如下：



经上网搜索后得知需要在编译链接器中添加库文件“-lws2_32”后编译成功；

②对于 htons 函数不理解。经查找资料后知道，在把端口号传给 sockaddr_in 时是需要一步转化的，调用 htons 函数将整型变量从主机字节顺序转变成网络字节顺序， 就是整数在地址空间存储方式变为高位字节存放在内存的低地址处。网络字节顺序是 TCP/IP 中规定好的一种数据表示格式，它与具体的 CPU 类型、操作系统等无关，从而可以保证数据在不同主机之间传输时能够被正确解释，网络字节顺序采用 big-endian 排序方式。

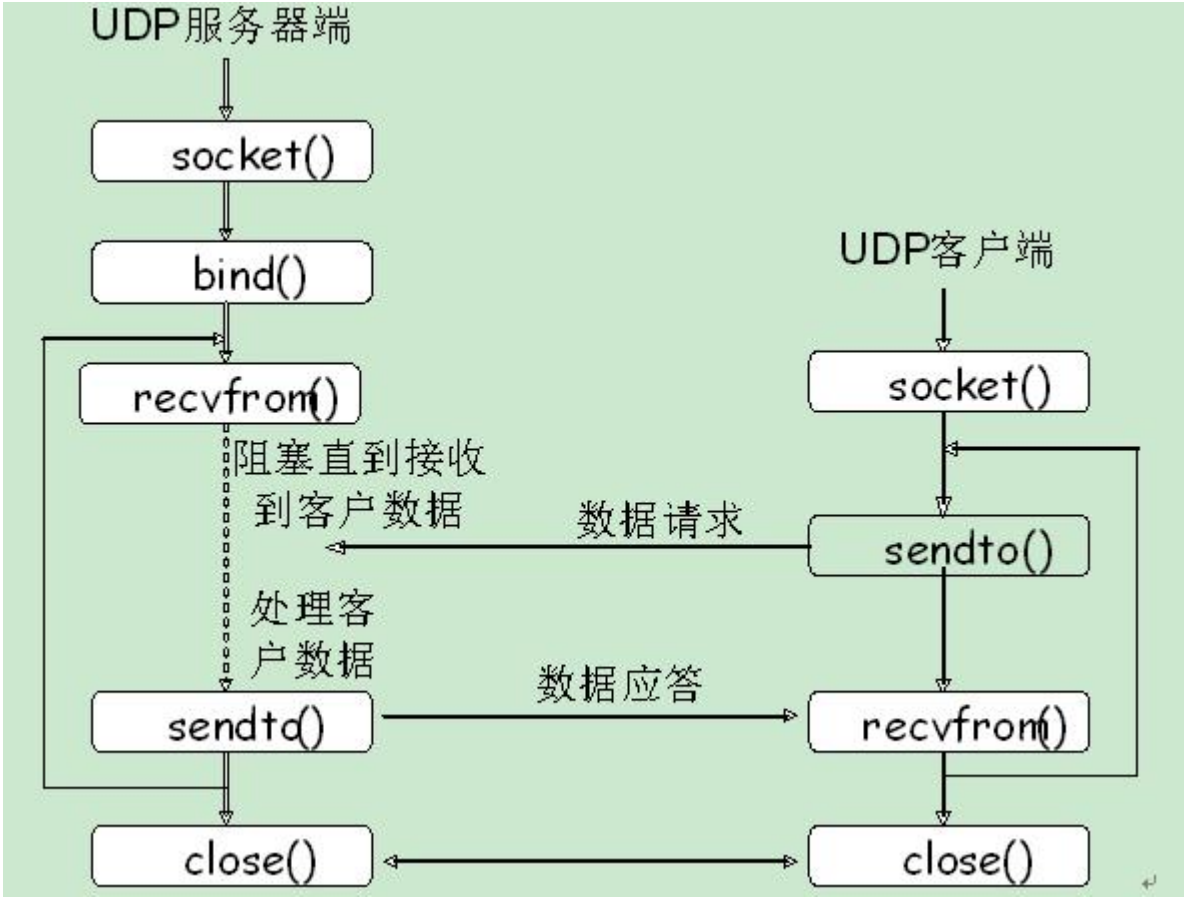
③在根据流程编码后实验数次，发现并不会丢包，我们把循环次数放大，把传递的字符串扩大都不会丢包。经过讨论和测试发现，我们的代码中

```
Sleep(10);
if (recvfrom(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&server, &len) != SOCKET_ERROR)
printf("rece from server:%s\n",buffer);
```

根据我们的理解，这一部分会导致不丢包。首先，sleep()会使得发送缓冲区不容易满，难以造成数据包的丢失。其次，recvfrom 会阻塞后续数据包的传输，直到服务器传回确认收到的信息。

（2）给出程序详细的流程图和对程序关键函数的详细说明。

程序详细流程图，由于我们流程和此图一致，所以直接粘贴了一个教程中的流程图。



程序的关键函数如下：

```
if (WSAStartup(MAKEWORD(2,2),&wsaData)) {       //调用Windows Sockets DLL
        printf("init error\n");
        WSACleanup();
        return 0;
}
```

客户端和服务器端的此函数是调用启动 Windows Sockets DLL 并检测是否启动成功，如果不成功输出错误信息并结束进程。

```
struct sockaddr_in server;
int len =sizeof(server);        //服务器地址长度

server.sin_family = AF_INET;          //AF_INET地址族
server.sin_port = htons(22222);                         ///server的监听端口
server.sin_addr.s_addr = inet_addr("192.168.199.208");          ///server的地址

socket1 = socket(AF_INET, SOCK_DGRAM, 0);      //创建套接字
```

这是客户端声明 sockaddr_in 结构体，以及给其中的参数赋值的代码，我们使用的是本机 IP 地址（192.168.199.208），如果使用局域网实验，到 localhost 将此处改为 172.0.0.1 即可。最后一句代码是创建套接字，参数已经在设计过程中有叙述。

计算机网络实验报告

```
for(int i = 0; i < 100; i++) {
    itoa(i, buffer, 10);
    printf(" message %d : %s\n", i, buffer);
    if (sendto(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&server, len) != SOCKET_ERROR) {      //参数中sockaddr_in转化sockaddr
        //Sleep(10);
        //if (recvfrom(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&server, &len) != SOCKET_ERROR)
        //  printf("rece from server:%s\n",buffer);
    }
    else {
        printf("send failed!\n");
    }
}
```

客户端循环发送数据包的代码，我们使用的数据是当前的发送次数。sendto 函数：UDP 使用 sendto()函数发送数据，他类似于标准的 write()，但是在 sendto()函数中要指明目的地址。int FAR sendto (IN SOCKET s, IN const char FAR * buf, IN int len, IN int flags, IN const struct sockaddr FAR *to, IN int tolen);

s 套接字

buf 待发送数据的缓冲区

len 缓冲区长度

flags 调用方式标志位，一般为 0，改变 Flags，将会改变 Sendto 发送的形式

addr (可选)指针，指向目的套接字的地址

tolen 所指地址的长度

在注释掉的地方是之前没有丢包的传输代码，会缓冲并等待返回数据。

服务端函数：

```
int count = 0;
SOCKET socket1;
WSADATA wsaData;                    //Socket的版本信息
if (WSAStartup(MAKEWORD(2,2),&wsaData)) {           //调用Windows Sockets DLL
        printf("init error\n");
        WSACleanup();
        return 0;
}

printf("server activited\n");

struct sockaddr_in local;
struct sockaddr_in from;
int fromlen =sizeof(from);                      //客户端地址长度

local.sin_family=AF_INET;
local.sin_port=htons(22222);                    //监听端口
local.sin_addr.s_addr=INADDR_ANY;               //本机

socket1=socket(AF_INET,SOCK_DGRAM,0);       //创建套接字
```

此部分与客户端声明大致一样：先声明了一个 sockaddr_in 和一个套接字。

```
bind(socket1,(struct sockaddr*)&local,sizeof(local));
while (1) {
    char buffer[1024]="\0";
    printf("waiting for receiving message\n");
    if (recvfrom(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&from, &fromlen) != SOCKET_ERROR) {     //参数中sockaddr_
        printf("Received datagram from %s : %s\n", inet_ntoa(from.sin_addr), buffer);     //inet_ntoa()将整型地址转化成点分十进
        count++;
        printf("You have received %d messgaes.\n\n", count);
        //sendto(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&from, fromlen);        //给cilent发信息
    }
}
```

服务器需要 bind()函数，作用是把端口，IP 和套接字绑定，然后客户端只要指定端口和 IP 就可以连接到 socket。其次，服务器通过 recvfrom（）不断接收客户端的数据包并且记录收到的数据包的个数以检查是否丢包。（代码中注释掉的部分与客户端注释掉的部分形成一个数据包检验过程，取消注释就可以有效避免丢包）。

```
closesocket(socket1);          //关闭套接字
WSACleanup();                  //终止使用Windows Sockets DLL
return 0;
```

这部分代码在客户端与服务器中都存在，用于关闭套接字和中止使用 DLL。

（3）使用 socket API 开发通信程序中的客户端程序和服务器程序时，各需要哪些不同的函数。

客户端用到了 socket（）构造函数、sendto（）、recvform（）、closesocket（）函数；

服务器端用到了 socket（）构造函数、sendto（）、recvform（）、closesocket（）函数，此外服务器端还使用了 bind（）函数把套接字和准备接收数据的接口绑定在一起。

（4）解释 connect（）、bind（）等函数中 struct sockaddr* addr 参数各个部分的含义，并用具体的数据举例说明。

首先看一下 sockaddr_in 和 sockaddr 结构体成员表：

```
Struct  sockaddr_in  {
    short  sin_family;    //AF_INET 地址族
    unsigned short  sin_port;    //使用的端口，2 字节
    struct  in_addr  sinaddr;        //IP 地址，4 字节
    char  sinzero[8];    //预留，8 字节
}

Struct  sockaddr  {
    u_short  sa_family;      //地址族
    char  sa_data[14];    //地址
}
```

sockaddr_in 和 sockaddr 是并列的结构，指向 sockaddr_in 的结构体的指针也可以指向 sockadd 的结构体，并代替它。在 sockaddr 中，IP 和端口号都存在 sa_data 中，所以我们只能通过 sockaddr_in 分别指定端口号和 IP，最后传给 connect()和 bind()时进行强制类型转换。

sa_family 是地址家族，一般都是“AF_xxx”的形式。通常是 AF_INET。

sa_data 是 14 字节协议地址。

例如：mysock.sa_family=AF_INET;

mysock.sin_addr.s_addr=inet_addr("192.168.0.1");

等到要做转换的时候用：

（struct sockaddr*）mysock；

（5）说明面向连接的客户端和面向非连接的客户端在建立 Socket 时有什么区别。

与面向连接的网络通信相比，面向非连接的网络通信不需要再服务器和客户端之间建立连接。面向非连接的 socket 通信是基于 UDP 的。在面向非连接的客户端建立 Socket 时不需要 connect()函数，connect()是用于建立与指定 socket 的连接。除此之外，面向非连接的客户端使用了 sendto()和 recvfrom()函数而不是面向连接的 TCP 客户端的 send()以及 recv()函数。

（6）说明面向连接的客户端和面向非连接的客户端在收发数据时有什么区别。面向非连接的客户端又是如何判断数据发送结束的。

“面向连接”就是在正式通信前必须要与对方建立起连接。TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，在正式收发数据前，必须和对方建立可靠的连接。一个 TCP 连

接必须要经过三次握手才能建立起来。当三次握手完成后才可以进行数据收发。

“面向非连接”是在正式通信前不必与对方先建立连接，不管对方状态就直接发送。UDP（User Data Protocol，用户数据报协议）是与 TCP 相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去。

面向连接的客户端在收发数据时传输速度较慢，而面向非连接的客户端传输速度快；面向连接的客户端保证数据顺序，而面向非连接的客户端不保证；面向连接的客户端保证数据正确性，而面向非连接的客户端可能丢包；面向连接的客户端对系统资源要求多，而面向非连接的客户端要求少。

sendto(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&server, len) != SOCKET_ERROR

使用 sendto 函数像服务器发送数据时，会得到返回值。成功则返回实际传送出去的字符数，失败返回－1，错误原因存于 errno 中。错误代码 EBADF 参数 s 非法的 socket 处理代码。EFAULT 参数中有一指针指向无法存取的内存空间。WNOTSOCK canshu s 为一文件描述词，非 socket。EINTR 被信号所中断。EAGAIN 此动作会令进程阻断，但参数 s 的 soket 为补课阻断的。ENOBUFS 系统的缓冲内存不足。EINVAL 传给系统调用的参数不正确。因此当收到返回值时不管成功失败都已经发送结束。

recvfrom(socket1,buffer,sizeof(buffer),0, (struct sockaddr*)&server, &len) != SOCKET_ERROR

我们在刚开始使得不丢包时还在客户端加了 recvfrom()函数以得到服务器接收到数据包后传回的信息，在得到信息后也表示本次数据发送结束。

（7）比较面向连接的通信和无连接通信，它们各有什么优点和缺点？适合在哪种场合下使用？

面向连接的通信的优点：

1. TCP 保证数据正确性，，无差错，不丢失，不重复
2.TCP 保证数据顺序
3.数据安全性容易保证

面向连接的通信的缺点：

1.对系统资源要求较大
2.首部较长，消耗较大
3.容易被利用来进行网络攻击

面向非连接的通信的优点:

1.UDP 的首部开销小
2.对系统资源的要求少
3.响应快，在网络负担重的时候能高效通信

面向非连接的通信的缺点:

1.通信不可靠。不保证送达
2.难以做到通信安全

3.数据不保证按顺序到达

分别适用的场景：

1.面向非连接的 UDP 通信：

a) 网络压力较大，又对信息传递速度要求高的时候（如网络聊天室）
b) 允许少量丢包的场景（低质量通话或者视频）
c) 不要求按序接受信息的时候（如下载）
d) 实时通信，有拥塞风险的时候
e) 用户数量庞大的时候

2.面向连接的 TCP 通信：

a) 安全要求高的通信
b) 高质量的网络通讯
c) 可靠的数据传输（如邮箱，ftp 等等）

（8）实验过程中使用 Socket 时是工作在阻塞方式还是非阻塞方式？通过网络检索阐述这两种操作方式的不同。

在实验中我们用的是 socket 阻塞模式。

```
printf("waiting for receiving message\n");
if (recvfrom(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&from, &fromlen) != SOCKET_ERROR) {
```

在服务器接收客户端发来的信息时，我们用的是 recvfrom(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&from, &fromlen) 函数。函数的第四个参数我们设为了 0，表示该接收函数的调用方式是阻塞模式的。如果该参数的值设为 MSG_DONTWAIT，则表示函数的调用方式是非阻塞模式的。

阻塞模式和非阻塞模式的主要区别在于无请求来到时，阻塞模式会一直停在接收函数即 recvfrom 函数，直到有请求到来才会继续向下进行处理。如果在队列中没有请求，该模式将会等待一个的到来.

而非阻塞模式下，运行接收函数，如果有请求，则会接收请求，如果无请求，会返回一个负值，并继续向下运行。即不管有没有接收到数据都会立即返回，而不会一直挂在此函数调用上。。一般来说，使用阻塞模式的程序比较多，

因为阻塞模式是由内核保障等待请求的，当他阻塞时不占用系统资源，而非阻塞模式需要我们人工轮询，占用资源较多。

【回答问题】引起 UDP 丢包的可能原因是什么？

1.发送的包过大丢包：虽然 sendto 方法会帮你做大包切割成小包发送的事情，但包太大也会造成一定程度的丢包。例如 64K 的一个 udp 包，不切割直接通过 sendto 方法发送也会导致这个包丢失。这种情况需要切割成小包再逐个 sendto。

2.发送的包频率太快丢包：UDP 的 sendto 不会像 TCP 中的 sendto 那样，直到数据完全发送才会 return 回调用函数，它不保证当执行下一条语句时数据是否被发送。（sendto 方法是异步的）这样，如果要发送的数据过多或者过大，那么在缓冲区满的那个瞬间要发送的报文就很有可能被丢失。这种情况有时可以通过设置 socket 接收缓冲解决，但有时解决不了。所以在发送频率过快的时候尽量采用 sleep（）来暂停一下使得缓冲区释放。

3.接收端处理时间过长导致丢包：调用 recvfrom 方法接收端收到数据后，处理数据花了一些时间，处理完后再次调用 recvfrom 方法，在这二次调用间隔里,发过来的包可能丢失。

4.发送的包超过接收端缓存丢包：包超过 mtu size 数倍，几个大的 udp 包可能会超过接收者的缓

冲，导致丢包。

附代码：

1. client.c

```
#include <stdio.h>
#include<iostream>
#include <string.h>
#include <stdlib.h>
#include <Winsock2.h>
int main() {
    char buffer[1024] = "\0";
    SOCKET socket1;
    WSADATA wsaData;          //Socket 的版本信息
    if (WSAStartup(MAKEWORD(2,2),&wsaData)) {          //调用 Windows Sockets DLL
        printf("init error\n");
        WSACleanup();
        return 0;
    }

    struct sockaddr_in server;
    int len =sizeof(server);          //服务器地址长度

    server.sin_family = AF_INET;          //AF_INET 地址族
    server.sin_port = htons(22222);                    ///server 的监听端口
    server.sin_addr.s_addr = inet_addr("192.168.199.208");          ///server 的地址

    socket1 = socket(AF_INET, SOCK_DGRAM, 0);       //创建套接字

    for(int i = 0; i < 1000; i++) {
        itoa(i, buffer, 10);
        printf(" message %d : %s\n", i, buffer);
        if (sendto(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&server, len) != SOCKET_ERROR) {          //参数中 sockaddr_in 转化 sockaddr
            //Sleep(10);
            //if (recvfrom(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&server, &len) != SOCKET_ERROR)
            // printf("rece from server:%s\n",buffer);
        }
        else {
            printf("send failed!\n");
        }
    }
```

```
        closesocket(socket1);          //关闭套接字
        WSACleanup();                  //终止使用 Windows Sockets DLL
        return 0;
    }
```

2. server.c

```
#include <stdio.h>
#include <Winsock2.h>
int main() {
    int count = 0;
    SOCKET socket1;
    WSADATA wsaData;                    //Socket 的版本信息
     if (WSAStartup(MAKEWORD(2,2),&wsaData)) {          //调用 Windows Sockets DLL
        printf("init error\n");
        WSACleanup();
        return 0;
     }
    printf("server activited\n");

    struct sockaddr_in local;
    struct sockaddr_in from;
    int fromlen =sizeof(from);                  //客户端地址长度

    local.sin_family=AF_INET;
    local.sin_port=htons(22222);                //监听端口
    local.sin_addr.s_addr=INADDR_ANY;           //本机

    socket1=socket(AF_INET,SOCK_DGRAM,0);       //创建套接字
    bind(socket1,(struct sockaddr*)&local,sizeof(local));
    while (1) {
        char buffer[1024]="\0";
        printf("waiting for receiving message\n");
        if (recvfrom(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&from, &fromlen) !=
SOCKET_ERROR) {       //参数中 sockaddr_in 转化 sockaddr
        printf("Received datagram from %s : %s\n", inet_ntoa(from.sin_addr), buffer);
//inet_ntoa()将整型地址转化成点分十进制
        count++;
        printf("You have received %d messgaes.\n\n", count);
        //sendto(socket1, buffer, sizeof(buffer), 0, (struct sockaddr*)&from, fromlen);
//给 cilent 发信息
        }
    }
    closesocket(socket1);               //关闭套接字
```

```
    WSACleanup();                    //终止使用 Windows Sockets DLL
    return 0;
}
```

本次实验完成后，请根据组员在实验中的贡献，请实事求是，自评在实验中应得的分数。（按百分制）

学号	学生	自评分
15331151	李佳	99
15331150	李辉旭	96
15331143	黎皓斌	96

【交实验报告】

上传实验报告：ftp://222.200.180.109/　　截止日期（不迟于）：1 周之内

上传包括两个文件：

（1）小组实验报告。上传文件名格式：小组号_ Ftp 协议分析实验.pdf （由组长负责上传）

例如: 文件名“10_ Ftp 协议分析实验.pdf”表示第 10 组的 Ftp 协议分析实验报告

（2）小组成员实验体会。每个同学单独交一份只填写了实验体会的实验报告。只需填写自己的学号和姓名。

文件名格式：小组号_学号_姓名_ Ftp 协议分析实验.pdf （由组员自行上传）

例如: 文件名“10_05373092_张三_ Ftp 协议分析实验.pdf”表示第 10 组的 Ftp 协议分析实验报告。

注意：不要打包上传！