中国矿业大学计算机学院

2019 级本科生计算机网络实验报告

实验内容 网络应用程序设计

学生姓名 李春阳 学 号 10193657

专业班级 信息安全 2019-1 班

学 院 计算机科学与技术学院

任课教师 姜秀柱

课程基础理论掌握程度	熟练 □	较熟练 □	一般 □	不熟练 □
综合知识应用能力	强 □	较强 □	一般 □	差 □
报告内容	完整 □	较完整 □	一般 □	不完整 □
报告格式	规范 □	较规范 □	一般 □	不规范 □
实验完成状况	好 □	较好 □	一般 □	差 □
工作量	饱满 □	适中 □	一般 □	欠缺 □
学习、工作态度	好 □	较好 □	一般 □	差 □
抄袭现象	无 □	有 □ 姓名:		
存在问题				
总体评价				

综合成绩: 任课教师签字:

年 月 日

实验编号：03

项目名称：网络应用程序设计

实验内容：

（1） 认识网络编程接口——Socket 的数据结构及常用 Socket 函数；

（2） 掌握基于 Socket 的网络编程方法和步骤；

（3） 熟悉 C（C++）或 Java 集成开发环境，及该环境下 Socket 资源的装入；

（4） 基于 Winsocket，完成简单通信程序编写和测试；

（5） 编写一个完整的 C/S 模式的网络应用程序；

（6） 编写一个实现 P2P 服务模式的网络应用程序（选做）。

实验要求：

（1） 熟练掌握 Socket 的概念及各个 Socket 函数的使用；

（2） 掌握面向连接和面向无连接的网络应用的区别及不同开发步骤；

（3） 掌握有状态和无状态网络服务及其开发；

（4） 掌握并发和循环两种服务方式下的网络服务程序的开发部署。

预习要求：

提前详细阅读实验指导书中该实验项目下的关于 Socket 的定义、数据结构和常用 Socket 函数类型及功能。以及面向连接和面向无连接的网络应用开发，对不同 Socket 函数的应用及步骤。

操作与观察：

正确按照实验指导书步骤操作，观察记录下操作结果。

实验报告要求：

（1） 按照实验要求，完成全部实验内容

（2） 在标准实验报告书上填写全部实验操作记录和观察结果

（3） 登录实验管理服务器，提交实验报告电子档。

实验报告内容：

1 认识网络编程接口

1.1socket 概述

socket 是在应用层和传输层之间的一个抽象层，它把 TCP/IP 层复杂的操作抽象为几个简单的接口供应用层调用已实现进程在网络中通信。

socket 起源于 UNIX，在 Unix 一切皆文件哲学的思想下，socket 是一种"打开—读/写—关闭"模式的实现，服务器和客户端各自维护一个"文件"，在建立连接打开后，可以向自己文件写入内容供对方读取或者读取对方内容，通讯结束时关闭文件。

1.2 接口详解

socket()：创建 socket

bind()：绑定 socket 到本地地址和端口，通常由服务端调用

listen()：TCP 专用，开启监听模式

accept()：TCP 专用，服务器等待客户端连接，一般是阻塞态

connect()：TCP 专用，客户端主动连接服务器

send()：TCP 专用，发送数据

recv()：TCP 专用，接收数据

sendto()：UDP 专用，发送数据到指定的 IP 地址和端口

recvfrom()：UDP 专用，接收数据，返回数据远端的 IP 地址和端口

closesocket()：关闭 socket

1.2.1 socket()

原型：int socket (int domain, int type, int protocol)

功能描述：

初始化创建 socket 对象，通常是第一个调用的 socket 函数。 成功时，返回非负数的 socket 描述符；失败是返回-1。socket 描述符是一个指向内部数据结构的指针，它指向描述符表入口。调用 socket()函数时，socket 执行体将建立一个 socket，实际上"建立一个 socket"意味着为一个 socket 数据结构分配存储空间。socket 执行体为你管理描述符表。

参数解释：

domain

domain – 指明使用的协议族。常用的协议族有，AF_INET、AF_INET6、AF_LOCAL（或称 AF_UNIX，Unix 域 socket）、AF_ROUTE 等等。协议族决定了 socket 的地址类型，在通信中必须采用对应的地址，如 AF_INET 决定了要用 ipv4 地址（32 位的）与端口号（16 位的）的组合、AF_UNIX 决定了要用一个绝对路径名作为地址。

type
指明 socket 类型，有 3 种:
SOCK_STREAM -- TCP 类型，保证数据顺序及可靠性;
SOCK_DGRAM -- UDP 类型，不保证数据接收的顺序，非可靠连接;
SOCK_RAW -- 原始类型，允许对底层协议如 IP 或 ICMP 进行直接访问，不太常用。

protocol
通常赋值"0"，由系统自动选择。

1.2.2 bind()

原型：int bind(int sockfd, const struct sockaddr* myaddr, socklen_t addrlen)

功能描述：

将创建的 socket 绑定到指定的 IP 地址和端口上，通常是第二个调用的 socket 接口。返回值：0 – 成功，-1 – 出错。当 socket 函数返回一个描述符时，只是存在于其协议族的空间中，并没有分配一个具体的协议地址（这里指 IPv4/IPv6 和端口号的组合），bind 函数可以将一组固定的地址绑定到 sockfd 上。

通常服务器在启动的时候都会绑定一个众所周知的协议地址，用于提供服务，客户就可以通过它来接连服务器；而客户端可以指定 IP 或端口也可以都不指定，未分配则系统自动分配。这就是为什么通常服务器端在 listen 之前会调用 bind()，而客户端就不会调用，而是在 connect()时由系统随机生成一个。

注意:
(1) 如果有多个可用的连接（多个 IP），内核会根据优先级选择一个 IP 作为源 IP 使用。
(2) 如果 socket 使用 bind 绑定到特定的 IP 和 port，则无论是 TCP 还是 UDP，都会从指定的 IP 和 port 发送数据。

参数解释:
sockfd:
socket()函数返回的描述符;

myaddr
指明要绑定的本地 IP 和端口号，使用网络字节序，即大端模式（详见 3.1）。

addrlen
常被设置为 sizeof(struct sockaddr)。
可以利用下边的赋值语句，自动绑定本地 IP 地址和随机端口:
my_addr.sin_port = 0; /* 系统随机选择一个未被使用的端口号 /
my_addr.sin_addr.s_addr = INADDR_ANY; / 填入本机 IP 地址 */

1。2.3 listen()

原型：int listen(int sockfd, int backlog)

功能描述:
listen()函数仅被 TCP 类型的服务器程序调用，实现监听服务，它实现 2 件事情:

“1. 当 socket()创建 1 个 socket 时，被假设为主动式套接字，也就是说它是一个将调用 connect()发起连接请求的客户端套接字；函数 listen()将套接口转换为被动式套接字，指示内核接受向此套接字的连接请求，调用此系统调用后 tcp 状态机由 close 转换到 listen。

2.第 2 个参数指定了内核为此套接字排队的最大连接个数。”

sten()成功时返回 0，错误时返回-1。

参数解释:

sockfd

socket()函数返回的描述符;

backlog

指定内核为此套接字维护的最大连接个数，包括“未完成连接队列–未完成 3 次握手”、“已完成连接队列–已完成 3 次握手，建立连接”。大多数系统缺省值为 20。

1.2.4 accept()

原型： int accept (int sockfd, struct sockaddr *addr, socklen_t *addrlen)

功能描述:

accept()函数仅被 TCP 类型的服务器程序调用，从已完成连接队列返回下一个建立成功的连接，如果已完成连接队列为空，线程进入阻塞态睡眠状态。成功时返回套接字描述符，错误时返回-1。

如果 accpet()执行成功，返回由内核自动生成的一个全新 socket 描述符，用它引用与客户端的 TCP 连接。通常我们把 accept()第一个参数成为监听套接字（listening socket），把 accept()功能返回值成为已连接套接字（connected socket）。一个服务器通常只有 1 个监听套接字，监听客户端的连接请求；服务器内核为每一个客户端的 TCP 连接维护 1 个已连接套接字，用它实现数据双向通信。

参数解释:

sockfd

socket()函数返回的描述符;

addr

输出一个的 sockaddr_in 变量地址，该变量用来存放发起连接请求的客户端的协议地址;

addrten

作为输入时指明缓冲器的长度，作为输出时指明 addr 的实际长度。

1.2.5 connetct()

原型： int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)

功能描述：connect()通常由 TCP 类型客户端调用，用来与服务器建立一个 TCP 连接，实际是发起 3 次握手过程，连接成功返回 0，连接失败返回 1。

注意：

(1) 可以在 UDP 连接使用使用 connect()，作用是在 UDP 套接字中记住目的地址和目的端口。

(2) UDP 套接字使用 connect 后，如果数据报不是 connect 中指定的地址和端口，将被

丢弃。没有调用 connect 的 UDP 套接字，将接收所有到达这个端口的 UDP 数据报，而不区分源端口和地址。

参数解释：
sockfd
本地客户端额 socket 描述符；

serv_addr
服务器协议地址；

addrlen
地址缓冲区的长度。

1.2.6 send()

原型：int send(int sockfd, const void *msg, int len, int flags)

功能描述：
TCP 类型的数据发送。
每个 TCP 套接口都有一个发送缓冲区，它的大小可以用 SO_SNDBUF 这个选项来改变。调用 send 函数的过程，实际是内核将用户数据拷贝至 TCP 套接口的发送缓冲区的过程：若 len 大于发送缓冲区大小，则返回-1；否则，查看缓冲区剩余空间是否容纳得下要发送的 len 长度，若不够，则拷贝一部分，并返回拷贝长度（指的是非阻塞 send，若为阻塞 send，则一定等待所有数据拷贝至缓冲区才返回，因此阻塞 send 返回值必定与 len 相等）；若缓冲区满，则等待发送，有剩余空间后拷贝至缓冲区；若在拷贝过程出现错误，则返回-1。关于错误的原因，查看 errno 的值。
如果 send 在等待协议发送数据时出现网络断开的情况，则会返回-1。注意：send 成功返回并不代表对方已接收到数据，如果后续的协议传输过程中出现网络错误，下一个 send 便会返回-1 发送错误。TCP 给对方的数据必须在对方给予确认时，方可删除发送缓冲区的数据。否则，会一直缓存在缓冲区直至发送成功（TCP 可靠数据传输决定的）。

参数解释：
sockfd
发送端套接字描述符（非监听描述符）。

msg
待发送数据的缓冲区。

len
待发送数据的字节长度。

flags
一般情况下置为 0。

2.7 recv()
原型：int recv(int sockfd, void *buf, int len, unsigned int flags)

功能描述：

TCP 类型的数据接收。

recv()从接收缓冲区拷贝数据。成功时，返回拷贝的字节数，失败返回-1。阻塞模式下，recv/recvfrom 将会阻塞到缓冲区里至少有一个字节(TCP)/至少有一个完整的 UDP 数据报才返回，没有数据时处于休眠状态。若非阻塞，则立即返回，有数据则返回拷贝的数据大小，否则返回错误-1。

参数解释：

sockefd
接收端套接字描述符（非监听描述符）；

buf
接收缓冲区的基地址；

len
以字节计算的接收缓冲区长度；

flags
一般情况下置为 0。

1.2.8 sendto()

原型：int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *dst_addr, int addrlen)

功能描述：

用于非可靠连接(UDP)的数据发送，因为 UDP 方式未建立连接 socket，因此需要制定目的协议地址。

当本地与不同目的地址通信时，只需指定目的地址，可使用同一个 UDP 套接口描述符 sockfd，而 TCP 要预先建立连接，每个连接都会产生不同的套接口描述符，体现在：客户端要使用不同的 fd 进行 connect，服务端每次 accept 产生不同的 fd。

因为 UDP 没有真正的发送缓冲区，因为是不可靠连接，不必保存应用进程的数据拷贝，应用进程中的数据在沿协议栈向下传递时，以某种形式拷贝到内核缓冲区，当数据链路层把数据传出后就把内核缓冲区中数据拷贝删除。因此它不需要一个发送缓冲区。写 UDP 套接口的 sendto/write 返回表示应用程序的数据或数据分片已经进入链路层的输出队列，如果输出队列没有足够的空间存放数据，将返回错误 ENOBUFS.

参数解释:

sockfd
发送端套接字描述符（非监听描述符）；

msg
待发送数据的缓冲区；

len
待发送数据的字节长度；

flags
一般情况下置为 0；

dst_addr
数据发送的目的地址；

addrlen
地址长度。

1.2.9 recvfrom()

原型：int recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr src_addr, intfromlen)

功能描述：
用于非可靠连接(UDP)的数据接收。

参数解释：
sockfd
接收端套接字描述；

buf
用于接收数据的应用缓冲区地址；

len
指名缓冲区大小；

flags
通常为 0；

src_addr
数据来源端的地址；

fromlen
作为输入时，fromlen 常置为 sizeof(struct sockaddr)；当输出时，fromlen 包含实际存入 buf 中的数据字节数。

2.socket 编程步骤

sockets（套接字）编程有三种，流式套接字（SOCK_STREAM），数据报套接字（SOCK_DGRAM），原始套接字（SOCK_RAW）；基于 TCP 的 socket 编程是采用的流式套接字。

2.1 服务器端编程的步骤：

1：加载套接字库，创建套接字(WSAStartup()/socket())；
2：绑定套接字到一个 IP 地址和一个端口上(bind())；
3：将套接字设置为监听模式等待连接请求(listen())；
4：请求到来后，接受连接请求，返回一个新的对应于此次连接的套接字(accept())；
5：用返回的套接字和客户端进行通信(send()/recv())；
6：返回，等待另一连接请求；

7：关闭套接字，关闭加载的套接字库(closesocket()/WSACleanup())。

2.2 客户端编程的步骤：

1：加载套接字库，创建套接字(WSAStartup()/socket())；
2：向服务器发出连接请求(connect())；
3：和服务器端进行通信(send()/recv())；
4：关闭套接字，关闭加载的套接字库(closesocket()/WSACleanup())。
第一式：加载/释放 Winsock 库:
1.加载方法:

```
WSADATA wsa;
/*初始化 socket 资源*/
if (WSAStartup(MAKEWORD(1,1),&wsa) != 0)
{
    return;   //代表失败
}
```

2.释放方法:

```
WSACleanup();
```

第二式：构造 SOCKET:
1.服务端:构造监听 SOCKET,流式 SOCKET.

```
   SOCKET Listen_Sock = socket(AF_INET, SOCK_STREAM, 0)
```

2.客户端:构造通讯 SOCKET,流式 SOCKET.

```
SOCKET Client_Sock = socket(AF_INET, SOCK_STREAM, 0)
```

第三式:配置监听地址和端口:
1.服务端: SOCKADDR_IN serverAddr

```
ZeroMemory((char *)&serverAddr,sizeof(serverAddr));
  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(1234);          /*本地监听端口:1234*/
  serverAddr.sin_addr.s_addr = htonl(INADDR_ANY); /*有 IP*/
```

第四式: 绑定 SOCKET:
1.服务端:绑定监听 SOCKET.

```
 bind(Listen_Sock,(struct sockaddr *)&serverAddr,sizeof(serverAddr))
```

第五式：服务端/客户端连接:
1.服务端:等待客户端接入.

```
   SOCKET Command_Sock = accept(Listen_Sock, ...)
```

2.客户端:请求与服务端连接.

```
int ret = connect(Client_Sock, ...)
```

第六式：收/发数据:
1.服务端:等待客户端接入.char buf[1024].
    接收数据:recv(Command_Sock,buf, ...)
或
    发送数据:send(Command_Sock,buf, ...)
2.客户端:请求与服务端连接.char buf[1024].

发送数据:send(Client_Sock,buf, ...)
或
接收数据:recv(Client_Sock,buf, ...)

第七式: 关闭 SOCKET:
1.服务端:关闭 SOCKET.
closesocket(Listen_Sock)
closesocket(Command_Sock)
2.客户端:关闭 SOCKET.
closesocket(Client_Sock)

3 基于 Winsocket，完成简单通信程序编写和测试

3.1TCP/IP 应用编程接口（API）

```
1.
2. #pragma comment(lib, "ws2_32.lib")
3. #include <Winsock2.h>
4. #include <stdio.h>
5. void main()
6. {
7.     //版本协商
8.     WORD wVersionRequested;
9.     WSADATA wsaData;
10.    int err;
11.    wVersionRequested = MAKEWORD(1, 1); //0x0101
12.    err = WSAStartup(wVersionRequested, &wsaData);
13.    if (err != 0)
14.    {
15.        return;
16.    }
17.    if (LOBYTE(wsaData.wVersion) != 1 || HIBYTE(wsaData.wVersion)
    != 1)  //wsaData.wVersion!=0x0101
18.    {
19.        WSACleanup();
20.        return;
21.    }
22.    //创建 Socket
23.    SOCKET sockSvr = socket(AF_INET, SOCK_STREAM, 0);
24.    //创建 IP 地址和端口
25.    SOCKADDR_IN addrSvr;
26.    addrSvr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
27.    addrSvr.sin_family = AF_INET;
28.    addrSvr.sin_port = htons(6000);
29.    //绑定端口监听
```

```
    bind(sockSvr, (SOCKADDR*)&addrSvr, sizeof(SOCKADDR));
    listen(sockSvr, 5);
    sockaddr_in addrClient;
    int len = sizeof(sockaddr);
    while (true)
    {
        //阻塞方法，获得一个客户Socket 连接
        SOCKET sockConn = accept(sockSvr, (sockaddr*)&addrClient, &len);
        char sendbuffer[128];
        sprintf(sendbuffer, "Welcom %s!", inet_ntoa(addrClient.sin_addr));
        //向客户Socket 发送数据
        send(sockConn, sendbuffer, strlen(sendbuffer) + 1, 0);
        char recvbuffer[128];
        //从客户Soc 接受数据
        recv(sockConn, recvbuffer, 128, 0);
        printf("%s\n", recvbuffer);
        //关闭Socket
        closesocket(sockConn);
    }
    closesocket(sockSvr);
    //释放Winsock 资源
    WSACleanup();
}
```

```

#pragma comment (lib,"ws2_32.lib")
#include <Winsock2.h>
#include <stdio.h>
void main()
{
    //版本协商
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    wVersionRequested = MAKEWORD(1, 1); //0x0101
    err = WSAStartup(wVersionRequested, &wsaData);
    if (err != 0)
    {
        return;
    }
```

```
    if (LOBYTE(wsaData.wVersion) != 1 || HIBYTE(wsaData.wVersion) != 1)     //wsaData.wVersion!=0x0101
    {
        WSACleanup();
        return;
    }
    //创建连向服务器的套接字
    SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
    //创建地址信息
    SOCKADDR_IN hostAddr;
    hostAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    hostAddr.sin_family = AF_INET;
    hostAddr.sin_port = htons(6000);
    //连接服务器
    connect(sock, (sockaddr*)&hostAddr, sizeof(sockaddr));
    char revBuf[128];
    //从服务器获得数据
    recv(sock, revBuf, 128, 0);
    printf("%s\n", revBuf);
    //向服务器发送数据
    send(sock, "Hello Host!", 12, 0);
    closesocket(sock);
}
```

3.2 UDP/IP 应用编程接口（API）

```
#pragma comment (lib,"ws2_32.lib")
#include <Winsock2.h>
#include <stdio.h>


////Server
void main()
{
    //版本协商
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    wVersionRequested = MAKEWORD(1, 1);
    err = WSAStartup(wVersionRequested, &wsaData);
    if (err != 0) {
        /* Tell the user that we could not find a usable */
        /* WinSock DLL.                                  */
```

```
18.         return;
19.     }
20.     /* Confirm that the WinSock DLL supports 2.2.*/
21.     /* Note that if the DLL supports versions greater    */
22.     /* than 2.2 in addition to 2.2, it will still return */
23.     /* 2.2 in wVersion since that is the version we      */
24.     /* requested.                                        */
25.     if (LOBYTE(wsaData.wVersion) != 1 ||
26.         HIBYTE(wsaData.wVersion) != 1) {
27.         /* Tell the user that we could not find a usable */
28.         /* WinSock DLL.                                  */
29.         WSACleanup();
30.         return;
31.     }
32.     /* The WinSock DLL is acceptable. Proceed. */
33.     //创建数据报套接字
34.     SOCKET svr = socket(AF_INET, SOCK_DGRAM, 0);
35.     //创建本地地址信息
36.     SOCKADDR_IN addr;
37.     addr.sin_family = AF_INET;
38.     addr.sin_port = htons(6000);
39.     addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
40.     int len = sizeof(sockaddr);
41.     bind(svr, (sockaddr*)&addr, len);
42.     //创建客户端地址对象
43.     SOCKADDR_IN addrClient;
44.     char recvBuf[128];
45.     char sendBuf[128];
46.     char tempBuf[256];
47.     while (true)
48.     {
49.         //接收数据
50.         printf("wait receive client message :\n");
51.         recvfrom(svr, recvBuf, 128, 0, (sockaddr*)&addrClient, &l
    en);
52.         char* ipClient = inet_ntoa(addrClient.sin_addr);
53.         sprintf(tempBuf, "%s said: %s\n", ipClient, recvBuf);
54.         printf("%s", tempBuf);
55.         gets(sendBuf);
56.         //发送数据
57.         sendto(svr, sendBuf, strlen(sendBuf) + 1, 0, (sockaddr*)&
    addrClient, len);
58.     }
59.     closesocket(svr);
```

```
    WSACleanup();
}
```

```
#pragma comment (lib,"ws2_32.lib")
#include <Winsock2.h>
#include <stdio.h>

//Client
void main()
{
    //版本协商
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;
    wVersionRequested = MAKEWORD(1, 1);
    err = WSAStartup(wVersionRequested, &wsaData);
    if (err != 0) {
        /* Tell the user that we could not find a usable */
        /* WinSock DLL.                                  */
        return;
    }
    /* Confirm that the WinSock DLL supports 2.2.*/
    /* Note that if the DLL supports versions greater    */
    /* than 2.2 in addition to 2.2, it will still return */
    /* 2.2 in wVersion since that is the version we      */
    /* requested.                                        */
    if (LOBYTE(wsaData.wVersion) != 1 ||
        HIBYTE(wsaData.wVersion) != 1) {
        /* Tell the user that we could not find a usable */
        /* WinSock DLL.                                  */
        WSACleanup();
        return;
    }
    /* The WinSock DLL is acceptable. Proceed. */
    //创建服务器套接字
    SOCKET Svr = socket(AF_INET, SOCK_DGRAM, 0);
    //创建地址
    SOCKADDR_IN addrSvr;
    addrSvr.sin_family = AF_INET;
    addrSvr.sin_port = htons(6000);
    addrSvr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    char recvBuf[128];
    char sendBuf[128];
```

```
    int len = sizeof(sockaddr);
    while (true)
    {
        printf("client frist send message:\n");
        gets(sendBuf);
        //发送数据

        sendto(Svr, sendBuf, strlen(sendBuf) + 1, 0, (sockaddr*)&addrSvr, len);   //sendBuf 调试在内存中显示的ASCII 码。
        //接收数据
        recvfrom(Svr, recvBuf, 128, 0, (sockaddr*)&addrSvr, &len);
        char* ipSvr = inet_ntoa(addrSvr.sin_addr);
        printf("%s said: %s\n", ipSvr, recvBuf);
    }
    closesocket(Svr);
    WSACleanup();
}
```

4 编写一个完整的 C/S 模式的网络应用程序

4.1 服务端

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include "WinSock2.h"
#include "iostream"
#pragma comment(lib,"ws2_32.lib")  //链接WinSock 导入库
#define PORT 65432
using namespace std;
int main(int argc, char** argv) {
 WSADATA wsaData;
 WORD wVersionRequested = MAKEWORD(2, 2);    //调用2.2 版本
 WSAStartup(wVersionRequested, &wsaData);

 SOCKET sock_server, newsock;    //监听套接字，已连接套接字
 struct sockaddr_in addr;        //填写绑定地址
 struct sockaddr_in client_addr;  //接收客户端发来的信息
 char msgbuffer[256];
 char msg[] = "Connect succeed.\n";  //发给客户端的信息

 //创建套接字
 sock_server = socket(AF_INET, SOCK_STREAM, 0);
```

```
20.
21. //填写要绑定的地址
22. int addr_len = sizeof(struct sockaddr_in);
23. memset((void*)&addr, 0, addr_len);
24. addr.sin_family = AF_INET;
25. addr.sin_port = htons(PORT);
26. addr.sin_addr.s_addr = htonl(INADDR_ANY);   //允许本机的任何IP地址
27.
28. //给监听套接字绑定地址
29. bind(sock_server, (struct sockaddr*)&addr, sizeof(addr));
30.
31. //将套接字设为监听状态
32. listen(sock_server, 0);
33. cout << "listenning......\n";
34.
35. //接收连接请求并收发数据
36. int size;
37. while (true) {
38.  newsock = accept(sock_server, (struct sockaddr*)&client_addr, &addr_len);  //接收请求
39.  cout << "成功接受一个连接请求！\n";
40.  size = send(newsock, msg, sizeof(msg), 0);  //给客户端发送一段信息
41.  if (size == SOCKET_ERROR) {
42.   cout << "发送信息失败！错误代码：" << WSAGetLastError() << endl;
43.   closesocket(newsock);
44.   continue;
45.  }
46.  else if (size == 0) {
47.   cout << "对方已关闭连接！\n";
48.   closesocket(newsock);
49.   continue;
50.  }
51.  else {
52.   cout << "信息发送成功！\n";
53.  }
54.  size = recv(newsock, msgbuffer, sizeof(msgbuffer), 0);
55.  if (size == 0) {
56.   cout << "对方已关闭链接！\n";
57.   closesocket(newsock);
58.   continue;
59.  }
60.  else {
```

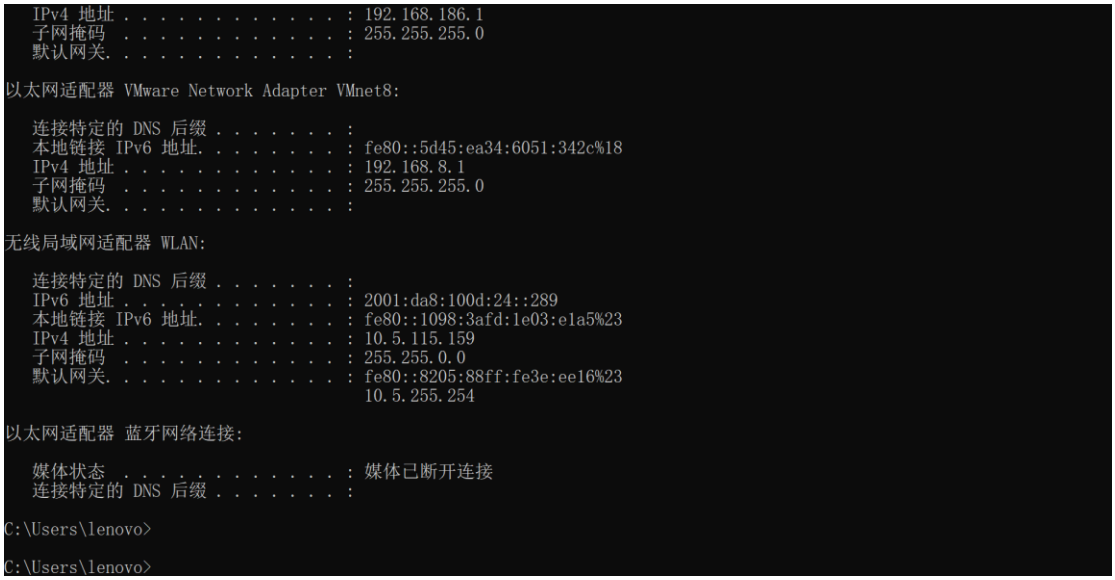
```
    cout << "收到的消息为:" << msgbuffer << endl;
    closesocket(newsock);
   }
 }
 closesocket(sock_server);
 WSACleanup();
 return 0;
}
```

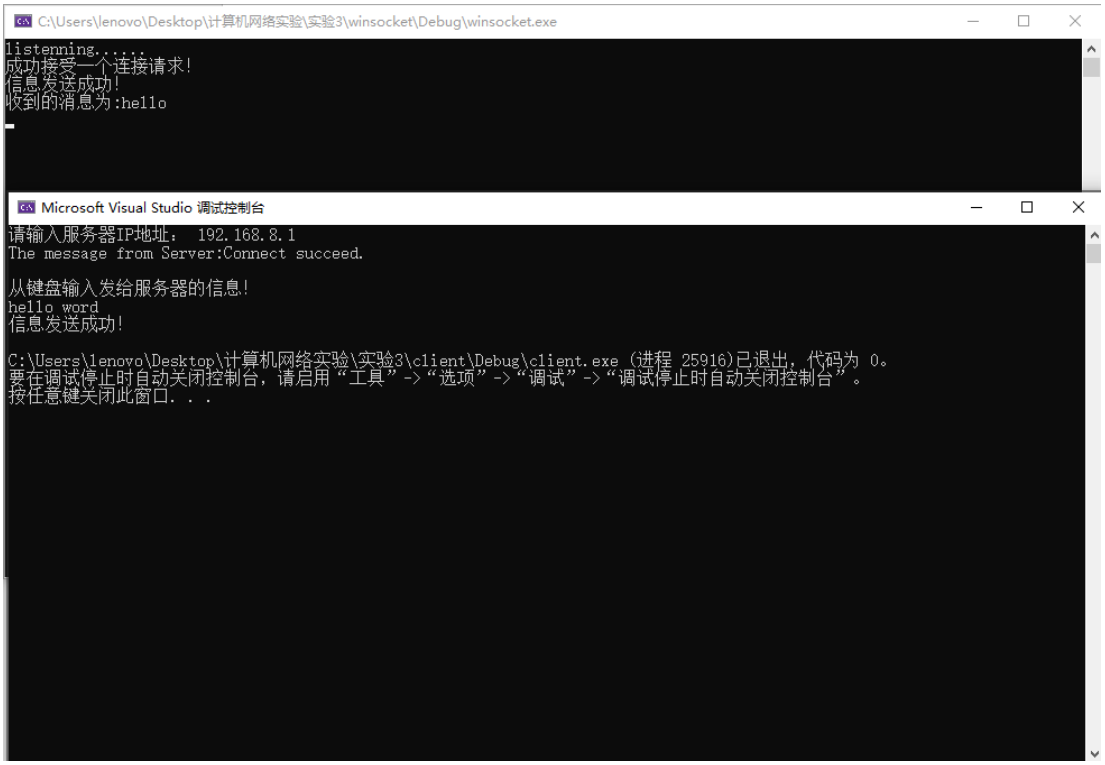
4.1 客户端

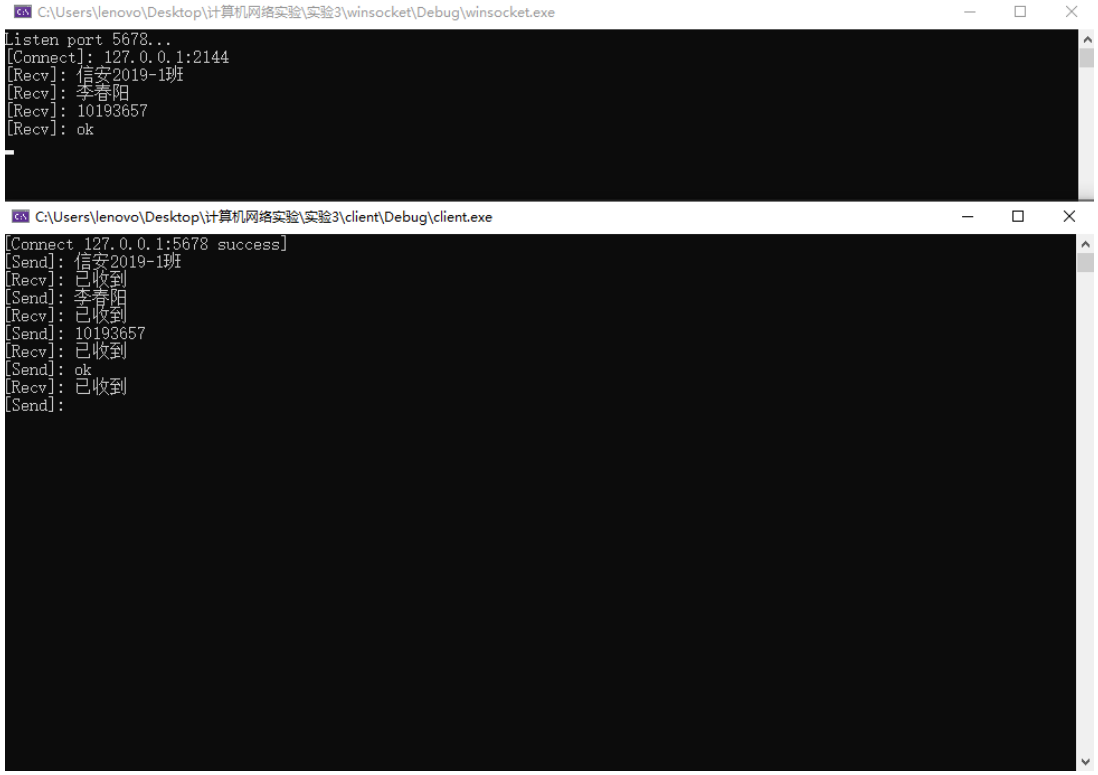
```
  #define _WINSOCK_DEPRECATED_NO_WARNINGS
#include "WinSock2.h"
#include "iostream"
#pragma comment(lib,"ws2_32.lib")  //链接WinSock 导入库
#define PORT 65432
using namespace std;
int main(int argc, char** argv) {
  WSADATA wsaData;
  WORD wVersionRequested = MAKEWORD(2, 2);   //调用2.2 版本
 WSAStartup(wVersionRequested, &wsaData);

 int sock_client;
 struct sockaddr_in server_addr;
 int addr_len = sizeof(struct sockaddr_in);
 char msgbuffer[1000];

 //创建套接字
 sock_client = socket(AF_INET, SOCK_STREAM, 0);

 //填写服务器地址
 char IP[20];
 cout << "请输入服务器 IP 地址：";
 cin >> IP;
 memset((void*)&server_addr, 0, addr_len);
 server_addr.sin_family = AF_INET;
 server_addr.sin_port = htons(PORT);
 server_addr.sin_addr.s_addr = inet_addr(IP);  //填写服务器IP 地址

 //连接
 connect(sock_client, (struct sockaddr*)&server_addr, addr_len);

 //接收信息
 int size;
```

```
34. size = recv(sock_client, msgbuffer, sizeof(msgbuffer), 0);
35. if (size == 0) {
36.  cout << "对方已关闭连接！\n";
37.  closesocket(sock_client);
38.  WSACleanup();
39.  return 0;
40. }
41. else {
42.  cout << "The message from Server:" << msgbuffer << endl;
43. }
44.
45. //发送
46. cout << "从键盘输入发给服务器的信息！\n";
47. cin >> msgbuffer;
48. size = send(sock_client, msgbuffer, sizeof(msgbuffer), 0);
49. if (size == 0) {
50.  cout << "对方已关闭链接！\n";
51. }
52. else {
53.  cout << "信息发送成功！\n";
54. }
55.
56. closesocket(sock_client);
57. WSACleanup();
58. return 0;
59.}
```

4.3 截图展示




C:\Users\lenovo\Desktop\计算机网络实验\实验3\winsocket\Debug\winsocket.exe
listenning.....
成功接受一个连接请求!
信息发送成功!
收到的消息为:hello
Microsoft Visual Studio 调试控制台
请输入服务器IP地址： 192.168.8.1
The message from Server:Connect succeed.
从键盘输入发给服务器的信息!
hello word
信息发送成功!
C:\Users\lenovo\Desktop\计算机网络实验\实验3\client\Debug\client.exe (进程 25916)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .


C:\Users\lenovo\Desktop\计算机网络实验\实验3\winsocket\Debug\winsocket.exe
Listen port 5678...
[Connect]: 127.0.0.1:2144
[Recv]: 信安2019-1班
[Recv]: 李春阳
[Recv]: 10193657
[Recv]: ok
C:\Users\lenovo\Desktop\计算机网络实验\实验3\client\Debug\client.exe
[Connect 127.0.0.1:5678 success]
[Send]: 信安2019-1班
[Recv]: 已收到
[Send]: 李春阳
[Recv]: 已收到
[Send]: 10193657
[Recv]: 已收到
[Send]: ok
[Recv]: 已收到
[Send]:

实验体会：

在本次实验中，我学习了如何利用 Socket 搭建网络交互进行编程

【TCP】

1. 服务器端
   1）创建套接字 create；
   2）绑定端口号 bind；
   3）监听连接 listen；
   4）接受连接请求 accept，并返回新的套接字；
   5）用新返回的套接字 recv/send；
   6）关闭套接字。
2. 客户端
   1）创建套接字 create;
   2）发起建立连接请求 connect;
   3）发送/接收数据 send/recv；
   4）关闭套接字。

【UDP】

1. 服务器端:
   1）创建套接字 create；
   2）绑定端口号 bind；
   3）接收/发送消息 recvfrom/sendto；
   4）关闭套接字。
2. 客户端:
   1）创建套接字 create；
   2）发送/接收消息 sendto/recvfrom；
   3）关闭套接字.

这对以后开发有极大帮助，顺便还更加深刻认识到了网络在应用层的应用以及如何进行通讯。什么是 socket，socket 的作用，基于 socket 的通信程序的编程步骤，客户端程序、服务器端程序的差异；基于 TCP（面向连接）和基于 UDP（无连接）的 socket 通信程序等。