**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. #pragma comment(lib, "ws2\_32.lib")
2. #include <Winsock2.h>
3. #include <stdio.h>
4. void main()
5. {
6. *//版本协商*
7. WORD wVersionRequested;
8. WSADATA wsaData;
9. int err;
10. wVersionRequested = MAKEWORD(1, 1); *//0x0101*
11. err = WSAStartup(wVersionRequested, &wsaData);
12. if (err != 0)
13. {
14. return;
15. }
16. if (LOBYTE(wsaData.wVersion) != 1 || HIBYTE(wsaData.wVersion) != 1)  *//wsaData.wVersion!=0x0101*
17. {
18. WSACleanup();
19. return;
20. }
21. *//创建Socket*
22. SOCKET sockSvr = socket(AF\_INET, SOCK\_STREAM, 0);
23. *//创建IP地址和端口*
24. SOCKADDR\_IN addrSvr;
25. addrSvr.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);
26. addrSvr.sin\_family = AF\_INET;
27. addrSvr.sin\_port = htons(6000);
28. *//绑定端口监听*
29. bind(sockSvr, (SOCKADDR\*)&addrSvr, sizeof(SOCKADDR));
30. listen(sockSvr, 5);
31. sockaddr\_in addrClient;
32. int len = sizeof(sockaddr);
33. while (true)
34. {
35. *//阻塞方法，获得一个客户Socket连接*
36. SOCKET sockConn = accept(sockSvr, (sockaddr\*)&addrClient, &len);
37. char sendbuffer[128];
38. sprintf(sendbuffer, "Welcom %s!", inet\_ntoa(addrClient.sin\_addr));
39. *//向客户Socket发送数据*
40. send(sockConn, sendbuffer, strlen(sendbuffer) + 1, 0);
41. char recvbuffer[128];
42. *//从客户Soc接受数据*
43. recv(sockConn, recvbuffer, 128, 0);
44. printf("%s\n", recvbuffer);
45. *//关闭Socket*
46. closesocket(sockConn);
47. }
48. closesocket(sockSvr);
49. *//释放Winsock资源*
50. WSACleanup();
51. }
52. #pragma comment (lib,"ws2\_32.lib")
53. #include <Winsock2.h>
54. #include <stdio.h>
55. void main()
56. {
57. *//版本协商*
58. WORD wVersionRequested;
59. WSADATA wsaData;
60. int err;
61. wVersionRequested = MAKEWORD(1, 1); *//0x0101*
62. err = WSAStartup(wVersionRequested, &wsaData);
63. if (err != 0)
64. {
65. return;
66. }
67. if (LOBYTE(wsaData.wVersion) != 1 || HIBYTE(wsaData.wVersion) != 1)     *//wsaData.wVersion!=0x0101*
68. {
69. WSACleanup();
70. return;
71. }
72. *//创建连向服务器的套接字*
73. SOCKET sock = socket(AF\_INET, SOCK\_STREAM, 0);
74. *//创建地址信息*
75. SOCKADDR\_IN hostAddr;
76. hostAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");
77. hostAddr.sin\_family = AF\_INET;
78. hostAddr.sin\_port = htons(6000);
79. *//连接服务器*
80. connect(sock, (sockaddr\*)&hostAddr, sizeof(sockaddr));
81. char revBuf[128];
82. *//从服务器获得数据*
83. recv(sock, revBuf, 128, 0);
84. printf("%s\n", revBuf);
85. *//向服务器发送数据*
86. send(sock, "Hello Host!", 12, 0);
87. closesocket(sock);
88. }

## 3.2 UDP/IP应用编程接口（API）

1. #pragma comment (lib,"ws2\_32.lib")
2. #include <Winsock2.h>
3. #include <stdio.h>
4. *////Server*
5. void main()
6. {
7. *//版本协商*
8. WORD wVersionRequested;
9. WSADATA wsaData;
10. int err;
11. wVersionRequested = MAKEWORD(1, 1);
12. err = WSAStartup(wVersionRequested, &wsaData);
13. if (err != 0) {
14. */\* Tell the user that we could not find a usable \*/*
15. */\* WinSock DLL.                                  \*/*
16. return;
17. }
18. */\* Confirm that the WinSock DLL supports 2.2.\*/*
19. */\* Note that if the DLL supports versions greater    \*/*
20. */\* than 2.2 in addition to 2.2, it will still return \*/*
21. */\* 2.2 in wVersion since that is the version we      \*/*
22. */\* requested.                                        \*/*
23. if (LOBYTE(wsaData.wVersion) != 1 ||
24. HIBYTE(wsaData.wVersion) != 1) {
25. */\* Tell the user that we could not find a usable \*/*
26. */\* WinSock DLL.                                  \*/*
27. WSACleanup();
28. return;
29. }
30. */\* The WinSock DLL is acceptable. Proceed. \*/*
31. *//创建数据报套接字*
32. SOCKET svr = socket(AF\_INET, SOCK\_DGRAM, 0);
33. *//创建本地地址信息*
34. SOCKADDR\_IN addr;
35. addr.sin\_family = AF\_INET;
36. addr.sin\_port = htons(6000);
37. addr.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);
38. int len = sizeof(sockaddr);
39. bind(svr, (sockaddr\*)&addr, len);
40. *//创建客户端地址对象*
41. SOCKADDR\_IN addrClient;
42. char recvBuf[128];
43. char sendBuf[128];
44. char tempBuf[256];
45. while (true)
46. {
47. *//接收数据*
48. printf("wait receive client message :\n");
49. recvfrom(svr, recvBuf, 128, 0, (sockaddr\*)&addrClient, &len);
50. char\* ipClient = inet\_ntoa(addrClient.sin\_addr);
51. sprintf(tempBuf, "%s said: %s\n", ipClient, recvBuf);
52. printf("%s", tempBuf);
53. gets(sendBuf);
54. *//发送数据*
55. sendto(svr, sendBuf, strlen(sendBuf) + 1, 0, (sockaddr\*)&addrClient, len);
56. }
57. closesocket(svr);
58. WSACleanup();
59. }
60. #pragma comment (lib,"ws2\_32.lib")
61. #include <Winsock2.h>
62. #include <stdio.h>
63. *//Client*
64. void main()
65. {
66. *//版本协商*
67. WORD wVersionRequested;
68. WSADATA wsaData;
69. int err;
70. wVersionRequested = MAKEWORD(1, 1);
71. err = WSAStartup(wVersionRequested, &wsaData);
72. if (err != 0) {
73. */\* Tell the user that we could not find a usable \*/*
74. */\* WinSock DLL.                                  \*/*
75. return;
76. }
77. */\* Confirm that the WinSock DLL supports 2.2.\*/*
78. */\* Note that if the DLL supports versions greater    \*/*
79. */\* than 2.2 in addition to 2.2, it will still return \*/*
80. */\* 2.2 in wVersion since that is the version we      \*/*
81. */\* requested.                                        \*/*
82. if (LOBYTE(wsaData.wVersion) != 1 ||
83. HIBYTE(wsaData.wVersion) != 1) {
84. */\* Tell the user that we could not find a usable \*/*
85. */\* WinSock DLL.                                  \*/*
86. WSACleanup();
87. return;
88. }
89. */\* The WinSock DLL is acceptable. Proceed. \*/*
90. *//创建服务器套接字*
91. SOCKET Svr = socket(AF\_INET, SOCK\_DGRAM, 0);
92. *//创建地址*
93. SOCKADDR\_IN addrSvr;
94. addrSvr.sin\_family = AF\_INET;
95. addrSvr.sin\_port = htons(6000);
96. addrSvr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");
97. char recvBuf[128];
98. char sendBuf[128];
99. int len = sizeof(sockaddr);
100. while (true)
101. {
102. printf("client frist send message:\n");
103. gets(sendBuf);
104. *//发送数据*
105. sendto(Svr, sendBuf, strlen(sendBuf) + 1, 0, (sockaddr\*)&addrSvr, len);   *//sendBuf调试在内存中显示的ASCII码。*
106. *//接收数据*
107. recvfrom(Svr, recvBuf, 128, 0, (sockaddr\*)&addrSvr, &len);
108. char\* ipSvr = inet\_ntoa(addrSvr.sin\_addr);
109. printf("%s said: %s\n", ipSvr, recvBuf);
110. }
111. closesocket(Svr);
112. WSACleanup();
113. }

# 4编写一个完整的C/S模式的网络应用程序

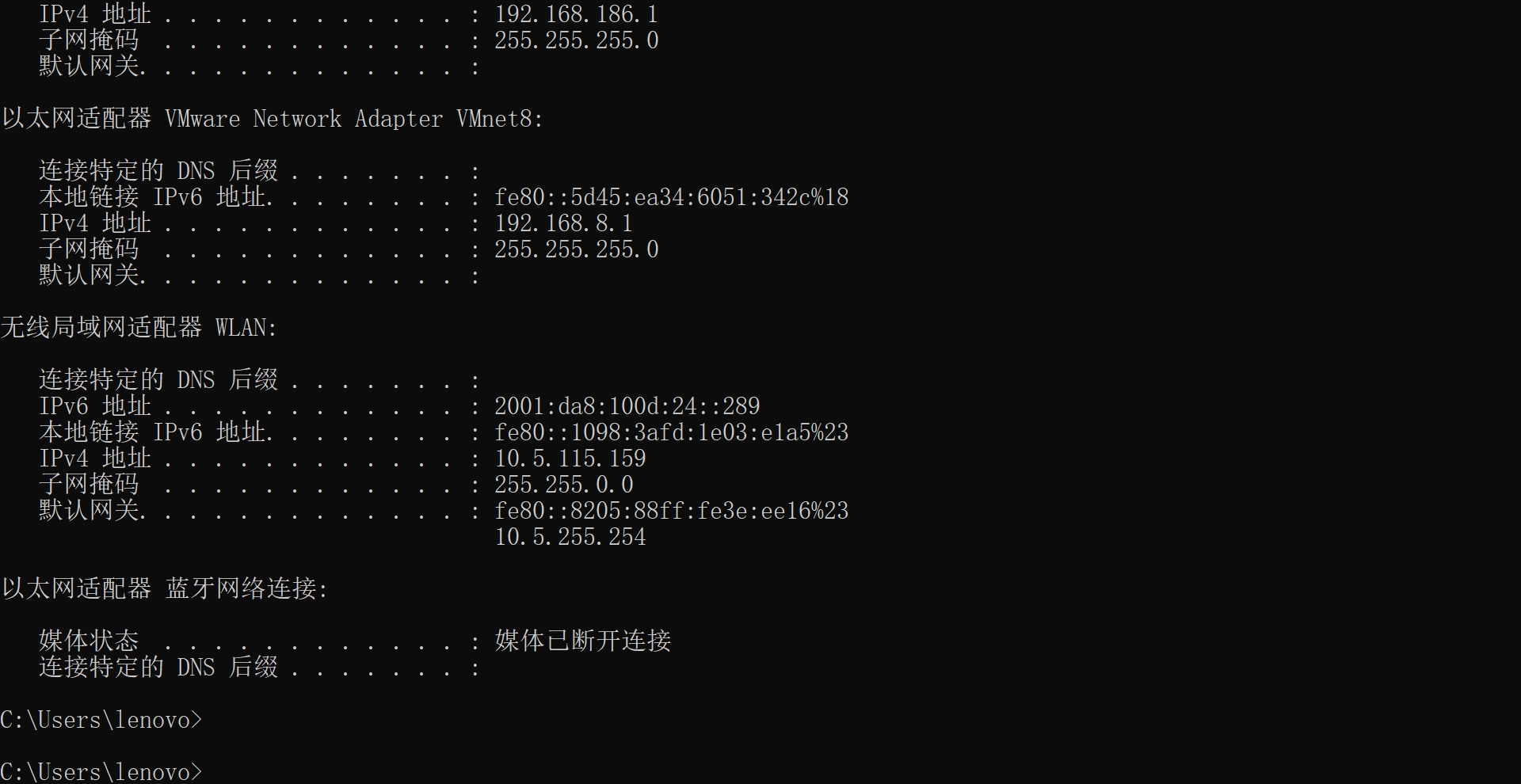
## 4.1服务端

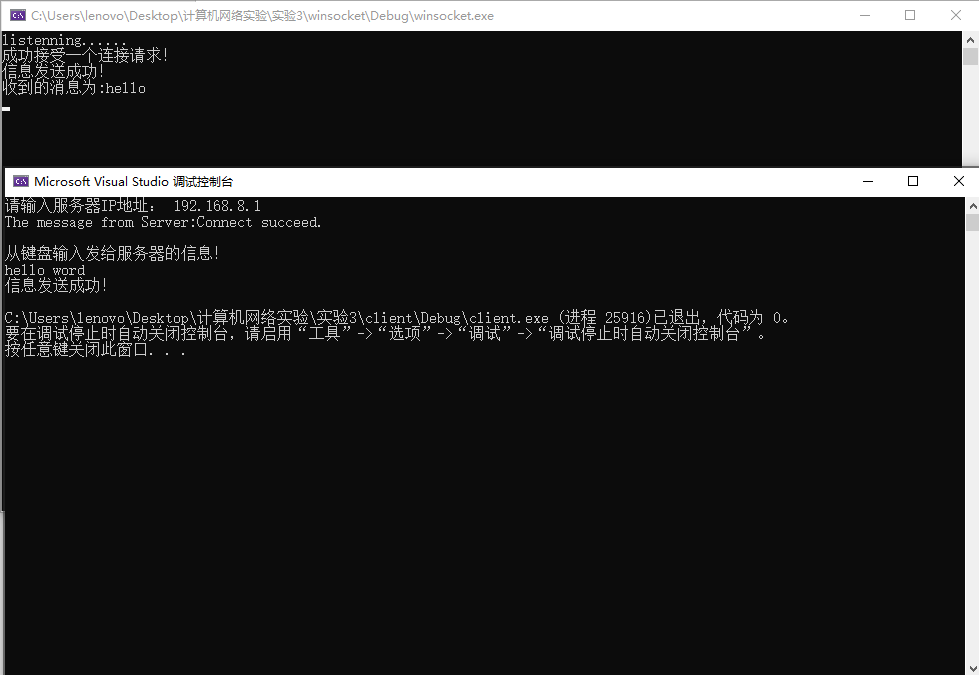
1. #define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS
2. #include "WinSock2.h"
3. #include "iostream"
4. #pragma comment(lib,"ws2\_32.lib")  *//链接WinSock导入库*
5. #define PORT 65432
6. using namespace std;
7. int main(int argc, char\*\* argv) {
8. WSADATA wsaData;
9. WORD wVersionRequested = MAKEWORD(2, 2);   *//调用2.2版本*
10. WSAStartup(wVersionRequested, &wsaData);
11. SOCKET sock\_server, newsock;   *//监听套接字，已连接套接字*
12. struct sockaddr\_in addr;       *//填写绑定地址*
13. struct sockaddr\_in client\_addr;  *//接收客户端发来的信息*
14. char msgbuffer[256];
15. char msg[] = "Connect succeed.\n";  *//发给客户端的信息*
16. *//创建套接字*
17. sock\_server = socket(AF\_INET, SOCK\_STREAM, 0);
18. *//填写要绑定的地址*
19. int addr\_len = sizeof(struct sockaddr\_in);
20. memset((void\*)&addr, 0, addr\_len);
21. addr.sin\_family = AF\_INET;
22. addr.sin\_port = htons(PORT);
23. addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);   *//允许本机的任何IP地址*
24. *//给监听套接字绑定地址*
25. bind(sock\_server, (struct sockaddr\*)&addr, sizeof(addr));
26. *//将套接字设为监听状态*
27. listen(sock\_server, 0);
28. cout << "listenning......\n";
29. *//接收连接请求并收发数据*
30. int size;
31. while (true) {
32. newsock = accept(sock\_server, (struct sockaddr\*)&client\_addr, &addr\_len);  *//接收请求*
33. cout << "成功接受一个连接请求！\n";
34. size = send(newsock, msg, sizeof(msg), 0);  *//给客户端发送一段信息*
35. if (size == SOCKET\_ERROR) {
36. cout << "发送信息失败！错误代码：" << WSAGetLastError() << endl;
37. closesocket(newsock);
38. continue;
39. }
40. else if (size == 0) {
41. cout << "对方已关闭连接！\n";
42. closesocket(newsock);
43. continue;
44. }
45. else {
46. cout << "信息发送成功！\n";
47. }
48. size = recv(newsock, msgbuffer, sizeof(msgbuffer), 0);
49. if (size == 0) {
50. cout << "对方已关闭链接！\n";
51. closesocket(newsock);
52. continue;
53. }
54. else {
55. cout << "收到的消息为:" << msgbuffer << endl;
56. closesocket(newsock);
57. }
58. }
59. closesocket(sock\_server);
60. WSACleanup();
61. return 0;
62. }

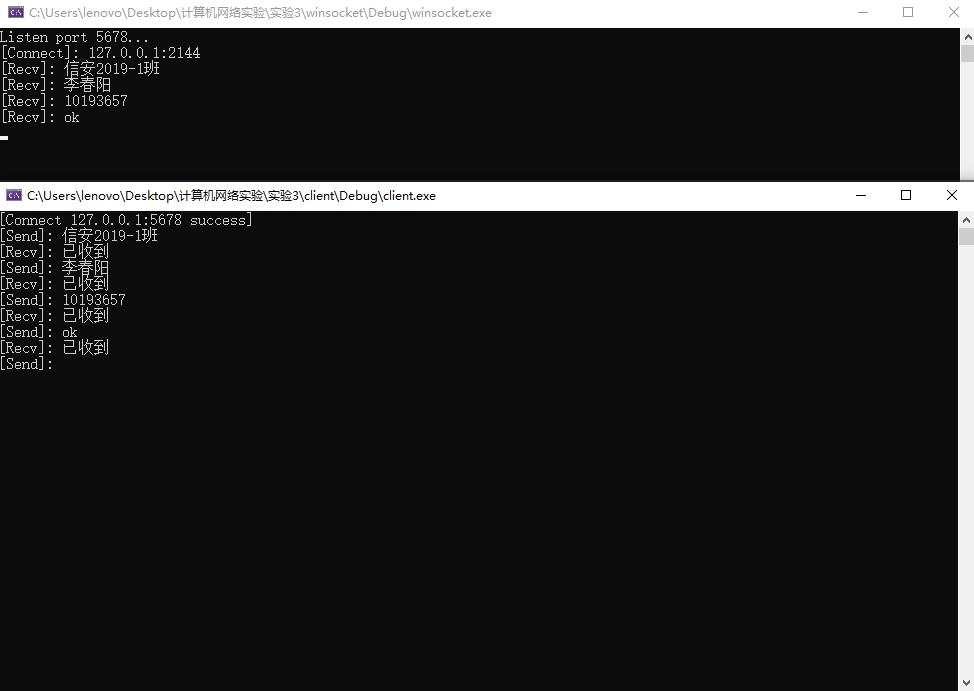
## 4.1客户端

1. #define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS
2. #include "WinSock2.h"
3. #include "iostream"
4. #pragma comment(lib,"ws2\_32.lib")  *//链接WinSock导入库*
5. #define PORT 65432
6. using namespace std;
7. int main(int argc, char\*\* argv) {
8. WSADATA wsaData;
9. WORD wVersionRequested = MAKEWORD(2, 2);   *//调用2.2版本*
10. WSAStartup(wVersionRequested, &wsaData);
11. int sock\_client;
12. struct sockaddr\_in server\_addr;
13. int addr\_len = sizeof(struct sockaddr\_in);
14. char msgbuffer[1000];
15. *//创建套接字*
16. sock\_client = socket(AF\_INET, SOCK\_STREAM, 0);
17. *//填写服务器地址*
18. char IP[20];
19. cout << "请输入服务器IP地址：";
20. cin >> IP;
21. memset((void\*)&server\_addr, 0, addr\_len);
22. server\_addr.sin\_family = AF\_INET;
23. server\_addr.sin\_port = htons(PORT);
24. server\_addr.sin\_addr.s\_addr = inet\_addr(IP);  *//填写服务器IP地址*
25. *//连接*
26. connect(sock\_client, (struct sockaddr\*)&server\_addr, addr\_len);
27. *//接收信息*
28. int size;
29. size = recv(sock\_client, msgbuffer, sizeof(msgbuffer), 0);
30. if (size == 0) {
31. cout << "对方已关闭连接！\n";
32. closesocket(sock\_client);
33. WSACleanup();
34. return 0;
35. }
36. else {
37. cout << "The message from Server:" << msgbuffer << endl;
38. }
39. *//发送*
40. cout << "从键盘输入发给服务器的信息！\n";
41. cin >> msgbuffer;
42. size = send(sock\_client, msgbuffer, sizeof(msgbuffer), 0);
43. if (size == 0) {
44. cout << "对方已关闭链接！\n";
45. }
46. else {
47. cout << "信息发送成功！\n";
48. }
49. closesocket(sock\_client);
50. WSACleanup();
51. return 0;
52. }

## 4.3截图展示







**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。