**MAKEWORD(a,b)函数**是将两个byte型合并成一个word型，一个在高8位(b)，一个在低8位(a)

**WSAStartup()函数**：使用Socket的程序，在使用Socket之前必须调用WSAStartup函数。该函数的第一个参数指明程序请求使用的Socket版本，其中高位字节指明副版本、低位字节指明主版本；操作系统利用第二个参数返回请求的Socket的版本信息。一个应用程序调用WSAStartup函数时，操作系统根据请求的Socket版本来搜索相应的Socket库，然后绑定找到的Socket库到该应用程序中。

**创建套接字──socket(int af, int type, int protocol)**

参数af指定通信发生的区域：AF\_UNIX、AF\_INET、AF\_NS等而DOS、WINDOWS中仅支持AF\_INET，它是网际网区域

参数type 描述要建立的套接字的类型,分三种：

SOCK\_STREAM TCP流式套接字，提供了一个面向连接、可靠的数据传输服务

SOCK\_DGRAM 数据报式套接字，提供了一个无连接服务。

SOCK\_RAW 原始式套接字，接口允许对较低层协议，如IP、ICMP直接访问

参数protocol说明该套接字使用的特定协议，如果调用者不希望特别指定使用的协议，则置为0，使用默认的连接模式。

**blind(SOCKET s,const struct SOCKADDR\_IN FAR \* name,int namelen)**

第一个参数就是套接字号

第二个参数结构体指针，其结构体定义如下：

第三个参数是name的长度

struct SOCKADDR\_IN  
{  
short sin\_family;//AF\_INET  
u\_short sin\_port;//16位端口号，网络字节顺序  
struct in\_addr sin\_addr;//32位IP地址，网络字节顺序  
char sin\_zero[8];//保留  
}

**accept(SOCKET s, struct sockaddr FAR\* addr, int FAR\* addrlen)函数**用于使服务器等待来自某客户进程的实际连接

参数s为本地套接字描述符，在用做accept()调用的参数前应该先调用过listen()，参数addr 指向客户方套接字地址结构的指针，用来接收连接实体的地址。addr的确切格式由套接字创建时建立的地址族决定。addrlen 为客户方套接字地址的长度（字节数）。如果没有错误发生，accept()返回一个SOCKET类型的值，表示接收到的套接字的描述符。否则返回值INVALID\_SOCKET。

accept()用于面向连接服务器。参数addr和addrlen存放客户方的地址信息。调用前，参数addr 指向一个初始值为空的地址结构，而addrlen 的初始值为0；调用accept()后，服务器等待从编号为s的套接字上接受客户连接请求，而连接请求是由客户方的connect()调用发出的。当有连接请求到达时，accept()调用将请求连接队列上的第一个客户方套接字地址及长度放入addr 和addrlen，并创建一个与s有相同特性的新套接字号。新的套接字可用于处理服务器并发请求。

**int PASCAL FAR connect(SOCKET s, const struct sockaddr FAR \* name, int namelen);**用于建立连接参数s是欲建立连接的本地套接字描述符。参数name指出说明对方套接字地址结构的指针。对方套接字地址长度由namelen。说明：如果没有错误发生，connect()返回0。否则返回值SOCKET\_ERROR。在面向连接的协议中，该调用导致本地系统和外部系统之间连接实际建立。

**send()调用**用于s指定的已连接的数据报或流套接字上发送输出数据，格式如下：

int PASCAL FAR send(SOCKET s, const char FAR \*buf, int len, int flags);

参数s为已连接的本地套接字描述符。buf 指向存有发送数据的缓冲区的指针，其长度由len 指定。flags 指定传输控制方式，如是否发送带外数据等。如果没有错误发生，send()返回总共发送的字节数。否则它返回SOCKET\_ERROR。

**recv()调用**用于s指定的已连接的数据报或流套接字上接收输入数据，格式如下

int PASCAL FAR recv(SOCKET s, char FAR \*buf, int len, int flags);

参数s 为已连接的套接字描述符。buf指向接收输入数据缓冲区的指针，其长度由len 指定。flags 指定传输控制方式，如是否接收带外数据等。如果没有错误发生，recv()返回总共接收的字节数。如果连接被关闭，返回0。否则它返回SOCKET\_ERROR。

**closesocket()关闭套接字s**，并释放分配给该套接字的资源；如果s涉及一个打开的TCP连接，则该连接被释放。 closesocket()的调用格式如下：BOOL PASCAL FAR closesocket(SOCKET s); 参数s待关闭的套接字描述符。如果没有错误发生，closesocket()返回0。否则返回值SOCKET\_ERROR。

**WSACleanup()**:中止了Windows Sockets在所有线程上的操作.

下面就是程序设计，如下：

//服务器端代码：

#include <WINSOCK2.H>

#include <stdio.h>

#pragma comment(lib,"ws2\_32.lib")

int main()

{

WORD myVersionRequest;

WSADATA wsaData;

int err;

char flag=0;

SOCKET serSocket;

SOCKADDR\_IN clientsocket;

SOCKADDR\_IN addr;

SOCKET serConn;

char receiveBuf[30]={0};//接收

char sendBuf[30]={0,0,0};

int len=sizeof(SOCKADDR);

myVersionRequest=MAKEWORD(1,1);

err=WSAStartup(myVersionRequest,&wsaData);

if (!err)

{

printf("已打开套接字\n");

}

else

{

printf("嵌套字未打开!");

return 0;

}

serSocket=socket(AF\_INET,SOCK\_STREAM,0);//创建了可识别套接字

addr.sin\_family=AF\_INET;

addr.sin\_addr.S\_un.S\_addr=htonl(INADDR\_ANY);//ip地址

addr.sin\_port=htons(789);//绑定端口

bind(serSocket,(SOCKADDR\*)&addr,sizeof(SOCKADDR));//绑定完成

listen(serSocket,5);//其中第二个参数代表能够接收的最多的连接数

while (1)

{

serConn=accept(serSocket,(SOCKADDR\*)&clientsocket,&len);

err=recv(serConn,receiveBuf,30,0);

if(err==30)

{

printf("接受成功!\n");

printf("%s\n",receiveBuf);

if(!strcmp("close serve",receiveBuf))flag=1;

sprintf(sendBuf,"hello client!");

err=send(serConn,sendBuf,strlen(sendBuf),0);

if(err==strlen(sendBuf))

{

printf("发送成功!\n");

}

else

{

printf("发送失败!\n");

}

}

else

{

printf("接受失败!\n");

}

if(flag)

{

closesocket(serConn);//关闭

break;

}

flag=0;

closesocket(serConn);//关闭

}

WSACleanup();//释放资源的操作

return 0;

}

//客户端代码

#include <WINSOCK2.H>

#include <stdio.h>

#pragma comment(lib,"ws2\_32.lib")

char \*sendBuf[3]={"hello,this is client","close serve","hello server"};

int main()

{

int err;

WORD versionRequired;

WSADATA wsaData;

char receiveBuf[30]={0,0,0};

SOCKET clientSocket;

SOCKADDR\_IN clientsock\_in;

versionRequired=MAKEWORD(1,1);

err=WSAStartup(versionRequired,&wsaData);//协议库的版本信息

if (!err)

{

printf("客户端嵌套字已经打开!\n");

}

else

{

printf("客户端的嵌套字打开失败!\n");

return 0;//结束

}

clientSocket=socket(AF\_INET,SOCK\_STREAM,0);

clientsock\_in.sin\_addr.S\_un.S\_addr=inet\_addr("127.0.0.1");

clientsock\_in.sin\_family=AF\_INET;

clientsock\_in.sin\_port=htons(789);

err=connect(clientSocket,(SOCKADDR\*)&clientsock\_in,sizeof(SOCKADDR));//开始连接

if(!err)

{

printf("连接成功!\n");

}

else

{

printf("连接失败!\n");

return 0;

}

err=send(clientSocket,sendBuf[2],30,0);

if(err==30)

{

printf("发送成功!\n");

}

else

{

printf("发送失败!\n");

}

err=recv(clientSocket,receiveBuf,101,0);

if(err>0)

{

printf("接受成功!\n");

printf("%s\n",receiveBuf);

}

else

{

printf("接受失败!\n");

}

closesocket(clientSocket);

WSACleanup();

return 0;

}

