# 一．socket 套接字

## 1. WSAStartup函数----异步套接字函数

WSAStartup必须是应用程序或DLL调用的第一个Windows Sockets函数。 它允许应用程序或DLL指明Windows Sockets API的版本号及获得特定Windows Sockets实现的细节。 应用程序或DLL只能在一次成功的WSAStartup ()调用之后才能调用进一步的Windows Sockets API函数 。

（1）功能

打开网络库/启动网络库，启动了这个库，这个库里的函数/功能才能使用。

WSAStartup：

W：windows

S：socket

A：Asynchronous 异步

同步：阻塞、卡死状态

异步：多个工作同时进行

Startup：启动

（2）原型：

int WSAAPI WSAStartup(

WORD wVersionRequested,

LPWSADATA lpWSAData

);

参数一：

WORD wVersionRequested----调用者可以使用的Windows套接字规范的最高版本。 高位字节指定次要版本号； 低位字节指定主要版本号（typedef unsigned short WORD）;

参数二：

LPWSADATA lpWSAData----指向WSADATA数据结构的指针，该数据结构将接收Windows套接字实现的详细信息。

返回值：

int 如果成功，则WSAStartup函数将返回0。 否则，它将返回下面列出的错误代码之一。WSAStartup函数直接在该函数的返回值中返回扩展错误代码。 不需要调用WSAGetLastError函数，并且不应使用该调用。



（3）使用

void network\_init(void)

{

#ifdef \_WIN32

WSADATA data;

(void) WSAStartup(MAKEWORD(2, 2), &data);

#endif /\* \_WIN32 \*/

//Sleep(2000);

}

## 2.socket()函数

这个函数建立一个协议族为domain、协议类型为type、协议编号为protocol的套接字文件描述符。如果函数调用成功，会返回一个标识这个套接字的文件描述符，失败的时候返回-1。

1. 函数原型

#include<sys/types.h>

#include<sys/socket.h>

int socket(int domain, int type, int protocol);

参数一domain：

函数socket()的参数domain用于设置网络通信的域，函数socket()根据这个参数选择通信协议的族。通信协议族在文件sys/socket.h中定义。也就是 IP 地址类型。AF\_INET 表示 IPv4 地址，例如 127.0.0.1；AF\_INET6 表示 IPv6 地址，例如 1030::C9B4:FF12:48AA:1A2B。

大家需要记住127.0.0.1，它是一个特殊IP地址，表示本机地址，后面的教程会经常用到。



参数二type：

函数socket()的参数type用于设置套接字通信的类型，主要有SOCKET\_STREAM（流式套接字）、SOCK——DGRAM（数据包套接字），一般用UDP等。



参数三protocol：

用于制定某个协议的特定类型，即type类型中的某个类型。通常某协议中只有一种特定类型，这样protocol参数仅能设置为0；但是有些协议有多种特定的类型，就需要设置这个参数来选择特定的类型。

* 类型为SOCK\_STREAM的套接字表示一个双向的字节流，与管道类似。流式的套接字在进行数据收发之前必须已经连接，连接使用connect()函数进行。一旦连接，可以使用read()或者write()函数进行数据的传输。流式通信方式保证数据不会丢失或者重复接收，当数据在一段时间内任然没有接受完毕，可以将这个连接人为已经死掉。
* SOCK\_DGRAM和SOCK\_RAW 这个两种套接字可以使用函数sendto()来发送数据，使用recvfrom()函数接受数据，recvfrom()接受来自制定IP地址的发送方的数据。
* SOCK\_PACKET是一种专用的数据包，它直接从设备驱动接受数据。

1. 使用

int fd = socket(PF\_INET, SOCK\_DGRAM , 0); //inport

## 3. ioctlsocket()函数

功能是设置[套接口](https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%8F%A3/10058888)的模式。

1. 原型

int ioctl[socket](https://baike.baidu.com/item/socket)( [i](https://baike.baidu.com/item/i)nt s, long [cmd](https://baike.baidu.com/item/cmd), u\_long \* argp);

参数一s：

一个标识套接口的描述字。也就是socket（）函数的返回值。

参数二cmd：

对套接口s的操作命令。支持下列命令：

* FIONBIO：

允许或禁止[套接口](https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%8F%A3)s的[非阻塞模式](https://baike.baidu.com/item/%E9%9D%9E%E9%98%BB%E5%A1%9E%E6%A8%A1%E5%BC%8F/12641137)。argp指向一个无符号[长整型](https://baike.baidu.com/item/%E9%95%BF%E6%95%B4%E5%9E%8B)，如允许非阻塞模式则非零，如禁止非阻塞模式则为零。当创建一个套接口时，它就处于阻塞模式（也就是说非阻塞模式被禁止）。这与BSD套接口是一致的。[WSAAsyncSelect()](https://baike.baidu.com/item/WSAAsyncSelect%28%29)函数将套接口自动设置为非阻塞模式。如果已对一个套接口进行了WSAAsyncSelect() 操作，则任何用ioctl[socket](https://baike.baidu.com/item/socket)()来把套接口重新设置成阻塞模式都会失败。为了把套接口重新设置成阻塞模式，应用程序必须首先用WSAAsyncSelect()调用（IEvent参数置为0）来禁止WSAAsyncSelect()。

* FIONREAD：

确定[套接口](https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%8F%A3)s自动读入的数据量。argp指向一个无符号[长整型](https://baike.baidu.com/item/%E9%95%BF%E6%95%B4%E5%9E%8B)，其中存有ioctl[socket](https://baike.baidu.com/item/socket)()的返回值。如果s是SOCKET\_STREAM类型，则FIONREAD返回在一次[recv()](https://baike.baidu.com/item/recv%28%29)中所接收的所有数据量。这通常与套接口中排队的数据总量相同。如果S是SOCK\_DGRAM 型，则FIONREAD返回套接口上排队的第一个数据报大小。

* SIOCATMARK：

确认是否所有的[带外数据](https://baike.baidu.com/item/%E5%B8%A6%E5%A4%96%E6%95%B0%E6%8D%AE)都已被读入。这个命令仅适用于SOCK\_STREAM类型的[套接口](https://baike.baidu.com/item/%E5%A5%97%E6%8E%A5%E5%8F%A3)，且该套接口已被设置为可以在线接收带外数据（SO\_OOBINLINE）。如无带外数据等待读入，则该操作返回TRUE真。否则的话返回FALSE假，下一个[recv()](https://baike.baidu.com/item/recv%28%29)或[recvfrom()](https://baike.baidu.com/item/recvfrom%28%29)操作将检索“标记”前一些或所有数据。应用程序可用SIOCATMARK操作来确定是否有数据剩下。如果在“紧急”（带外）数据[前有常规数据，则按序接收这些数据（请注意，recv()和recvfrom()操作不会在一次调用中混淆常规数据与带外数]据）。argp指向一个BOOL型数，ioctl[socket](https://baike.baidu.com/item/socket)()在其中存入返回值。

参数三argp：

指向cmd命令所带参数的[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88)。

1. 使用

static void set\_socket\_nonblocking(socket\_t s)

{

u\_long o = 1U; //非零

int r = ioctlsocket(s, FIONBIO, &o); //把S 改成非阻塞模式

}

## 4. setsockopt()函数

设置socket状态

1. 原型

#include <sys/types.h>

 #include <sys/socket.h>  
int setsockopt(int s, int level, int optname, const void \* optval, ,socklen\_toptlen);

setsockopt()用来设置参数s 所指定的socket 状态.

参数一s:

将要被设置或者获取选项的套接字。也就是socket（）函数的返回值。

参数二level：

选项所在的协议层, 指定控制套接字的层次。 一般设成SOL\_SOCKET 以存取socket 层.

1)SOL\_SOCKET:通用套接字选项.

2)IPPROTO\_IP:IP选项.

3)IPPROTO\_TCP:TCP选项.

参数三optname： 一般用SO\_REUSEADDR

需要访问的选项名, 有下列几种数值:

SO\_DEBUG 打开或关闭排错模式

SO\_REUSEADDR 允许重用本地地址和端口

SO\_TYPE 返回socket 形态.

SO\_ERROR 返回socket 已发生的错误原因

SO\_DONTROUTE 送出的数据包不要利用路由设备来传输.

SO\_BROADCAST 使用广播方式传送

SO\_SNDBUF 设置送出的暂存区大小

SO\_RCVBUF 设置接收的暂存区大小

SO\_KEEPALIVE 定期确定连线是否已终止.

SO\_OOBINLINE 当接收到OOB 数据时会马上送至标准输入设备

SO\_LINGER 确保数据安全且可靠的传送出去.

参数四optval：

指向包含新选项值的缓冲。

参数五optlen：

则为optval 的长度.

返回值：成功则返回0, 若有错误则返回-1, 错误原因存于errno.

1. 使用

int opt = 1;

(void) setsockopt(fd, SOL\_SOCKET, SO\_REUSEADDR, (void \*) &opt, (int) (sizeof opt));

## 5.bind()函数和connect()函数

socket() 函数用来创建套接字，确定套接字的各种属性，然后服务器端要用 bind() 函数将套接字与特定的IP地址和端口绑定起来，只有这样，流经该IP地址和端口的数据才能交给套接字处理；TCP socket客户端通过调用Connect函数来连接server端（socket执行体为你的程序自动选择一个未被占用的端口，并通知你的程序数据什么时候打开端口，因此无须调用bind()）。一般情况下客户端可以不绑定bind（）端口。

本函数适用于未连接的数据报或流类套接口，在connect()或listen()调用前使用。

注：一般不要将端口号置为小于1024的值，因为1到1024是保留端口号，你可以选择大于1024中的任何一个没有被占用的端口号

### bind() 函数

(1)原型：

int bind(int sock, struct sockaddr \*addr, socklen\_t addrlen);

sock 为 socket 文件描述符，addr 为 sockaddr 结构体变量的指针，addrlen 为 addr 变量的大小，可由 sizeof() 计算得出。

(2)使用

下面的代码，将创建的套接字与IP地址 127.0.0.1、端口 1234 绑定：

//创建套接字

int serv\_sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

//创建sockaddr\_in结构体变量

struct sockaddr\_in serv\_addr;

memset(&serv\_addr, 0, sizeof(serv\_addr)); //每个字节都用0填充

serv\_addr.sin\_family = AF\_INET; //使用IPv4地址

serv\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //具体的IP地址

serv\_addr.sin\_port = htons(1234); //端口

//将套接字和IP、端口绑定

bind(serv\_sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));

注：将sin\_addr设置为INADDR\_ANY的含义：

INADDR\_ANY转换过来就是0.0.0.0，泛指本机的意思，也就是表示本机的所有IP，因为有些机子不止一块网卡，多网卡的情况下，这个就表示所有网卡ip地址的意思。

当服务器的监听地址是INADDR\_ANY时，意思不是监听所有的客户端IP。而是服务器端的IP地址可以随意配置，这样使得该服务器端程序可以运行在任意计算机上，可使任意计算机作为服务器，便于程序移植。将INADDR\_ANY换成127.0.0.1也可以达到同样的目的。这样，当作为服务器的计算机的IP有变动或者网卡数量有增减，服务器端程序都能够正常监听来自客户端的请求。

### connect() 函数

connect() 函数是客户端用来建立连接，它的原型为：

int connect(int sock, struct sockaddr \*serv\_addr,socklen\_t addrlen);

## 6. send , sendto , recv, recvfrom函数

send、recv和sendto、recvfrom，一般情况下，send、recv在TCP协议下使用，sendto、recvfrom在UDP协议下使用，也可以在TCP协议下使用，不过用的很少。sendto可以在参数中指定发送的目标地址 , send需要socket已建立连接, sendto 可用于无连接的 socket 对于send的有连接socket,两者一样,sendto最后两个参数没用.

send 函数

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

不论是客户还是服务器应用程序都用send函数来向TCP连接的另一端发送数据。

客户程序一般用send函数向服务器发送请求，而服务器则通常用send函数来向客户程序发送应答。

第一个参数指定发送端套接字描述符；

第二个参数指明一个存放应用程序要发送数据的缓冲区；

第三个参数指明实际要发送的数据的字节数；

第四个参数一般设置为0。

recv函数

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

不论是客户还是服务器应用程序都用recv函数从TCP连接的另一端接收数据。

第一个参数该函数的第一个参数指定接收端套接字描述符；

第二个参数指明一个缓冲区，该缓冲区用来存放recv函数接收到的数据；

第三个参数指明buf的长度；

第四个参数一般置0。

sendto函数 和 recvfrom 函数一般用于UDP协议中,但是如果在 TCP 中 connect 函数调用后也可以用.

sendto() 和recvfrom() --------> 利用数据报文方式进行数据传输

　　在无连接的数据报socket方式下，由于本地socket并没有与远端机器建立连接，所以在发送数据时应指明目的地址，sendto() 函数原型为：

　　 int sendto（int sockfd， const void \*msg，int len , unsigned int flags， const struct sockaddr \*to， int tolen）；

该函数比 send() 函数多了两个参数，to表示目地机的IP地址和端口号信息，而tolen常常被赋值为sizeof （struct sockaddr）。sendto 函数也返回实际发送的数据字节长度或在出现发送错误时返回－1。

recvfrom() 函数原型为：

　　int recvfrom（int sockfd，void \*buf，int len，unsigned int lags，struct sockaddr \*from，int \*fromlen）；

　from是一个struct sockaddr类型的变量，该变量保存源机的IP地址及端口号。fromlen常置为sizeof （struct sockaddr）。当recvfrom() 返回时，fromlen 包含实际存入from中的数据字节数。recvfrom() 函数返回接收到的字节数或当出现错误时返回－1，并置相应的errno。

　 应注意的一点是，当你对于数据报socket调用了connect() 函数时，你也可以利用send() 和recv() 进行数据传输，但该socket仍然是数据报socket，并且利用传输层的UDP服务。但在发送或接收数据报时，内核会自动为之加上目地和源地址信息。

## 7. closesocket()函数

本函数关闭一个套接口。更确切地说，它释放套接口描述字s，以后对s的访问均以WSAENOTSOCK错误返回。若本次为对套接口的最后一次访问，则相应的名字信息及数据队列都将被释放。

#include <winsock.h>

int PASCAL FAR closesocket( SOCKET s);

s：一个套接口的描述字

# 二．AFDX总线协议规范

一种采用航空电子全双工通信以太网交换（AFDX），基于商业以太网标准，采用目前已被广泛接受的IEEE802.3/IP/UDP协议，并增加了特殊的功能来保证带宽和服务质量，实现了低成本的快速开发。该方案还可以简化布线，减轻飞机重量，易于航空电子子系统的维护升级等。（传统的航空总线如ARINC429等传输带宽只有100KHZ,远远不能满足要求，而ARINC629因价格昂贵，使飞机制造商的难以接受；因此需要以最小的代价和实现成本进行快速开发。）

# 三.创建线程

Windows使用CreateThread函数创建线程。线程是进程中的一个实体，是被系统独立调度和分派的基本单位。一个进程可以拥有多个线程，但是一个线程必须有一个进程。线程自己不拥有系统资源，只有运行所必须的一些数据结构，但它可以与同属于一个进程的其它线程共享进程所拥有的全部资源，同一个进程中的多个线程可以并发执行。

在C/C++中可以通过CreateThread函数在进程中创建线程，函数的具体格式如下：

HANDLE CreateThread(

LPSECURITY\_ATTRIBUTES lpThreadAttributes,

DWORD dwStackSize,

LPTHREAD\_START\_ROUTINE lpStartAddress,

LPVOID lpParameter,

DWORD dwCreationFlags,

LPDWORD lpThreadID

);

参数的含义如下：

参数一

lpThreadAttrivutes：指向SECURITY\_ATTRIBUTES的指针，用于定义新线程的安全属性，一般设置成NULL；

参数二

dwStackSize：分配以字节数表示的线程堆栈的大小，默认值是0；

参数三

lpStartAddress：指向一个线程函数地址。每个线程都有自己的线程函数，线程函数是线程具体的执行代码；

参数四

lpParameter：传递给线程函数的参数；

参数五

dwCreationFlags：表示创建线程的运行状态，其中CREATE\_SUSPEND表示挂起当前创建的线程，而0表示立即执行当前创建的进程；

参数六

lpThreadID：返回新创建的线程的ID编号；

CreateThread () 是原始Win32 API调用，用于在内核级别创建另一个控制线程。 \_beginthread () 和 \_beginthreadex () 是C运行时库调用，它们在后台调用 CreateThread () 。 一旦返回 CreateThread () ， \_beginthread/ex () 就会进行额外的记账，以使C运行时库在新线程中可用并保持一致。