**Socket原理与编程基础**

**一、Socket简介**

Socket是进程通讯的一种方式，即调用这个网络库的一些API函数（*操作系统除了协调应用程序的执行、内存分配、系统资源管理外，同时也是一个很大的服务中心，调用这个服务中心的各种服务（每一种服务是一个函数），可以帮助应用程序达到开启视窗、描绘图形、使用周边设备的目的，由于这些函数服务的对象是应用程序(Application),所以称之为Application Programming Interface，简称API函数。*）实现分布在不同主机的相关进程之间的数据交换。

几个定义：

（1）IP地址：即依照TCP/IP协议分配给本地主机的网络地址，两个进程要通讯，任一进程首先要知道通讯对方的位置，即对方的IP。

（2）端口号：用来辨别本地通讯进程，一个本地的进程在通讯时均会占用一个端口号，不同的进程端口号不同，因此在通讯前必须要分配一个没有被访问的端口号。

（3）连接：指两个进程间的通讯链路。

（4）半相关：网络中用一个三元组可以在全局唯一标志一个进程：

（协议，本地地址，本地端口号）

这样一个三元组，叫做一个半相关,它指定连接的每半部分。

（4）全相关：一个完整的网间进程通信需要由两个进程组成，并且只能使用同一种高层协议。也就是说，不可能通信的一端用TCP协议，而另一端用UDP协议。因此一个完整的网间通信需要一个五元组来标识：

（协议，本地地址，本地端口号，远地地址，远地端口号）

这样一个五元组，叫做一个相关（association），即两个协议相同的半相关才能组合成一个合适的相关，或完全指定组成一连接。

**二、客户/服务器模式**

在TCP/IP网络应用中，通信的两个进程间相互作用的主要模式是客户/服务器（Client/Server, C/S）模式，即客户向服务器发出服务请求，服务器接收到请求后，提供相应的服务。客户/服务器模式的建立基于以下两点：

（1）首先，建立网络的起因是网络中软硬件资源、运算能力和信息不均等，需要共享，从而造就拥有众多资源的主机提供服务，资源较少的客户请求服务这一非对等作用。

（2）其次，网间进程通信完全是异步的，相互通信的进程间既不存在父子关系，又不共享内存缓冲区，因此需要一种机制为希望通信的进程间建立联系，为二者的数据交换提供同步，这就是基于客户/服务器模式的TCP/IP。

**服务器端：**

其过程是首先服务器方要先启动，并根据请求提供相应服务：

（1）打开一通信通道并告知本地主机，它愿意在某一公认地址上的某端口（如FTP的端口可能为21）接收客户请求；

（2）等待客户请求到达该端口；

（3）接收到客户端的服务请求时，处理该请求并发送应答信号。接收到并发服务请求，要激活一新进程来处理这个客户请求（如UNIX系统中用fork、exec）。新进程处理此客户请求，并不需要对其它请求作出应答。服务完成后，关闭此新进程与客户的通信链路，并终止。

（4）返回第（2）步，等待另一客户请求。

（5）关闭服务器

**客户端：**

（1）打开一通信通道，并连接到服务器所在主机的特定端口；

（2）向服务器发服务请求报文，等待并接收应答；继续提出请求......

（3）请求结束后关闭通信通道并终止。

从上面所描述过程可知：

（1）客户与服务器进程的作用是非对称的，因此代码不同。

（2）服务器进程一般是先启动的。只要系统运行，该服务进程一直存在，直到正常或强迫终止。

**介绍完基础知识，下面就介绍一些API函数：**

**创建套接字──socket()**

应用程序在使用套接字前，首先必须拥有一个套接字，系统调用socket()向应用程序提供创建套接字的手段，其**调用格式**如下：

SOCKET PASCAL FAR socket(int af, int type, int protocol);

**该调用要接收三个参数：af、type、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。因此，地址族与协议族相同。参数type 描述要建立的**套接字的类型**。这里分三种：

（1）一是TCP流式套接字(SOCK\_STREAM)提供了一个面向连接、可靠的数据传输服务，数据无差错、无重复地发送，且按发送顺序接收。内设流量控制，避免数据流超限；数据被看作是字节流，无长度限制。文件传送协议（FTP）即使用流式套接字。

（2）二是数据报式套接字(SOCK\_DGRAM)提供了一个无连接服务。数据包以独立包形式被发送，不提供无错保证，数据可能丢失或重复，并且接收顺序混乱。网络文件系统（NFS）使用数据报式套接字。

（3）三是原始式套接字(SOCK\_RAW)该接口允许对较低层协议，如IP、ICMP直接访问。常用于检验新的协议实现或访问现有服务中配置的新设备。

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。根据这三个参数建立一个套接字，并将相应的资源分配给它，同时返回一个整型套接字号。因此，socket()系统调用实际上指定了相关五元组中的“协议”这一元。

**指定本地地址──bind()**

当一个套接字用socket()创建后，存在一个名字空间(地址族)，但它没有被命名。bind()将套接字地址（包括本地主机地址和本地端口地址）与所创建的套接字号联系起来，即将名字赋予套接字，以指定本地半相关。其调用格式如下：

int PASCAL FAR bind(SOCKET s, const struct sockaddr FAR \* name, int namelen);

参数s是由socket()调用返回的并且未作连接的套接字描述符(套接字号)。参数name 是赋给套接字s的本地地址（名字），其长度可变，结构随通信域的不同而不同。namelen表明了name的长度。如果没有错误发生，bind()返回0。否则返回SOCKET\_ERROR。

**建立套接字连接──connect()与accept()**

这两个系统调用用于完成一个完整相关的建立，其中connect()用于建立连接。accept()用于使服务器等待来自某客户进程的实际连接。

connect()的调用格式如下：

int PASCAL FAR connect(SOCKET s, const struct sockaddr FAR \* name, int namelen);

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。

由于地址族总被包含在套接字地址结构的前两个字节中，并通过socket()调用与某个协议族相关。因此bind()和connect()无须协议作为参数。

accept()的调用格式如下：

SOCKET PASCAL FAR accept(SOCKET s, struct sockaddr FAR\* addr, int FAR\* addrlen);

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()。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。

四个套接字系统调用，socket()、bind()、connect()、accept()，可以完成一个完全五元相关的建立。socket()指定五元组中的协议元，它的用法与是否为客户或服务器、是否面向连接无关。bind()指定五元组中的本地二元，即本地主机地址和端口号，其用法与是否面向连接有关：在服务器方，无论是否面向连接，均要调用bind()，若采用面向连接，则可以不调用bind()，而通过connect()自动完成。若采用无连接，客户方必须使用bind()以获得一个唯一的地址。

**监听连接──listen()**

此调用用于面向连接服务器，表明它愿意接收连接。listen()需在accept()之前调用，其调用格式如下：

int PASCAL FAR listen(SOCKET s, int backlog);

参数s标识一个本地已建立、尚未连接的套接字号，服务器愿意从它上面接收请求。backlog表示请求连接队列的最大长度，用于限制排队请求的个数，目前允许的最大值为5。如果没有错误发生，listen()返回0。否则它返回SOCKET\_ERROR。

listen()在执行调用过程中可为没有调用过bind()的套接字s完成所必须的连接，并建立长度为backlog的请求连接队列。

调用listen()是服务器接收一个连接请求的四个步骤中的第三步。它在调用socket()分配一个流套接字，且调用bind()给s赋于一个名字之后调用，而且一定要在accept()之前调用。

**数据传输──send()与recv()**

当一个连接建立以后，就可以传输数据了。常用的系统调用有send()和recv()。

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。

输入/输出多路复用──select()

select()调用用来检测一个或多个套接字的状态。对每一个套接字来说，这个调用可以请求读、写或错误状态方面的信息。请求给定状态的套接字集合由一个fd\_set结构指示。在返回时，此结构被更新，以反映那些满足特定条件的套接字的子集，同时， select()调用返回满足条件的套接字的数目，其调用格式如下：

int PASCAL FAR select(int nfds, fd\_set FAR \* readfds, fd\_set FAR \* writefds, fd\_set FAR \* exceptfds, const struct timeval FAR \* timeout);

参数nfds指明被检查的套接字描述符的值域，此变量一般被忽略。

参数readfds指向要做读检测的套接字描述符集合的指针，调用者希望从中读取数据。参数writefds 指向要做写检测的套接字描述符集合的指针。exceptfds指向要检测是否出错的套接字描述符集合的指针。timeout指向select()函数等待的最大时间，如果设为NULL则为阻塞操作。select()返回包含在fd\_set结构中已准备好的套接字描述符的总数目，或者是发生错误则返回SOCKET\_ERROR。

**关闭套接字──closesocket()**

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。

**以上就是SOCKET API一些常用的API函数，下面是一段代码：**

//客户端代码：

#include <WINSOCK2.H>

#include <stdio.h>

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

int main()

{

int err;

WORD versionRequired;

WSADATA wsaData;

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

}

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

SOCKADDR\_IN clientsock\_in;

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(6000);

//bind(clientSocket,(SOCKADDR\*)&clientsock\_in,strlen(SOCKADDR));//注意第三个参数

//listen(clientSocket,5);

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

char receiveBuf[100];

recv(clientSocket,receiveBuf,101,0);

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

send(clientSocket,"hello,this is client",strlen("hello,this is client")+1,0);

closesocket(clientSocket);

WSACleanup();

return 0;

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//服务器端代码：

#include <WINSOCK2.H>

#include <stdio.h>

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

int main()

{

//创建套接字

WORD myVersionRequest;

WSADATA wsaData;

myVersionRequest=MAKEWORD(1,1);

int err;

err=WSAStartup(myVersionRequest,&wsaData);

if (!err)

{

printf("已打开套接字\n");

}

else

{

//进一步绑定套接字

printf("嵌套字未打开!");

return 0;

}

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

//需要绑定的参数

SOCKADDR\_IN addr;

addr.sin\_family=AF\_INET;

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

addr.sin\_port=htons(6000);//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

//////////////////////////////////////////////////////////////////////////

//开始进行监听

//////////////////////////////////////////////////////////////////////////

SOCKADDR\_IN clientsocket;

int len=sizeof(SOCKADDR);

while (1)

{

SOCKET serConn=accept(serSocket,(SOCKADDR\*)&clientsocket,&len);//如果这里不是accept而是conection的话。。就会不断的监听

char sendBuf[100];

sprintf(sendBuf,"welcome %s to bejing",inet\_ntoa(clientsocket.sin\_addr));//找对对应的IP并且将这行字打印到那里

send(serConn,sendBuf,strlen(sendBuf)+1,0);

char receiveBuf[100];//接收

recv(serConn,receiveBuf,strlen(receiveBuf)+1,0);

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

closesocket(serConn);//关闭

WSACleanup();//释放资源的操作

}

return 0;

}