1:UDP中可以使用connect系统调用  
  
  
2:UDP中connect操作与TCP中connect操作有着本质区别。  
  
  
TCP中调用connect会引起三次握手,client与server建立连结.UDP中调用connect内核仅仅把对端ip&port记录下来.  
  
  
3:UDP中可以多次调用connect,TCP只能调用一次connect.    
  
  
UDP多次调用connect有两种用途:1,指定一个新的ip&port连结. 2,断开和之前的ip&port的连结.  
  
  
指定新连结,直接设置connect第二个参数即可.  
  
  
断开连结,需要将connect第二个参数中的sin\_family设置成 AF\_UNSPEC即可.   
  
  
4:UDP中使用connect可以提高效率.原因如下:  
  
  
普通的UDP发送两个报文内核做了如下:#1:建立连结#2:发送报文#3:断开连结#4:建立连结#5:发送报文#6:断开连结  
  
  
采用connect方式的UDP发送两个报文内核如下处理:#1:建立连结#2:发送报文#3:发送报文另外一点,  每次发送报文内核都由可能要做路由查询.  
  
  
5:采用connect的UDP发送接受报文可以调用send,write和recv,read操作.当然也可以调用sendto,recvfrom.  
  
  
调用sendto的时候第五个参数必须是NULL,第六个参数是0.调用recvfrom,recv,read系统调用只能获取到先前connect的ip&port发送的报文.   
  
  
  
  
UDP中使用connect的好处:1:会提升效率.前面已经描述了.2:高并发服务中会增加系统稳定性.原因:假设client A 通过非connect的UDP与server B,C通信.B,C提供相同服务.为了负载均衡,我们让A与B,C交替通信.A 与 B通信IPa:PORTa <----> IPb:PORTb；  
  
  
A 与 C通信IPa:PORTa' <---->IPc:PORTc   
  
  
假设PORTa 与 PORTa'相同了(在大并发情况下会发生这种情况),那么就有可能出现A等待B的报文,却收到了C的报文.导致收报错误.解决方法内就是采用connect的UDP通信方式.在A中创建两个udp,然后分别connect到B,C.  
  
  
================  
  
  
  
  
http://hi.baidu.com/rwen2012/item/545a39ba741307d085dd7957  
  
  
UDP编程中的connect(zt)  
标准的udp客户端开了套接口后，一般使用sendto和recvfrom函数来发数据，最近看到ntpclient的代码里面是使用send函数直接法的，就分析了一下，原来udp发送数据有两种方法供大家选用的，顺便把udp的connect用法也就解释清楚了。方法一： socket----->sendto()或recvfrom() 方法二： socket----->connect()----->send()或recv()首先从这里看出udp中也是可以使用connect的，但是这两种方法到底有什么区别呢？首先把这四个发送函数的定义列出来： int send(int s, const void \*msg, size\_t len, int flags); int sendto(int s, const void \*msg, size\_t len, int flags, const struct sockaddr \*to, socklen\_t tolen);int recv(int s, void \*buf, size\_t len, int flags);int  recvfrom(int  s, void \*buf, size\_t len, int flags, struct sockaddr \*from,  socklen\_t \*fromlen);从他们的定义可以看出，sendto和recvfrom在收发时指定地址，而send和recv则没有，那么他们的地址是在那里指定的呢，答案就在于connect.int  connect(int  sockfd,  const  struct sockaddr \*serv\_addr, socklen\_t addrlen);在udp编程中，如果你只往一个地址发送，那么你可以使用send和recv，在使用它们之前用connect把它们的目的地址指定一下就可以了。connect函数在udp中就是这个作用，用它来检测udp端口的是否开放是没有用的。下面是ntpclient中的代码struct sockaddr\_in sa\_dest;bzero((char \*) sa\_dest, sizeof(\*sa\_dest));sa\_dest->sin\_family=AF\_INET;if(StuffNetAddr(&(sa\_dest->sin\_addr),host))return 1;sa\_dest->sin\_port=htons(port);if (connect(usd,(struct sockaddr \*)&sa\_dest,sizeof(sa\_dest))==-1){perror("connect");return 1;}return 0;=================================  
除非套接口已连接，否则异步错误是不会返回到UDP套接口的，我们确实可以给UDP套接口调用connect，然而这样做的结果却与TCP连接大相径庭：没有三路握手过程。  
相反内核只是检查是否存在立即可知的错误(例如一个显然不可达的目的地)，记录对端的IP地址和端口号（取自传递给connect的套接口地址结构），然后立即返回到调用进程。  
对于已连接UDP套接口，与缺省的未连接套接口相比，发生了三个变化：1 我们再也不能给输出操作指定宿IP和端口号，也就是说我们不使用sendto，而改用write或send，写到已连接UDP套接口上的任何内容都自动发送到由connect指定的协议地址（例如IP地址和端口号）2 我们不必使用recvfrom以获悉数据报的发送者，而改用read，recv或recvmsg，在一个已连接UDP套接口上，由内核为输入操作返回的数据 报，仅仅是那些来自connect所指定协议地址的数据报。目的地为这个已连接UDP套接口的本地协议地址，发源地却不是该套接口早先connect到的协 议地址的数据报，不会投递到该套接口。这样就限制了一个已连接UDP套接口而且仅能与一个对端交换数据报。3 由已连接的UDP套接口引发的异步错误，返回给他们所在的进程。相反我们说过，未连接UDP套接口不接收任何异步错误给一个UDP套接口。  
多次调用connect拥有一个已连接UDP套接口的进程可以为下列2个目的之一：a.指定新的IP地址和端口号； b.断开套接口 第一个目的（即给一个已连接UDP套接口指定新的对端）不同于TCP套接口中connect的使用:对于TCP套接口，connect只能调用一次。为了断开一个已connect的UDP套接口连接，我们再次调用connect时把套接口地址结构的地址簇成员（sin\_family）设置为AF\_UNSPEC。 这么做可能返回一个EAFNOSUPPORT错误，不过没有关系。使得套接口断开连接的是在已连接UDP套接口上调用connect的进程。  
=================================有 **如下的一些好处** ：1）选定了对端， 内核只会将帮定对象的对端发来的数据报传给套接口 ，因此在一定环境下可以提升安全性；2)会返回异步错误，如果对端没启动，默认情况下发送的包对应的 ICMP回射包不会给调用进程 ，如果用了connect，嘿嘿3）发送两个包间不要 先断开再连接，提升了效率 。做个实验测试下吧先弄个UDP回射服务器，把所有收到的数据报回射回去：  
a@a-desktop:~/d/lab$ cat rollbackserver.cpp  
#include<iostream>  
#include<stdlib.h>  
#include<string.h>  
#include<unistd.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<arpa/inet.h>  
using namespace std;  
int main()  
{  
int sockListener,nMsgLen;  
char szBuf[1024];  
struct sockaddr\_in addrListener;  
socklen\_t addrLen;  
addrLen=sizeof(struct sockaddr\_in);  
bzero(&addrListener,sizeof(addrListener));  
addrListener.sin\_family=AF\_INET;  
addrListener.sin\_port=htons(8000);  
  
  
if((sockListener=socket(AF\_INET,SOCK\_DGRAM,0))==-1)  
{  
perror("error in getting a socket");  
exit(1);  
}  
  
  
if(bind(sockListener,(struct sockaddr\*)&addrListener,sizeof(addrListener))==-1)  
{  
perror("bind a listener for a socket");  
exit(2);  
}  
  
  
struct sockaddr\_in addrClient;  
cout<<"callback server begin to listen"<<endl;  
while(true)  
{  
nMsgLen=recvfrom(sockListener,szBuf,1024,0,(struct sockaddr\*)&addrClient,&addrLen);  
if(nMsgLen>0)  
{  
szBuf[nMsgLen]='\0';  
cout<<"send back:"<<szBuf<<endl;  
sendto(sockListener,szBuf,nMsgLen,0,(struct sockaddr\*)&addrClient,addrLen);  
}  
}  
  
  
}  
  
  
  
  
再写个客户端，绑定个端口，再连接服务器端。随时接受键盘输入并发送到服务器端，随时接受端口到来的数据并打印。如果没有连接 ，发送到此端口的数据会被接受，但是调用connect后会怎样呢？  
a-desktop:~/d/lab$ cat udpclient.cpp  
#include<iostream>  
#include<stdlib.h>  
#include<string.h>  
#include<unistd.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<arpa/inet.h>  
#include<sys/select.h>  
using namespace std;  
int main()  
{  
int sockClient,nMsgLen,nReady;  
char szRecv[1024],szSend[1024],szMsg[1024];  
struct sockaddr\_in addrServer,addrClient,addrLocal;  
socklen\_t addrLen;  
fd\_set setHold,setTest;  
  
  
sockClient=socket(AF\_INET,SOCK\_DGRAM,0);  
addrLen=sizeof(struct sockaddr\_in);  
bzero(&addrServer,sizeof(addrServer));  
addrServer.sin\_family=AF\_INET;  
addrServer.sin\_addr.s\_addr=inet\_addr("127.0.0.1");  
addrServer.sin\_port=htons(8000);  
  
  
addrLocal.sin\_family=AF\_INET;//bind to a local port  
addrLocal.sin\_addr.s\_addr=htonl(INADDR\_ANY);  
addrLocal.sin\_port=htons(9000);  
if(bind(sockClient,(struct sockaddr\*)&addrLocal,sizeof(addrLocal))==-1)  
{  
perror("error in binding");  
exit(2);  
}  
  
  
if(connect(sockClient,(struct sockaddr\*)&addrServer,sizeof(addrServer))==-1)  
{  
perror("error in connecting");  
exit(1);  
}  
  
  
FD\_ZERO(&setHold);  
FD\_SET(STDIN\_FILENO,&setHold);  
FD\_SET(sockClient,&setHold);  
cout<<"you can type in sentences any time"<<endl;  
while(true)  
{  
setTest=setHold;  
nReady=select(sockClient+1,&setTest,NULL,NULL,NULL);  
if(FD\_ISSET(0,&setTest))  
{  
nMsgLen=read(0,szMsg,1024);  
write(sockClient,szMsg,nMsgLen);  
}  
if(FD\_ISSET(sockClient,&setTest))  
{  
nMsgLen=read(sockClient,szRecv,1024);  
szRecv[nMsgLen]='\0';  
cout<<"read:"<<szRecv<<endl;  
}  
  
  
}  
  
  
}  
最后来个“第三者”，向第二个的端口发数据报。看她会不会成为忠贞的感情守护人:  
a@a-desktop:~/d/lab$ cat clienta.cpp  
#include<string.h>  
#include<iostream>  
#include<stdlib.h>  
#include<unistd.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<arpa/inet.h>  
using namespace std;  
int main()  
{  
socklen\_t addrLen=sizeof(struct sockaddr\_in);  
struct sockaddr\_in addrServer;  
char szMsg[1024];  
int sockClient;  
  
  
addrServer.sin\_family=AF\_INET;  
addrServer.sin\_addr.s\_addr=inet\_addr("127.0.0.1");  
addrServer.sin\_port=htons(9000);  
  
  
sockClient=socket(AF\_INET,SOCK\_DGRAM,0);  
while(true)  
{  
static int id=0;  
snprintf(szMsg,sizeof(szMsg),"this is %d",id++);  
sendto(sockClient,szMsg,strlen(szMsg),0,(struct sockaddr\*)&addrServer,sizeof(addrServer));  
sleep(1);  
}  
}  
  
  
  
  
  
  
实验结果：  
现运行第一个程序，再运行第三个程序，然后运行第二个程序。  
服务器端：  
  
  
a@a-desktop:~/d/lab$ ./rollback  
callback server begin to listen  
send back:xinheblue likes playing  
  
  
send back:and listenning to music  
  
  
  
  
第二个程序：  
a@a-desktop:~/d/lab$ ./udpclient  
you can type in sentences any time  
xinheblue likes playing  
read:xinheblue likes playing  
  
  
and listenning to music  
read:and listenning to music  
  
  
  
  
实现结果证明，第二个程序调用connect后，不甩第三个程序发来的数据包。  
/  
http://bbs.csdn.net/topics/290070552   有关UDP套接口connect()后  
有个事我一直不明白，connect后的udp套接口是怎么做到发送数据成功与否的检测的？直接sendto无法检测，为何connect后write就可以？  
  
  
udp connect 后 内核 记录住 你的connect中目的 IP 和 PORT 以后你就可以read 和 调用write 同时内核会告诉你所连接的套接字的异步错误  
比如：  
一旦出错向一个不存在的主机发送 会收到ICMP host unreachable 内核会帮你处理这个icmp报文 同时 write置错  
  
  
如果是非connect的话 如果内核也会收到这个ICMP（显然这肯定不是俺们能控制的，路由器发的），但是它就不care这个东东。  
  
  
  
  
至于原因：  
据说是 从发送 到 收到 icmp是有一定的时延的， 如果是 Sendto 你往二个目的地址 写数据报 1成功1失败 如果这时候内核收到icmp 报文它不知道 是哪个sendto。  
  
  
好象是UNP说的 ，不过感觉 icmp 如果返回的话，应该有发送的UDP头，其中包含 目的IP 和 目的端口 ，照理不也能 区分嘛？ 也没看完全明白。期待更强解释。。。   
/  
http://blog.csdn.net/rissonal/article/details/2816690  
  
  
转]UDP和socket函数(绑定端口)  
UDP是一个无连接的协议，因此socket函数connect似乎对UDP是没有意义的，  
然而事实不是这样。  
  
  
        一个插口有几个属性，其中包括协议，本地地址/端口，目的地址/端口。  
  
  
          对于UDP来说，socket函数建立一个插口；bind函数指明了本地地址/端口  
（包括ADDR\_ANY，通配所有本地网络接口）；connect可以用来指明目的地  
址/端口；  
  
  
         一般来说，UDP客户端在建立了插口后会直接用sendto函数发送数据，需要  
在sendto函数的参数里指明目的地址/端口。如果一个UDP客户端在建立了插  
口后首先用connect函数指明了目的地址/端口，然后也可以用send函数发送  
数据，因为此时send函数已经知道对方地址/端口，用getsockname也可以得  
到这个信息。  
  
  
         UDP客户端在建立了插口后会直接用sendto函数发送数据，还隐含了一个操作，  
那就是在发送数据之前，UDP会首先为该插口选择一个独立的UDP端口（在1024  
-5000之间），将该插口置为已绑定状态。如果一个UDP客户端在建立了插口后  
首先用bind函数指明了本地地址/端口，也是可以的，这样可以强迫UDP使用指  
定的端口发送数据。（事实上，UDP无所谓服务器和客户端，这里的界限已经模  
糊了。）  
  
  
       UDP服务器也可以使用connect，如上面所述，connect可以用来指明目的地址  
/端口；这将导致服务器只接受特定一个主机的请求。