                                     Windows socket select模型开发。

      套接字select模型是一种比较常用的IO模型。利用该模型可以使Windows socket应用程序可以同时管理多个套接字。

     使用select模型，可以使当执行操作的套接字满足可读可写条件时，给应用程序发送通知。收到这个通知后，应用程序再去调用相应的Windows socket API去执行函数调用。

     Select模型的核心是select函数。调用select函数检查当前各个套接字的状态。根据函数的返回值判断套接字的可读可写性。然后调用相应的Windows Sockets API完成数据的发送、接收等。

阻塞模式和非阻塞模式的优点和不足:

     阻塞模式套接字执行IO操作时，如果执行操作的条件未满足，线程就会阻塞在调用的函数上。程序不得不处于等待状态，但是由于并不知道客户请求何时到来，因此函数在何时返回不得而知。

    非阻塞模式套接字执行IO操作时，在任何时候函数都会立即返回。但程序员必须为此编写更多的代码。这增加了开发Windows socket应用程序的难度。另外由于不断的循环调用导致程序效率很低。

     Select模型是Windows sockets中最常见的IO模型。它利用select函数实现IO 管理。通过对select函数的调用，应用程序可以判断套接字是否存在数据、能否向该套接字写入数据。

     如：在调用recv函数之前，先调用select函数，如果系统没有可读数据那么select函数就会阻塞在这里。当系统存在可读或可写数据时，select函数返回，就可以调用recv函数接收数据了。

     可以看出使用select模型，需要两次调用函数。第一次调用select函数第二次socket API。使用该模式的好处是：可以等待多个套接字。

select函数

**[cpp]** [view plain](http://blog.csdn.net/ithzhang/article/details/8363951) [copy](http://blog.csdn.net/ithzhang/article/details/8363951)

1. <span style="font-size:18px;">**int** select (
2. Int nfds,//被忽略。传入0即可。
3. fd\_set \*readfds,//可读套接字集合。
4. fd\_set \*writefds,//可写套接字集合。
5. fd\_set \*exceptfds,//错误套接字集合。
6. **const** **struct** timeval\*timeout);//select函数等待时间。</span>

    该函数返回处于就绪态并且已经被包含在fd\_set结构中的套接字总数。如果超时则返回0。

    第一个参数nfds被忽略。

    第二个参数readfds，可读性套接字集合指针。

    第三个参数writefds，可写性套接字集合指针。

    第四个参数exceptfds，检查错误套接字集合指针。

    第五个参数timeout，等待时间。

fd\_set结构是一个结构体。

**[cpp]** [view plain](http://blog.csdn.net/ithzhang/article/details/8363951) [copy](http://blog.csdn.net/ithzhang/article/details/8363951)

1. <span style="font-size:18px;">**typedef** **struct** fd\_set
2. {
3. u\_int fd\_count;
4. socket fd\_array[FD\_SETSIZE];
5. }fd\_set;</span>

fd\_cout表示该集合套接字数量。最大为64.

fd\_array套接字数组。

select函数中需要三个fd\_set结构:

    一：准备接收数据的套接字集合，即可读性集合。

    二：准备发送数据的套接字集合，即可写性集合。

     在select函数返回时，会在fd\_set结构中，填入相应的套接字。

readfds数组将包括满足以下条件的套接字：

     1：有数据可读。此时在此套接字上调用recv，立即收到对方的数据。

     2：连接已经关闭、重设或终止。

     3：正在请求建立连接的套接字。此时调用accept函数会成功。

writefds数组包含满足下列条件的套接字：

    1：有数据可以发出。此时在此套接字上调用send,可以向对方发送数据。

    2：调用connect函数，并连接成功的套接字。

exceptfds数组将包括满足下列条件的套接字：

    1：调用connection函数，但连接失败的套接字。

    2：有带外（out of band）数据可读。

select函数的使用:

    在调用select函数对套接字进行监视之前，必须将要监视的套接字分配给上述三个数组中的一个。然后调用select函数，再次判断需要监视的套接字是否还在原来的集合中。就可以知道该集合是否正在发生IO操作。

    例如：应用程序想要判断某个套接字是否存在可读的数据，需要进行如下步骤：

    1：将该套接字加入到readfds集合。

    2：以readfds作为第二个参数调用select函数。

    3：当select函数返回时，应用程序判断该套接字是否仍然存在于readfds集合。

    4：如果该套接字存在与readfds集合，则表明该套接字可读。此时就可以调用recv函数接收数据。否则，该套接字不可读。

     在调用select函数时，readfds、writefds和exceptfds三个参数至少有一个为非空。并且在该非空的参数中，必须至少包含一个套接字。否则select函数将没有任何套接字可以等待。

timeval结构体用于定义select的等待时间。

**[cpp]** [view plain](http://blog.csdn.net/ithzhang/article/details/8363951) [copy](http://blog.csdn.net/ithzhang/article/details/8363951)

1. <span style="font-size:18px;">structure timeval
2. {
3. **long** tv\_sec;//秒。
4. **long** tv\_usec;//毫秒。
5. };</span>

    当timeval为空指针时，select会一直等待，直到有符合条件的套接字时才返回。

    当tv\_sec和tv\_usec之和为0时，无论是否有符合条件的套接字，select都会立即返回。

    当tv\_sec和tv\_usec之和为非0时，如果在等待的时间内有套接字满足条件，则该函数将返回符合条件的套接字。如果在等待的时间内没有套接字满足设置的条件，则select会在时间用完时返回，并且返回值为0。

    为了方便使用，windows sockets提供了下列宏，用来对fd\_set进行一系列操作。使用以下宏可以使编程工作简化。

    FD\_CLR(s,\*set);从set集合中删除s套接字。

    FD\_ISSET(s,\*set);检查s是否为set集合的成员。

    FD\_SET(s,\*set);将套接字加入到set集合中。

    FD\_ZERO(\*set);将set集合初始化为空集合。

   在开发Windows sockets应用程序时，通过下面的步骤，可以完成对套接字的可读写判断:

    1：使用FD\_ZERO初始化套接字集合。如FD\_ZERO(&readfds);

    2:使用FD\_SET将某套接字放到readfds内。如：

      FD\_SET(s,&readfds);

    3:以readfds为第二个参数调用select函数。select在返回时会返回所有fd\_set集合中套接字的总个数，并对每个集合进行相应的更新。将满足条件的套接字放在相应的集合中。

    4:使用FD\_ISSET判断s是否还在某个集合中。如：

       FD\_ISSET(s,&readfds);

     5:调用相应的Windows socket api 对某套接字进行操作。

     select返回后会修改每个fd\_set结构。删除不存在的或没有完成IO操作的套接字。这也正是在第四步中可以使用FD\_ISSET来判断一个套接字是否仍在集合中的原因。

     看例子，该例演示了一个服务器程序使用select模型管理套接字。

**[cpp]** [view plain](http://blog.csdn.net/ithzhang/article/details/8363951) [copy](http://blog.csdn.net/ithzhang/article/details/8363951)

1. SOCKET listenSocket;

**[cpp]** [view plain](http://blog.csdn.net/ithzhang/article/details/8363951) [copy](http://blog.csdn.net/ithzhang/article/details/8363951)

1. SOCKET acceptSocket;
2. FD\_SET socketSet;
3. FD\_SET writeSet;
4. FD\_SET readSet;


8. FD\_ZERO(&socketSet);
9. FD\_SET(listenSocket,&socketSet);
10. **while**(**true**)
11. {
12. FD\_ZERO(&readSet);
13. FD\_ZERO(&writeSet);
14. readSet=socketSet;
15. writeSet=socketSet;
17. //同时检查套接字的可读可写性。
18. **int** ret=select(0,&readSet,&writeSet,NULL,NULL);//为等待时间传入NULL，则永久等待。传入0立即返回。不要勿用。
19. **if**(ret==SOCKET\_ERROR)
20. {
21. **return** **false**;
22. }
23. sockaddr\_in addr;
24. **int** len=**sizeof**(addr);
25. //是否存在客户端的连接请求。
26. **if**(FD\_ISSET(listenSocket,&readSet))//在readset中会返回已经调用过listen的套接字。
27. {
28. acceptSocket=accept(listenSocket,(sockaddr\*)&addr,&len);
29. **if**(acceptSocket==INVALID\_SOCKET)
30. {
31. **return** **false**;
32. }
33. **else**
34. {
35. FD\_SET(acceptSocket,&socketSet);
36. }
37. }
39. **for**(**int** i=0;i<socketSet.fd\_count;i++)
40. {
41. **if**(FD\_ISSET(socketSet.fd\_array[i],&readSet))
42. {
43. //调用recv，接收数据。
44. }
45. **if**(FD\_ISSET(socketSet.fd\_array[i]),&writeSet)
46. {
47. //调用send，发送数据。
48. }
49. }
50. }

         以下展示了一个客户端程序使用select模型的用法。注意与服务器用法相区别。主要区别就是不可能有请求进来，也就不需要使用allsocketfds。仅仅对一个套接字进行判断：

**[cpp]** [view plain](http://blog.csdn.net/ithzhang/article/details/8363951) [copy](http://blog.csdn.net/ithzhang/article/details/8363951)

1. CRemoteFileDownloadClientDlg\*pdlg=(CRemoteFileDownloadClientDlg\*)ppram;
3. FD\_SET readfds;
4. FD\_SET writefds;
6. **while**(pdlg->m\_IsConnected)
7. {
8. FD\_ZERO(&readfds);
9. FD\_ZERO(&writefds);
10. FD\_SET(pdlg->m\_ServerSocket,&readfds);
11. FD\_SET(pdlg->m\_ServerSocket,&writefds);

14. **int** ret=select(0,&readfds,&writefds,NULL,NULL);//NULL为无限等待。0立即返回。
15. **if**(ret>0)
16. {
17. **if**(FD\_ISSET(pdlg->m\_ServerSocket,&readfds));//注意与服务器此处写法相区别。
18. {
19. pdlg->recvData();
20. }
21. **if**(FD\_ISSET(pdlg->m\_ServerSocket,&writefds))
22. {
23. //可写。
24. pdlg->sendData();
25. }
27. }
29. }