# WinSock IO模型之WSAEventSelect

## 一、原理及主要函数

### 1.1原理

使用WSAEventSelect模型时行I/O异步的关键是将SOCKET和一个WSAEvent关联起来，当SOCKET上有网络事件发生时，会触发WSAEvent事件，通过WSAWaitForMultipleEvents函数可以等待事件触发，并通过返回值得到发生事件的SOCKET序号，通过WSAEnumNetworkEvents函数可以获得网络事件类型，进而可以作相应的处理。

服务器端流程：

初始化套接字库

创建监听套接字

绑定套接字

设置监听状态

创建手动重置事件对象，关联套接字

循环等待并处理网络事件

### 1.2 主要函数

#### 1、初始化套接字库

WSADATA data;

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

#### 2、创建事件对象

WSACreateEvent()

该函数创建一个人工重置的匿名事件对象，该对象要与SOCKET一一对应，每一个SOCKET需要一个事件对象相关联。

#### 3、关联SOCKET和事件对象

WSAEventSelect函数，其原型如下：

int WSAEventSelect(

\_In\_  SOCKET s,

\_In\_  WSAEVENT hEventObject,

\_In\_  long lNetworkEvents

);

第一个参数为SOCKET，第二个参数为事件对象，第三个参数为需要关注的网络事件，对于监听SOCKET可以设置为FD\_CLOSE|FD\_ACCEPT，对于通信SOCKET，可以设置为FD\_READ|FD\_WRITE|FD\_CLOSE，其它值不常用。

#### 4、等待网络事件

WSAWaitForMultipleEvents函数，该函数原型如下：

DWORD WSAWaitForMultipleEvents(

\_In\_  DWORD cEvents,

\_In\_  const WSAEVENT \*lphEvents,

\_In\_  BOOL fWaitAll,

\_In\_  DWORD dwTimeout,

\_In\_  BOOL fAlertable

);

第一个参数为等待的事件对象个数，第二个参数为事件对象数组，第三个参数是否等待全部事件对象触发，第四个参数为等待时间，第五个参数需要设置为FALSE。函数返回值与事件对象数组中被触发的事件对象序号有关，序号可以这样得到：

nIndex=WSAWaitForMultipleEvents(nTotalEvent,EventArray,FALSE,WSA\_INFINITE,FALSE);

if (nIndex==WSA\_WAIT\_FAILED||nIndex==WSA\_WAIT\_TIMEOUT)

{

continue;

}

nIndex=nIndex-WSA\_WAIT\_EVENT\_0;

#### 5、获取网络事件类型

WSAEnumNetworkEvents函数，该函数原型为：

int WSAEnumNetworkEvents(

\_In\_   SOCKET s,

\_In\_   WSAEVENT hEventObject,

\_Out\_  LPWSANETWORKEVENTS lpNetworkEvents

);

其中结构体WSANETWORKEVENTS定义如下：

typedef struct \_WSANETWORKEVENTS {

long lNetworkEvents;

int  iErrorCode[FD\_MAX\_EVENTS];

} WSANETWORKEVENTS, \*LPWSANETWORKEVENTS;

第一个元素为事件类型，第二个参数为错误码，当发生网络事件错误时，相应的错误码会标识错误码信息，其序号为FD\_XXXX\_BIT。

判断事件类型的程序代码片段如下：

WSANETWORKEVENTS event;

WSAEnumNetworkEvents(SocketArray[nIndex],EventArray[nIndex],&event);

if (event.lNetworkEvents & FD\_ACCEPT)

{

if (event.iErrorCode[FD\_ACCEPT\_BIT]==0)

{

//处理连接请求

}

}

if (event.lNetworkEvents & FD\_READ)

{

if (event.iErrorCode[FD\_READ\_BIT]==0)

{

//处理读事件

}

}

if (event.lNetworkEvents & FD\_CLOSE)

{

if (event.iErrorCode[FD\_CLOSE\_BIT]==0)

{

//处理关闭事件

}

}

WSAResetEvent(EventArray[nIndex]);

## 二、代码

开启服务的线程：

UINT \_\_cdecl ThreadProc(LPVOID pParam)

{

CWSAEventTestDlg\* pDlg=(CWSAEventTestDlg\*)pParam;

//初始化套接字库

WSADATA data;

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

//创建套接字

pDlg->m\_ListenSock=INVALID\_SOCKET;

pDlg->m\_ListenSock=socket(AF\_INET,SOCK\_STREAM,IPPROTO\_TCP);

if (pDlg->m\_ListenSock==INVALID\_SOCKET)

{

AfxMessageBox(\_T("Create Socket Failed"));

WSACleanup();

return 0;

}

//绑定套接字

sockaddr\_in ServerAddr;

ServerAddr.sin\_family=AF\_INET;

ServerAddr.sin\_port=htons(9527);

ServerAddr.sin\_addr.s\_addr=INADDR\_ANY;

if (SOCKET\_ERROR==bind(pDlg->m\_ListenSock,(sockaddr\*)&ServerAddr,sizeof(sockaddr\_in)))

{

AfxMessageBox(\_T("Bind Socket Failed"));

closesocket(pDlg->m\_ListenSock);

WSACleanup();

return 0;

}

//设置监听状态

if (listen(pDlg->m\_ListenSock,5)==INVALID\_SOCKET)

{

AfxMessageBox(\_T("Listen Socket Failed"));

closesocket(pDlg->m\_ListenSock);

WSACleanup();

return 0;

}

//创建手动重置事件对象，关联套接字

WSAEVENT ListenEvent=::WSACreateEvent();

WSAEventSelect(pDlg->m\_ListenSock,ListenEvent,FD\_CLOSE|FD\_ACCEPT);

pDlg->EventArray[pDlg->nTotalEvent]=ListenEvent;

pDlg->SocketArray[pDlg->nTotalEvent]=pDlg->m\_ListenSock;

pDlg->nTotalEvent++;

//处理网络事件

while(TRUE)

{

int nIndex=WSAWaitForMultipleEvents(pDlg->nTotalEvent,

pDlg->EventArray,

FALSE,

WSA\_INFINITE,

FALSE);

if (nIndex==WSA\_WAIT\_FAILED||nIndex==WSA\_WAIT\_TIMEOUT)

{

continue;

}

nIndex=nIndex-WSA\_WAIT\_EVENT\_0;

//判断序号为nIndex事件的事件类型

WSANETWORKEVENTS event;

WSAEnumNetworkEvents(pDlg->SocketArray[nIndex],pDlg->EventArray[nIndex],&event);

if (event.lNetworkEvents & FD\_ACCEPT)

{

//接到新的连接请求

if (event.iErrorCode[FD\_ACCEPT\_BIT]==0)

{

WSAEVENT NewEvent=::WSACreateEvent();

sockaddr\_in ClientAddr;

int nLen=sizeof(sockaddr\_in);

SOCKET NewSocket=accept(pDlg->SocketArray[nIndex],(sockaddr\*)&ClientAddr,&nLen);

WSAEventSelect(NewSocket,NewEvent,FD\_READ|FD\_CLOSE);

pDlg->SocketArray[pDlg->nTotalEvent]=NewSocket;

pDlg->EventArray[pDlg->nTotalEvent]=NewEvent;

pDlg->nTotalEvent++;

}

}

if (event.lNetworkEvents & FD\_READ)

{

//有网络读事件

if (event.iErrorCode[FD\_READ\_BIT]==0)

{

TCHAR szBuf[1000]={0};

int iRet=recv(pDlg->SocketArray[nIndex],szBuf,1000,0);

if (iRet>0)

{

pDlg->SetDlgItemText(IDC\_EDIT\_RECV,szBuf);

}

}

}

if (event.lNetworkEvents & FD\_CLOSE)

{

//网络关闭事件

if (event.iErrorCode[FD\_CLOSE\_BIT]==0)

{

closesocket(pDlg->SocketArray[nIndex]);

for (int i=nIndex;i<pDlg->nTotalEvent-1;i++)

{

pDlg->EventArray[i]=pDlg->EventArray[i+1];

pDlg->SocketArray[i]=pDlg->SocketArray[i+1];

}

pDlg->nTotalEvent--;

}

}

WSAResetEvent(pDlg->EventArray[nIndex]);

}

}