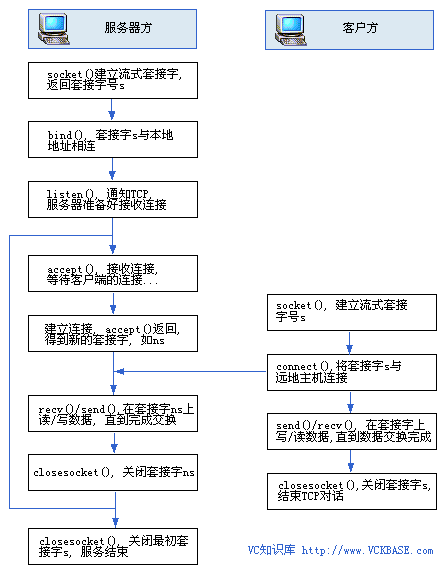
# WinSock IO模型之Select

## 一、套接字编程（流式套接字）



## 客户端流程：

客户端只需要一个SOCKET

### 1、加载套接字库

这一步比较固定，使用以下代码即可：

BOOL CClientDlg::WinSockInit()

{

WSADATA data = {0};

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

return FALSE;

if ( LOBYTE(data.wVersion) !=2 || HIBYTE(data.wVersion) != 2 ){

WSACleanup();

return FALSE;

}

return TRUE;

}

### 2、创建套接字

使用函数socket，原型：

SOCKET socket(

int af,

int type,

int protocol

);

对于流式套接字，使用以下代码即可：

m\_Socket = socket(AF\_INET , SOCK\_STREAM , IPPROTO\_TCP);

### 3、向服务器发出连接请求

使用connect函数，函数原型：

int connect(

SOCKET s,

const struct sockaddr FAR\* name,

int namelen

);

该函数将套接字连接到服务器端，因此需要服务器端的地址，地址存储在结构体sockaddr\_in中，在连接前需要为该结构体赋值：

sockaddr\_in server;

server.sin\_family = AF\_INET;

server.sin\_port = htons(9527);//端口

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");//IP地址

连接服务器：

connect(pThis->m\_Socket, (struct sockaddr \*)&server, sizeof(struct sockaddr)

该函数失败返回SOCKET\_ERROR；

### 4、接收和发送数据

接收数据recv函数

recv(m\_Socket, (char \*)szBuf, MAX\_BUF\_SIZE, 0);

发送数据send函数

send(m\_Socket, (char \*)strText.GetBuffer(), strText.GetLength()\*sizeof(TCHAR), 0);

### 5、关闭套接字

Closesoket函数

### 6、卸载WINSOCK库

WSACleanup函数

## 服务器端流程：

服务器端需要两个SOCKET，一个用于监听，一个用于通信

### 1、加载套接字库

### 2、创建套接字

只需创建监听套接字：

m\_SockListen = socket(AF\_INET , SOCK\_STREAM , IPPROTO\_TCP);

### 3、绑定套接字到本地地址和端口

这里同样需要一个地址结构体sockaddr\_in，绑定函数是bind。

sockaddr\_in service;

service.sin\_family = AF\_INET;

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

service.sin\_port = htons(9527);

bind(m\_SockListen, (sockaddr\*)&service, sizeof(sockaddr\_in))

### 4、将套接字设定为监听模式，准备接收数据

使用函数listen，函数原型：

int listen(

SOCKET s,

int backlog

);

代码如下：

listen(m\_SockListen, SOMAXCONN)

### 5、等待客户请求

函数accept，原型：

SOCKET accept(

SOCKET s,

struct sockaddr FAR\* addr,

int FAR\* addrlen

);

如果有客户请求到达，该函数会返回一个新的SOCKET，该SOCKET用于与客户端通信，包括接收和发送数据。参数addr用于接收客户端的地址信息。

m\_SockClient = accept(pThis->m\_SockListen, (struct sockaddr \*)&clientAddr , &iLen);

### 6、接收和发送数据

接收recv，通过accept返回的套接字

recv(m\_SockClient, (char \*)szBuf, MAX\_BUF\_SIZE, 0)

发送send，通过accept返回的套接字

send(m\_SockClient, (char\*)strText.GetBuffer(), strText.GetLength()\*sizeof(TCHAR), 0);

### 7、关闭套接字

### 8、卸载WINSOCK库

## 二、select函数

选择（select）模型是Winsock中最常见的 I/O模型。核心便是利用 select 函数，实现对 I/O的管理！

利用 select 函数来判断某Socket上是否有数据可读，或者能否向一个套接字写入数据，防止程序在Socket处于阻塞模式中时，

在一次 I/O 调用（如send或recv、accept等）过程中，被迫进入“锁定”状态；同时防止在套接字处于非阻塞模

式中时，产生WSAEWOULDBLOCK错误。

█ select 的函数原型如下：

int select(

\_\_in int nfds,

\_\_in\_out fd\_set\* readfds,

\_\_in\_out fd\_set\* writefds,

\_\_in\_out fd\_set\* exceptfds,

\_\_in const struct timeval\* timeout

);

其中，第一个参数nfds会被忽略。之所以仍然要提供这个参数，只是为了保持与Berkeley套接字兼容。

后面大家看到有三个 fd\_set类型的参数：

一个用于检查可读性（readfds），

一个用于检查可写性（writefds），

一个用于例外数据（exceptfds）。

fd\_set 结构的定义如下：

typedef struct fd\_set {

u\_int fd\_count;

SOCKET fd\_array[FD\_SETSIZE];

} fd\_set;

#define FD\_SETSIZE 64

所以 fd\_set 结构中最多只能监视64个套接字。

fdset 代表着一系列特定套接字的集合。其中， readfds 集合包括符合下述任何一个条件的套接字：

● 有数据可以读入。

● 连接已经关闭、重设或中止。

● 假如已调用了listen，而且一个连接正在建立，那么accept函数调用会成功。

writefds 集合包括符合下述任何一个条件的套接字：

● 有数据可以发出。

● 如果已完成了对一个非锁定连接调用的处理，连接就会成功。

exceptfds 集合包括符合下述任何一个条件的套接字：

● 假如已完成了对一个非锁定连接调用的处理，连接尝试就会失败。

● 有带外（Out-of-band，OOB）数据可供读取。

举个例子，假设我们想测试一个套接字是否“可读”，必须将自己的套接字增添到readfds集合中，

然后调用 select 函数并等待其完成。select 完成之后，再次判断自己的套接字是否仍为 readfds 集合的一部分。

若答案是肯定的，则表明该套接字“可读”，可立即着手从它上面读取数据。

在三个参数中（readfds、writefds 和 exceptfds），任何两个都可以是空值（ NULL）；

但是，至少有一个不能为空值！在任何不为空的集合中，必须包含至少一个套接字句柄；

否则， select 函数便没有任何东西可以等待。最后一个参数 timeout 对应的是一个指针，它指向一个timeval 结构，

用于决定select 最多等待 I/O操作完成多久的时间。如 timeout 是一个空指针，那么 select 调用会无限

期地“锁定”或停顿下去，直到至少有一个描述符符合指定的条件后结束。

对 timeval 结构的定义如下：

tv\_sec 字段以秒为单位指定等待时间；

tv\_usec 字段则以毫秒为单位指定等待时间。

1秒 = 1000毫秒

若将超时值设置为（0 , 0），表明 select 会立即返回，出于对性能方面的考虑，应避免这样的设置。

█ select 函数返回值：

select 成功完成后，会在 fdset 结构中，返回刚好有未完成的 I/O操作的所有套接字句柄的总量。

若超过 timeval 设定的时间，便会返回0。若 select 调用失败，都会返回 SOCKET\_ERROR，

应该调用 WSAGetLastError 获取错误码！

用 select 对套接字进行监视之前，必须将套接字句柄分配给一个fdset的结构集合，

之后再来调用 select，便可知道一个套接字上是否正在发生上述的 I/O 活动。

Winsock 提供了下列宏操作，可用来针对 I/O活动，对 fdset 进行处理与检查：

● FD\_CLR(s, \*set)：从set中删除套接字s。

● FD\_ISSET(s, \*set)：检查s是否set集合的一名成员；如答案是肯定的是，则返回TRUE。

● FD\_SET(s, \*set)：将套接字s加入集合set。

● FD\_ZERO( \* set)：将set初始化成空集合。

例如，假定我们想知道是否可从一个套接字中安全地读取数据，同时不会陷于无休止的

“锁定”状态，便可使用 FDSET 宏，将自己的套接字分配给 fdread 集合，再来调用 select。要

想检测自己的套接字是否仍属 fdread 集合的一部分，可使用 FD\_ISSET 宏。采用下述步骤，便

可完成用 select 操作一个或多个套接字句柄的全过程：

1) 使用FDZERO宏，初始化一个fdset对象；

2) 使用FDSET宏，将套接字句柄加入到fdset集合中；

3) 调用 select 函数，等待其返回……select 完成后，会返回在所有 fdset 集合中设置的套接字句柄总数，

并对每个集合进行相应的更新。

4) 根据 select的返回值和 FDISSET宏，对 fdset 集合进行检查。

5) 知道了每个集合中“待决”的 I/O操作之后，对 I/O进行处理，

然后返回步骤1 )，继续进行 select 处理。

select 函数返回后，会修改 fdset 结构，删除那些不存在待决 I/O 操作的套接字句柄。

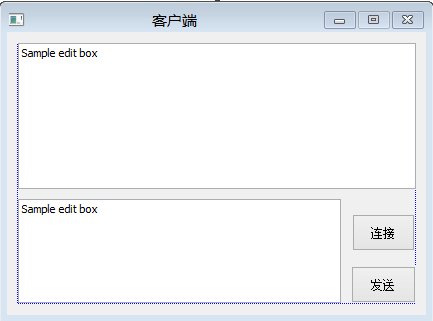
这正是我们在上述的步骤 ( 4 ) 中，为何要使用 FDISSET 宏来判断一个特定的套接字是否仍在集合中的原因。

**注意，select函数是阻塞式的，因此需要在while循环中不断用select检查可读可写性。**

## 三、实例程序

### 3.1客户端

#### 界面：



#### 包含文件：

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdio.h>

// Need to link with Ws2\_32.lib

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

#### 对话框类H文件：

// ClientDlg.h : header file

//

#pragma once

// CClientDlg dialog

class CClientDlg : public CDialog

{

// Construction

public:

CClientDlg(CWnd\* pParent = NULL); // standard constructor

// Dialog Data

enum { IDD = IDD\_CLIENT\_DIALOG };

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

// Implementation

protected:

HICON m\_hIcon;

// Generated message map functions

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

DECLARE\_MESSAGE\_MAP()

public:

SOCKET m\_Socket;

void ShowMsg(CString strMsg);

BOOL WinSockInit();

BOOL SOCKET\_Select(SOCKET hSocket, int nTimeOut = 100, BOOL bRead = TRUE);

static UINT \_\_cdecl ThreadProc(LPVOID pParam);

afx\_msg void OnBnClickedBtnConnect();

afx\_msg void OnBnClickedBtnSend();

};

#### 对话框类CPP文件主要代码：

#include "stdafx.h"

#include "Client.h"

#include "ClientDlg.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

CClientDlg::CClientDlg(CWnd\* pParent /\*=NULL\*/)

: CDialog(CClientDlg::IDD, pParent)

{

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

m\_Socket = INVALID\_SOCKET;

}

BOOL CClientDlg::WinSockInit()

{

WSADATA data = {0};

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

return FALSE;

if ( LOBYTE(data.wVersion) !=2 || HIBYTE(data.wVersion) != 2 ){

WSACleanup();

return FALSE;

}

return TRUE;

}

BOOL CClientDlg::SOCKET\_Select(SOCKET hSocket, int nTimeOut, BOOL bRead)

{

fd\_set fdset;

timeval tv;

FD\_ZERO(&fdset);

FD\_SET(hSocket, &fdset);

nTimeOut = nTimeOut > 1000 ? 1000 : nTimeOut;

tv.tv\_sec = 0;

tv.tv\_usec = nTimeOut;

int iRet = 0;

if ( bRead ) {

iRet = select(0, &fdset, NULL , NULL, &tv);

}else{

iRet = select(0, NULL , &fdset, NULL, &tv);

}

if(iRet <= 0) {

return FALSE;

} else if (FD\_ISSET(hSocket, &fdset)){

return TRUE;

}

return FALSE;

}

#define MAX\_BUF\_SIZE 4096

UINT CClientDlg::ThreadProc(LPVOID pParam)

{

ASSERT(pParam);

CClientDlg \*pThis = (CClientDlg \*)pParam;

pThis->WinSockInit();

pThis->m\_Socket = INVALID\_SOCKET;

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

if ( pThis->m\_Socket == INVALID\_SOCKET ) {

AfxMessageBox(\_T("新建Socket失败！"));

goto \_\_Error\_End;

}

sockaddr\_in server;

server.sin\_family = AF\_INET;

server.sin\_port = htons(9527);

server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if ( connect(pThis->m\_Socket, (struct sockaddr \*)&server, sizeof(struct sockaddr)) == SOCKET\_ERROR ) {

AfxMessageBox(\_T("连接失败，请重试！"));

goto \_\_Error\_End;

}

while(TRUE) {

if ( pThis->SOCKET\_Select(pThis->m\_Socket) ) {

TCHAR szBuf[MAX\_BUF\_SIZE] = {0};

int iRet = recv(pThis->m\_Socket, (char \*)szBuf, MAX\_BUF\_SIZE, 0);

if ( iRet > 0 ) {

//right;

pThis->ShowMsg(szBuf);

}else{

//close socket;

pThis->ShowMsg(\_T("聊天室服务器已停止，请重新进行连接！"));

break;

}

}

Sleep(500);

}

\_\_Error\_End:

if (pThis->m\_Socket != INVALID\_SOCKET) {

closesocket(pThis->m\_Socket);

}

WSACleanup();

return TRUE;

}

void CClientDlg::ShowMsg(CString strMsg)

{

CEdit \*pEditShow = (CEdit \*)GetDlgItem(IDC\_EDIT\_SHOW);

ASSERT(pEditShow);

pEditShow->SetSel(-1, -1);

pEditShow->ReplaceSel(\_T("Server : ") + strMsg+\_T("\r\n"));

}

void CClientDlg::OnBnClickedBtnConnect()

{

AfxBeginThread(ThreadProc, this);

}

void CClientDlg::OnBnClickedBtnSend()

{

CString strText;

ASSERT(m\_Socket != INVALID\_SOCKET);

GetDlgItemText(IDC\_EDIT\_WRITE, strText);

if (!strText.IsEmpty() && SOCKET\_Select(m\_Socket, 100, FALSE)) {

send(m\_Socket, (char \*)strText.GetBuffer(), strText.GetLength()\*sizeof(TCHAR), 0);

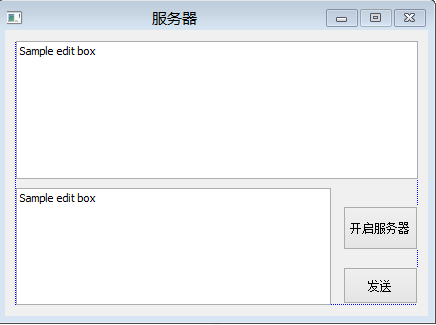
SetDlgItemText(IDC\_EDIT\_WRITE, \_T(""));

}

}

### 3.2 服务器端

#### 界面



#### 包含文件：

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdio.h>

// Need to link with Ws2\_32.lib

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

#### 对话框类H文件：

#pragma once

// CServerDlg dialog

class CServerDlg : public CDialog

{

// Construction

public:

CServerDlg(CWnd\* pParent = NULL); // standard constructor

// Dialog Data

enum { IDD = IDD\_SERVER\_DIALOG };

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV support

// Implementation

protected:

HICON m\_hIcon;

// Generated message map functions

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

DECLARE\_MESSAGE\_MAP()

public:

SOCKET m\_SockListen;

SOCKET m\_SockClient;

BOOL WinSockInit();

void ShowMsg(CString strMsg);

BOOL SOCKET\_Select(SOCKET hSocket, int nTimeOut = 100, BOOL bRead = TRUE);

static UINT \_\_cdecl ThreadProc(LPVOID pParam);

afx\_msg void OnBnClickedBtnStart();

afx\_msg void OnBnClickedBtnSend();

};

#### 对话框类CPP文件主要代码：

#include "stdafx.h"

#include "Server.h"

#include "ServerDlg.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

CServerDlg::CServerDlg(CWnd\* pParent /\*=NULL\*/)

: CDialog(CServerDlg::IDD, pParent)

{

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

m\_SockListen = INVALID\_SOCKET;

m\_SockClient = INVALID\_SOCKET;

}

BOOL CServerDlg::SOCKET\_Select(SOCKET hSocket, int nTimeOut, BOOL bRead)

{

fd\_set fdset;

timeval tv;

FD\_ZERO(&fdset);

FD\_SET(hSocket, &fdset);

nTimeOut = nTimeOut > 1000 ? 1000 : nTimeOut;

tv.tv\_sec = 0;

tv.tv\_usec = nTimeOut;

int iRet = 0;

if ( bRead ) {

iRet = select(0, &fdset, NULL , NULL, &tv);

}else{

iRet = select(0, NULL , &fdset, NULL, &tv);

}

if(iRet <= 0) {

return FALSE;

} else if (FD\_ISSET(hSocket, &fdset)){

return TRUE;

}

return FALSE;

}

#define MAX\_BUF\_SIZE 4096

UINT CServerDlg::ThreadProc(LPVOID pParam)

{

ASSERT(pParam);

sockaddr\_in clientAddr = {0};

CServerDlg \*pThis = (CServerDlg \*)pParam;

pThis->WinSockInit();

pThis->m\_SockListen = INVALID\_SOCKET;

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

if ( pThis->m\_SockListen == INVALID\_SOCKET ) {

AfxMessageBox(\_T("新建Socket失败！"));

goto \_\_Error\_End;

}

sockaddr\_in service;

service.sin\_family = AF\_INET;

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

service.sin\_port = htons(9527);

if ( bind(pThis->m\_SockListen, (sockaddr\*)&service, sizeof(sockaddr\_in)) == SOCKET\_ERROR ) {

AfxMessageBox(\_T("绑定端口失败！"));

goto \_\_Error\_End;

}

if( listen(pThis->m\_SockListen, SOMAXCONN) == SOCKET\_ERROR ) {

AfxMessageBox(\_T("监听失败！"));

goto \_\_Error\_End;

}

int iLen = sizeof(sockaddr\_in);

pThis->m\_SockClient = accept(pThis->m\_SockListen, (struct sockaddr \*)&clientAddr , &iLen);

if (pThis->m\_SockClient == INVALID\_SOCKET) {

goto \_\_Error\_End;

}

while(TRUE) {

if ( pThis->SOCKET\_Select(pThis->m\_SockClient) ) {

TCHAR szBuf[MAX\_BUF\_SIZE] = {0};

int iRet = recv(pThis->m\_SockClient, (char \*)szBuf, MAX\_BUF\_SIZE, 0);

if ( iRet > 0 ) {

//right;

pThis->ShowMsg(szBuf);

}else{

//close socket;

pThis->ShowMsg(\_T("客户端已下线，请重新开启服务器等待客户端连接！"));

break;

}

}

Sleep(500);

}

\_\_Error\_End:

if (pThis->m\_SockListen != INVALID\_SOCKET) {

closesocket(pThis->m\_SockListen);

}

if (pThis->m\_SockClient != INVALID\_SOCKET) {

closesocket(pThis->m\_SockClient);

}

WSACleanup();

return TRUE;

}

void CServerDlg::ShowMsg(CString strMsg)

{

CEdit \*pEditShow = (CEdit \*)GetDlgItem(IDC\_EDIT\_SHOW);

ASSERT(pEditShow);

pEditShow->SetSel(-1, -1);

pEditShow->ReplaceSel(\_T("Client : ") + strMsg+\_T("\r\n"));

}

void CServerDlg::OnBnClickedBtnStart()

{

AfxBeginThread(ThreadProc, this);

}

void CServerDlg::OnBnClickedBtnSend()

{

CString strText;

ASSERT(m\_SockClient != INVALID\_SOCKET);

GetDlgItemText(IDC\_EDIT\_WRITE, strText);

if (!strText.IsEmpty() && SOCKET\_Select(m\_SockClient, 100, FALSE)) {

send(m\_SockClient, (char \*)strText.GetBuffer(), strText.GetLength()\*sizeof(TCHAR), 0);

SetDlgItemText(IDC\_EDIT\_WRITE, \_T(""));

}

}