驱动开发之五 --- TDI之一（飞雪楚狂人）

转载

trustnature 于 2012-08-09 23:22:12 发布 1502 收藏
分类专栏： driver


driver 专栏收录该内容
13 篇文章 0 订阅
订阅专栏

驱动开发之五 --- TDI之一

转载自看雪论坛驱动专题

本篇的题目有点容易让人误解。出于演示目的，我们写一个TDI客户端，但这并不是我们本篇的目的所在。本篇意在进一步探究怎样处理irp和怎样与之交互。本篇将会介绍怎样排队和处理要取消的IRP. 本篇真正的题目应该是“IRP处理介绍“，然而它不是一个非常引人注意的名字。并且我们的这个题目并不完全是一个谎言，因为我们会把TDI客户端驱动作为一个演示实现。所以我们不得不介绍一下这部分的实现。这个例子是一个非常简单的客户端/服务器通讯的程序，我们将通过它来揭示IRP的处理。

套接字回顾

我们开始介绍的部分或许你已经很清楚，如果你不太了解的话，请阅读相关的文章。即便这样，我也提供了一个根据例子源程序来快速的回顾下如何实现套接字。

什么是IP?

IP或者“互联网协议”的本质是在两台机器之间用来发送数据或者包的协议。这个协议不需要任何的设置，仅仅需要网络上的每台机器有一个唯一的“IP地址“。这个“IP地址“被用来在通讯终端之间路由数据包。这个协议提供了路由功能，但是没有可靠性。仅仅依赖IP来发送数据包，会造成数据包被破坏、颠倒次序和不完整。然而在IP之上的其他协议弥补了这些缺陷。IP层位于OSI模型中的网络层。

什么是TCP?

TCP是指“传输控制协议“。它位于”IP“协议的上层。因此，我们通常讲的TCP/IP就是指的他们两个。”IP“层提供路由功能，”TCP“层提供可靠的，顺序无错的数据传送。为了区分机器上的多个TCP传输，每个TCP提供了一个唯一的TCP端口号。在这种方式下，多个应用程序或者甚至是同一个应用程序都可以打开一个通讯通路，底层的通讯可以在各个终端之间正确的路由数据。”TCP“协议为与OSI模型的传输层。后面，还有像FTP,HTTP等等的协议位于tcp协议的上层。这些协议位于OSI模型的应用层。

协议层

从某种意义上讲，协议栈中的任何部分都可以被一个等价的协议替换。例如，如果FTP需要可靠的传输和路由，那么把它放到能够提供这些功能的协议的上层，它仍然可以工作。在那个例子里，如果一个应用程序使用”SPX”代替“TCP/IP”，他不会产生不同。换句话说，如果“TCP”或者一些”TCP”的实现位于像”ipx“这样不可靠的协议的上层，它也可以工作。可以工作的原因很明显是取决于上层协议对于实际执行和底层协议的内部工作的依赖程度。

什么是套接字？

套接字通常是指被一个套接字库实现的通讯终端。套接字库API是实现用户模式应用程序最简单的办法。Socket API种类很少，我们在windows下使用”WINSOCK”.有一些WINSOCK是兼容的（我曾经实现了一个winsock应用，可以在unix和windows nt下编译，只有很小的冲突，当然这个程序也很简单）还有一些不能直接兼容的。

Socket服务程序

Socket连接的服务端仅仅用于接受进来的连接。每一个连接都有一个独立的句柄，服务程序可以单独的跟每一个客户端进行通讯。下面是概述了通讯中的步骤。

第一步：创建套接字

第一步创建socket. 下面的代码演示了如何创建一个流式socket (TCP/IP)。

```
hSocket = socket(PF_INET, SOCK_STREAM, 0);

if(hSocket == INVALID_SOCKET)

{

  /* Error */

}
```

然后它仅仅返回网络驱动一个句柄，你可以在其他的socket API中使用这个句柄。

第二步: 绑定套接字

第二步是绑定socket到一个TCP/IP端口和IP地址。下面的代码演示了这个过程。例子中我们创建的套接字，直接使用了一个常数作为套接字端口，然而实际上你应该使用一个宏，把它转换成网络字节顺序。

```
SockAddr.sin_family = PF_INET;

SockAddr.sin_port = htons(4000);      /* Must be in NETWORK BYTE ORDER */

/*

* BIND the Socket to a Port

*/

uiErrorStatus =

      bind(hSocket, (struct sockaddr *)&SockAddr, sizeof(SOCKADDR_IN));

if(uiErrorStatus == INVALID_SOCKET)

{

  /* Error */

}
```

这个操作把socket句柄和端口地址绑在了一起。你也可以把IP地址设置为0，让驱动程序绑定任何的IP地址（本地的一个）。你也可以把端口地址设为0来绑定一个随机的端口。通常，服务端都使用固定的端口号，这样方便客户端找到他们，但也有例外。

第三步：侦听

把套接字设置在侦听状态，套接字将能够在有连接呼叫后侦听到连接。参数后面的常数指明了同一时刻套接字允许接受的连接请求数。

```
if(listen(hSocket, 5) != 0)

{
```

```
  /* Error */
}
```

第四步: 接受连接

accept API给每一个接收进来的连接提供给你一个新的句柄。下面的代码使用accept的例子。

```
if((hNewClient = accept(pServerInfo->hServerSocket,
    (struct sockaddr *)&NewClientSockAddr, &uiLength)) != INVALID_SOCKET)
{
```

返回的句柄可以用来收发数据。

第五步：关闭套接字

当你用完时，你需要关闭所有的句柄。

```
closesocket(hNewClient);
```

使用select api在连接建立和数据到达时可以获得通知，在这里省略了这些细节。如果想进一步了解这些细节，你可以参考MSDN.

SOCKET客户端应用

Socket客户端用来同服务端建立连接和传送数据。下面的步骤说明了两者怎样进行通讯。

第一步：创建套接字

第一步创建套接字。下面的代码演示了如何创建一个流式套接字（TCP/IP）

```
hSocket = socket(PF_INET, SOCK_STREAM, 0);
if(hSocket == INVALID_SOCKET)
{
  /* Error */
}
```

然后它仅仅返回网络驱动一个句柄。我们在其它的socket api中将使用这个句柄。

第二步：连接到服务端

你需要设置服务端的地址和端口号（他们必须是网络字节顺序）来连接到服务端。然后调用connect API建立客户端与服务端的连接。

```
pClientConnectInfo->SockHostAddress.sin_family = PF_INET;
pClientConnectInfo->SockHostAddress.sin_port   =
            htons(4000);     /* Network Byte Order! */
printf("Enter Host IP Address like: 127.0.0.1\n");
fgets(szHostName, 100, stdin);
pClientConnectInfo->SockHostAddress.sin_addr.s_addr =
```

```
        inet_addr(szHostName);     /* Network Byte Order! */
iRetVal =
connect(hSocket, (LPSOCKADDR)&pClientConnectInfo->SockHostAddress,
                                    sizeof(SOCKADDR_IN));
if(iRetVal == INVALID_SOCKET)
{
   /* Error */
}
```

第三步：收发数据

一旦你建立连接，你就可以在你需要的时候使用recv和send api 进行数据传输。

```
iRetVal = send(hSocket, szBuffer, strlen(szBuffer), 0);
if(iRetVal == SOCKET_ERROR)
{
/* Error */
}
iRetVal = recv(hSocket, szBuffer, 1000, 0);
if(iRetVal == 0 || iRetVal == SOCKET_ERROR)
{
/* Error */
}
```

请注意，在这里的例子中我们收发的数据是字符串，当然也可以是任意的二进制数据。

第四步：关闭socket

当你使用完后，你需要关闭所有的句柄。

```
closesocket(hSocket);
```

使用select api在连接建立和数据到达时可以获得通知，在这里省略了这些细节。如果想进一步了解这些细节，你可以参考MSDN