Exploring Socket Programming

Jeremy.chiu@msa.hinet.net

June 26, 2001

本文尚未完成,僅供中央機械網管使用...!!
要公開也等我寫完吧...G

Prefect

聽說,鳥人超強....

Who Should Read This Book ?

本書適合想要實作 Winsock 程式的 Programmer 閱讀。

這本書的原則是：

1. 以實作爲主，廢話哈啦很少☺
2. 章節內容以講解每章主題範例程式爲主，因此 Code 會佔很重份量
3. 討論實際設計 Winsock 程式容易遇到的問題(精選了一些 Winsock FAQ)

因爲本書設定爲 Winsock 主題，所以我已經假設正在讀本文的你具備下列 background :

已有寫 C 的能力(這是廢話吧)

已有 Win32 SDK Programming 經驗

最好的話對 Win32 系統機制有一些概念，例如 Process, Threads, Synchronous Mechanism, Event Driven, Overlapped I/O

IP, TCP/UDP 的基本概念

不過～還不熟也沒關係，有聽過就好。你也可以在看本書時，一邊複習其他書

Thanks

第 1 章 Introduction to Socket and TCP/IP

這開始的第一章主要是介紹 Socket 的概念，還有跟 Socket 有相關的 TCP/IP 知識背景，爲後面的幾章做一個暖身運動。

要寫好一個網路程式需要相當多的網路知識，否則只會幾個 Socket APIs 想要寫出有效率的網路程式是十分困難的，因爲許多的 Socket APIs 所具有的特性跟限制都與下層的 TCP/IP 有著密不可分的關係。網路的概念有了，寫起 Socket 程式自然是水到渠成。

1.1 Introduction to TCP/IP

要講起網路故事，還真的是長的說不完壓…(話如果要講透支，眼淚就撥不離壓…)

TCP/IP，它的全名是 Transmission Control Protocol/Internet Protocol，這個通訊協定是由 Internet Engineering Task Force 發展出來，而且被廣爲應用在網路各地電腦，舉凡 E-mail, http, ftp, BBS…等等的網路應用早已入侵我們的生活，而變得理所當然…(續待)

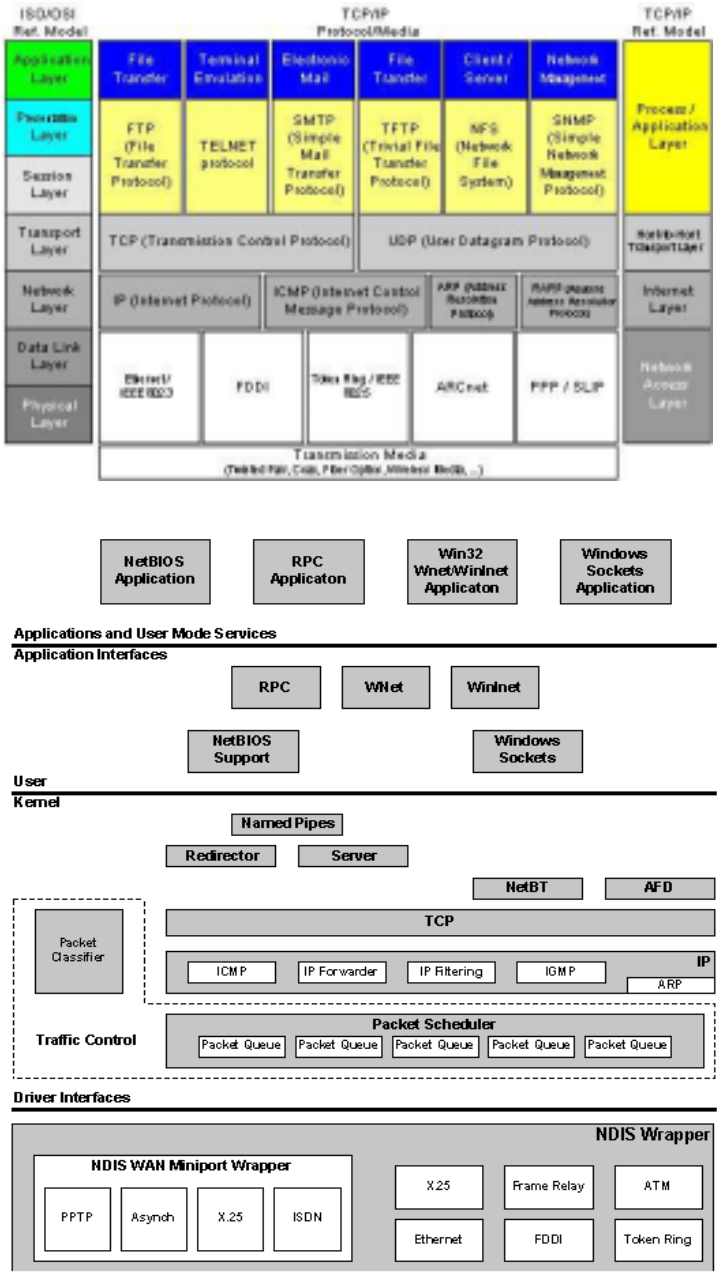
其中底層的 IP(Internet Protocol)通訊協定，主要的任務是定址網路上各個異質網路上的主機，給於編號，稱爲 IP Address (註 1)

而在 IP 層之上的便是 TCP 與 UDP。
TCP :
UDP :

1.2 OSI 架構與 TCP/IP 關係

OSI 即是 Open System Interconnect ，由 ISO 組織提出，OSI 把網路分層分成 7 層概念…

下圖是 OSI 網路模型跟 TCP/IP 之間的對應圖

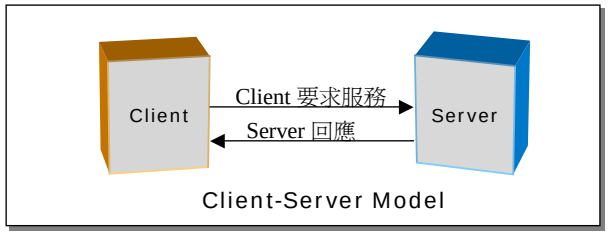


1.3 Unix 與 Berkeley Socket

啦啦啦.....鳥人萬歲 !

1.4 Client-Server Model

Client-Server 是一個很常見的網路程式工作模式。網路上的 Server 是一直等待 Client 來連上線來，並根據 Client 的要求給於服務，因此在這個架構中 Client 是扮演一個主動連線的角色，而 Server 則爲一個被動的服務提供者。



這樣的例子在我們常用的網路程式十分常見，例如 FTP Server 跟 FTP Client 的關係，Web Server 跟 Browser 程式的關係...等等。不過注意看一下，你會發現其實有很多網路程式，實際上通常扮演 Client, Server 兩個角色，因爲他們不但是在等 Client 連上來，而且也會去連其他的 Server。

第 2 章 BSD Socket Programming

本章主要講的是“標準”的 BSD Socket 程式，畢竟 Windows 上的 Socket 界面就是從 BSD Socket 改良過去的，使之能夠跟 Win32 系統的一些特性配合，如 Event Driven, Overlapped I/O, Threads...。因此，爲了簡化複雜度，讓我們能夠把重點放在 Socket Programming 上，所以先將 platform 鎖定在 Unix 上，以 BSD Socket 爲主，藉此也可以讓各位看官在讀到後面幾章時，順便可以比較一下 Winsock 與 BSD Socket 同異之處。

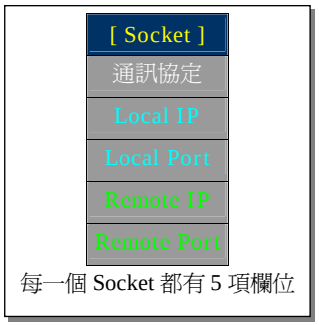
注意，本章的 Socket 是以 TCP 爲主！

2.1 What is the Socket like?

BSD 的把 TCP 跟 UDP 等 protocol 包裝成了 BSD Socket 界面，所以在 Programmer 眼中 Socket 不過是一堆網路函示中在用的 Descriptor 而已。但是，Socket 所表現的不只是個 Descriptor 而已，它還代表了一個連線(Session)的兩端所使用的窗口。藉由這個窗口(Socket)可以讓應用程式跟網路連線的另一端溝通。

我們已經知道了，Socket 包裝了 TCP 和 UDP，既然有 TCP/UDP，那就表示它至少含有兩項資訊：Port Number 跟 IP Address。因爲 TCP/UDP 屬於 IP 通訊協定，故必然有 IP Address，再者因爲到了 TCP/UDP 層，所以也會有 Port 等兩項資訊在一個 Socket 中。

下圖是 Socket 結構意示圖



由上圖，我們可以知道，既然有這 5 個欄位，所以透過這一個建立好的

Socket，程式可以將資料送到連線的另一端電腦，也可以從這個 Socket 接收對方送來的資料。

但是要建立一個 Socket，在 Client 及 Server 程式中並不一樣，所以兩者程式程式的寫法也就不一樣，下一章會就 Client 及 Server 的寫法做一個詳細介紹。

2.2 概觀 Client , Server 程式

上一節，提到了 Socket 所扮演的角色以及它的擁有的資料結構，在這一節中，我要開始真正的產生並建立一個 Socket（以 Stream Socket），並且帶各位看倌來瀏覽一遍 Client-Server 程式的結構。

講了這麼多，都還沒介紹到 Socket 的函示呢！

一個基本 Server 程式會用到的 Socket APIs 有 `socket()`, `bind()`, `listen()`, `accept()`, `recv()`, `send()`, `close()` 等 Socket 函示，其中 `listen()` 跟 `accept()` 是 Server 程式才會有的，而剩下的函示則 Client 及 Server 都有。

1. `socket()` 產生一個新的 socket
2. `bind()` 爲新的 socket 指定 local IP 及 Port
3. `listen()` 使 socket 進入等待 Client 連接要求狀態
4. `accept()` 接受 Client 連線要求。建立 scoket 欄位中的 remote IP,Port
5. `recv()` 接收對方資料
6. `send()` 傳送資料給對方
7. `close()` 關閉 Socket

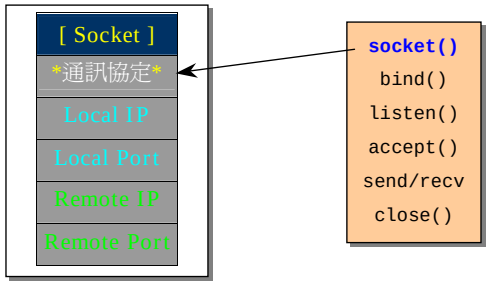
接下來是 Client 程式中會用到的 Socket APIs 有 `socket()`, `bind()`, `connect()`, `recv()`, `send()`, `close()`。不過在 Client 程式裡，是可以不用 `bind()`這個函示的。因爲會使用到的 Socket APIs 比較少，引此總體而言 Client 程式是比較好寫的。

讓我們先來看看 Server 的部份，一一介紹每個 Socket 函示的用法，還有注意的地方。

1. 建立一個 Socket

```
int socket( int domain, int type, int protocol );
```

這是第一個 Socket 函示，用它來產生往後要通訊用的 Socket Descriptor。



呼叫 `socket()`函式後，產生一個新的 Socket，並且指定 “通訊協定”

`socket()`的第一個參數 domain 是指定要使用的的網域形態或是位址形態，在整篇文章裡全都是 AF_INET 也就是普通 Internet 的位址。
第二個參數 type 是指定要使用何種通訊方式，有兩種可以選擇

SOCKET_STREAM

這種通訊連接方式也稱爲 Virtual Circuit。Stream 對應的是 TCP 通訊協定，是一種 Connection-oriented style，提供了幾個性質 “Tow-way”, “reliable”, “sequenced”(FIFO), “full-duplex”。資料必須等到 Connection 建立後才可以傳送，一但建立必須等到 Disconnection 後才結束。

SOCKET_DGRAM

這種通訊方式又稱 Datagrams，是來傳送不連續的資料封包。Datagrams 跟 Stream 最大的不同就是它不保證資料能順序的送到對方主機，連且也不一定送的到(unreliable)，此外也不用建立 Connection。因此這種方式是對應 UDP 通訊協定。

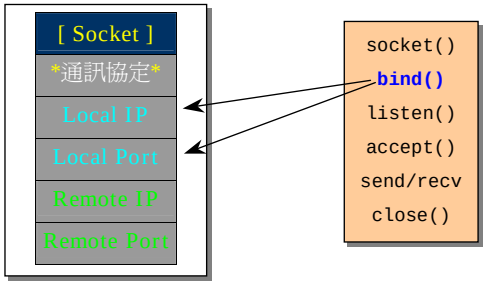
而第三個參數 protocol 是指 socket 所對應的傳輸協定編號。這個參數只要給 0 就可以，系統就會自動幫你設定。

傳回值：如果成功傳回 Socket Descriptor，爲非負數值。如果 `socket()` 失敗則傳回 -1，錯誤訊息在 `errno` 裡。(在 Winsock 中是以 `WSAGetLastError()`來取得錯誤代碼)。

2. 爲 Socket 命名

```
int bind(int socket, const struct sockaddr* name, int addrlen);
```

我們在第一個步驟中用 socket()產生了一個新的 Socket，但是裡面的 5 個欄位還是空的，尚未指定。因此，這裡的 bind()就是用來設定 socket



名稱。

bind()的功用是設定了 Socket 的本機 IP 跟 Port，注意，此時 Scoket 尚未建立完成，還有 Remote IP 跟 Port 未設定。如果本機的 Port 已經先被其他程式先使用了，那麼 bind()就會失敗。

bind()需要“sockaddr* name”這樣的一個參數來設定 Local IP 跟 Local

```
struct sockaddr{
    u_short sa_family;          // address family
    char sa_data[14];           // 位址內容（未定義）
}
```

Port，sockaddr 的結構如下

上面這個 sockaddr struct 只是一個位址結構的基本型態，是爲了擴充成各個協定之用，所以有一個 char sa_data[14]欄位是給其他協定自行定義。而在 Internet 中是用 AF_INET，因此，下面就是 AF_INET 用的 sockaddr 結構。

```
struct sockaddr_in{
    short    sin_family;            // 通常是 AF_INET
    u_short  sin_port;              // Port number
    struct   in_addr sin_addr       // IP address
    char sin_zero[8]                // reserved
}
```

在 BSD Socket 中是可以支援各種不同的通訊協定，因此在位址的定義上就有很多種，不只有 sockaddr_in。不過在 Winsock 1.1 中只支援 Internet Protocol，但是到了 Winsock 2 就支援不同的通訊協定，所以你有可能會看到像這樣的 ATM 位址結構 sockaddr_atm。

上述結構名稱是 sockaddr_in，因爲屬於 Internet 協定，所以才有 “in” 的字尾。要注意的是，在這個 Struct 中多定義了 2 項欄位，Port number 跟 IP Address，而在 IP Address 這項又是一個 Struct →in_addr，希望你不要眼花才好 ;p 。接下來就是 in_addr 的內容：

```
struct in_addr{
    union{
        struct{ u_char    s_b1,s_b2,s_b3,s_b4;} S_un_b;
        struct{ u_short   s_w1,s_w2;}  S_un_w;
        u_long   S_addr;
    } S_un;
}
#define s_addr    S_un.S_addr
```

這看起來更複雜了…不過你可以想成 in_addr 裡面就只有一個 u_long，一個 32 位元的數值來表示一個 IP。

拉拉雜雜地講了一堆結構，一堆位址，一定有人現在一團混亂…ccc。不過，只要看個例子，馬上會用 bind()！反正到頭來你只想透過 bind()來設定 Local IP 跟 Port。

```
struct sockaddr_in name;
name.sin_family         = AF_INET;
name.sin_port           = htons(5000);
name.sin_addr.s_addr    = inet_addr("140.115.65.1");
bind(sock,(struct sockaddr *)&name, sizeof (name) );
```

說明一下，htons()是將數字轉換成網路位元排列，而 inet_addr()則是將 IP 字串轉成 32-bit 無號整數，這兩個小函示在 Socket 程式中很好用。

如果程式不在乎 Local IP，可以在 name.sin_addr.s_addr 填入 INADDR_ANY。不在乎 Port 的話可以填 0。大部分情況，Server 程式在 bind()時都不需要特別指定 IP，所以給 INADDR_ANY 就可以。而且一種情況是我們所在的機器有多個 IP，用 INADDR_ANY 可以讓系統自動選擇。

3. 等待 Client 連線

```
int listen(int ServerSocket, int backlog);
```

listen()是使該 Socket 進入等待連線狀態，等待 Client 連上線來。如果有 Client 想要連過來，此時可以呼叫 connect()來跟 Server 連線。

而第二個參數 int backlog，是指在完成真正連線前(即 Server 未 accept()

Client 要求前)，所能接受的要求次數，並不是指說該 Server Socket 可以服務的 Client 數量喔，請特別注意。如果連線要求超過 backlog 值，此時要連線的 Client 呼叫 `connect()`後，會得到“connect refused”錯誤(在 Winsock 中是 WSAECONNREFUSED)。

如果要讓系統自己選擇目前可用的最大值，可以用 **SOMAXCONN**，一般來講 BSD Socket 預設最大值是 5（Winsock 1.1 的最大值也是 5）。

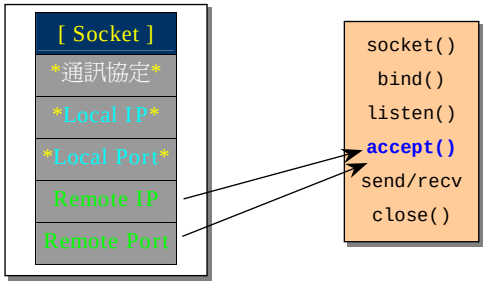
其實 `Listen()`這個動作，是對應建立 TCP 連線時的 three-way handshake。Client 會透過 `connect()`送出 SYN 給 Server，如果 TCP/IP Kernel 發現該 Socket 的 Connect Queue 還沒大於 backlog 時，表示此時還可以接受連線要求，因此，Kernel 會在回復一個 SYN 給 Client。
至於，一些連線要求頻繁的 Server(如 Web Server)，該如何決定適合 backlog 值，Richard Stevens 發表過一些測試，他建議一個 Web Server 的 backlog 可以用 16 或更多。
＊不過還得小心 SYN Attack，像是 SYN-Flood…(口述資料部份)

---

4. 接受連線

```
int accept(int SeverSocket)
```

當 `listen()`成功後，TCP/IP Kernel 會幫我們的 Socket 建立一個 Connect Queue(Queue 的長度爲 backlog)。而 `accept()`的作用就是從這 Queue 取出需求，並把 Client 的資料跟 Server Socket 資料組合成新的 Socket。喔～此時我們歷經千辛萬苦的連線終於完成！`accept()`成功後會傳回一個建立完的 Socket（5 項資料都完成），往後，就可以靠這個 Client Socket



來與對方傳輸資料。
在預設的 Socket I/O mode → Blocking mode，這個 `accept()`函示會停住，一直等到有 Client 連上來爲止，才會繼續下面的工作，這個特性很重要，詳細的部份會在 2.6 節，講到 Non-Blocking 實作一個比較。

一開始接觸 Socket 的人很容易搞混一件事情，就是 `accept()`完之後，會丟出一個新的 Socket，因此，他就有了兩個 Socket Descriptor 了，當需要呼叫 `send()/recv()`時，往往會不知道該傳入哪個 Socket 來作網路傳輸。不過，我想你只要記住一件事情就可以避開這種狀況，只有建立好連線的 Socket 才可以傳輸資料(Socket 5 項資料都完成)。而我們傳進去 accept()的那一個 ServeSocket 參數，是不可以拿來作 `send()/recv()`的，因爲那個 Socket 是代表 Server 在做 `listen()`的角色。

5. 傳輸資料

```
int recv( int socket, char* buf, int len, int flags );
int send( int socket, char* buf, int len, flags );
```

`accept()`做完了。Connection 也完成了，所以 Socket Descriptor 的五項欄位也已經填完，everything is ok，終於可以開始利用 Socket 來跟對方溝通了。

我們可以利用 `recv()`來接收資料，`send()`來傳送資料。這 BSD Socket 裡的跟 `recv(), send()`跟檔案的讀寫很類似，因爲在 Unix 中幾乎所有的東西都是 File，因此對 Socket 還可以用 `read(), write()`來作 Socket 的 I/O，不過 `recv(), send()` 多了第四個參數 `int flags`。

在一般情形下，是使用 Blocking mode 的 Socket，在這個模式下的 `send()/recv()`是除非工作做完，才會結束，也就是傳完或收完所指定的長度資料，程式才會繼續 run，不然就會一直停在那(Blocking)，一動也不動☺。

6. 關閉 Socket

```
int close(int socket);
```

這個動作跟關檔的意思一樣。用完就關也才不會浪費資源。(在 Winsock 中是用 `closesocket()`)

雖然是簡單的關檔動作，但是要程式要關的好，可就多需要一點努力了。問甚麼這樣說呢，絕大部分時候當你下 `close()`時，該 Socket 是並不會馬上關閉，因爲 Socket 關閉時會等到 Kernel 把資料送完才關閉該 Socket（會在背景執行該行爲）。因此會遇到一種情形，即使執行過了 `close()`但是 Socket 卻不會馬上關閉，要過一下才會關閉。其實，這跟 TCP 的 Sliding window 有關。

關閉一個 Socket 我們有兩種做法：

1. Graceful Close

就是先利用 shutdown()來關閉 Socket 的 I/O 能力，

```
int shutdown(int socket, int how);
```

其中的參數 int how 有 3 種 flags 可以設定

how = 0　關閉該 Socket 的接受能力。這個選向會強迫 TCP Kernel 關閉 Sliding Window。

how = 1　關閉該 Socket 的傳送能力。此時 TCP Kernel 會送出 FIN 給對方的電腦，以完成正常連線。

how = 2　關閉 Socket 的傳送及接受能力，等於前兩項加起的功用。

所以當你要很客氣的關閉一個 Connection 時，希望你是這麼做的：

1. Finish sending data.
2. Call `shutdown()` with the *how* parameter set to 1.
3. Loop on `recv()` until it returns 0.
4. Call `close()`.

如此一來，才可以避免資料在關閉實有遺失的機會。這是一個嚴謹而良好的習慣。

2. Force Close

雖然平常時學著客氣是不錯的選擇，不過我們寫程式時有時候希望對這個的機制做一個控制，畢竟，沒人知道到底多久會關閉。此時，另一個函示就派上用場了→ `setsocketopt()`這個函示適用來設定 Socket 的一些細部選項，詳細的部分在第五章會有介紹，這裡呢我們將 Socket 的 LINGER 設成 0，如此可以讓 `close()`強制性地關閉 Socket，而所有地資料爲做完 I/O 地將被放棄。對方也會都到 Connection Reset 的斷線訊息。

```
// Linger 設定範例
linger LingerOption;
LingerOption.l_onoff  = TRUE;
LingerOption.l_linger = 0;       //設定 TimeOut 秒數

setsockopt( hSocket ,SOL_SOCKET,SO_LINGER,
           (char*)&LingerOption,sizeof(linger));
```


Linger 值是指要等多少秒之後在強制關閉 Socket。

Server	Client
`sock = socket(AF_INET, SOCK_STREAM, 0);`	`sock = socket(AF_INET, SOCK_STREAM,0);`
`name.sin_family = AF_INET;`	`server.sin_family = AF_INET;`
`name.sin_addr.s_addr = INADDR_ANY;`	`hp = gethostbyname(argv[1]);`
`name.sin_port = 5000;`	`bcopy((char*)hp->h_addr,(char*)`
	`&server.sin_addr, hp->h_length);`
`bind(sock,(struct sockaddr *)&name, sizeof`	`server.sin_port = 5000;`
`(name) );`	
`listen(sock,5);`	
`msgsock=accept(sock,(struct ockaddr,0,0);`	`connect(sock, (struct sockaddr*)&server,`
	`sizeof(server))`
`rval = read(msgsock, buf, sizeof(buf));`	`wrval = write(sock, "Hello!",6 );`
`printf("The Message is %s \\n",buf);`	
`. . . .`	
`close(msgsock);`	
`close(sock);`	`close(sock);`

Client Server 程式結構比較圖

簡單的 Client-Server 程式大略的結構

當你程式是有使用到 socket 函示時，同時也必須含入 2 個檔頭。

```
#include <sys/types.h>
#include <sys/socket.h>
```

第 6 章 The Conslution

Reference

1. Winsock 網路程式設計之鑰, 1996 資訊人出版
2. UNIX 系統程式設計 (UNIX System Programming for SVR4), 1997 松格資訊出版
3. Linux Socket Programming, Que, April 2000
4. TCP/IP Illustrated, Volume I, W. Richard Stevens
5. Win32 多續程式設計 (Multithreading Application in Win32), 1997 碁峰資訊出版
6. Winsock Programmer's FAQ , Warren Young , 26 March 2001
7. Advanced Windows 3rd, Jeff Richter
8. MSDN – “ Windows Sockets 2 ” Section