網路程式設計（WinSock）

一、簡介

個人電腦及微軟視窗已經非常普遍，加上利用電腦網路的種種好處，有不少廠商在這樣的環境下開發一些給使用者使用的軟體（如 Telnet、FTP、News、Mail 等等）或是提供網路發展環境給使用者來開發其網路軟體。但是早期發展時，由於沒有共同的標準介面，所以各家廠商均各自發展其環境系統及應用軟體；使用者在購買了這樣的一套系統之後，不論是使用其應用軟體或是在上面開發自己的程式都必須受限於這家廠商了，因為彼此的系統開發工具（SDK,System Development Kid）的應用程式介面（API, Application Interface）並不相同，所以開發出來的應用程式也無法在不同廠商的系統上執行；如果該廠商沒有進一步提供技術支援、發展新的軟體或是連絡不便，甚或不幸倒閉了，那將是一件很令人頭痛的事。

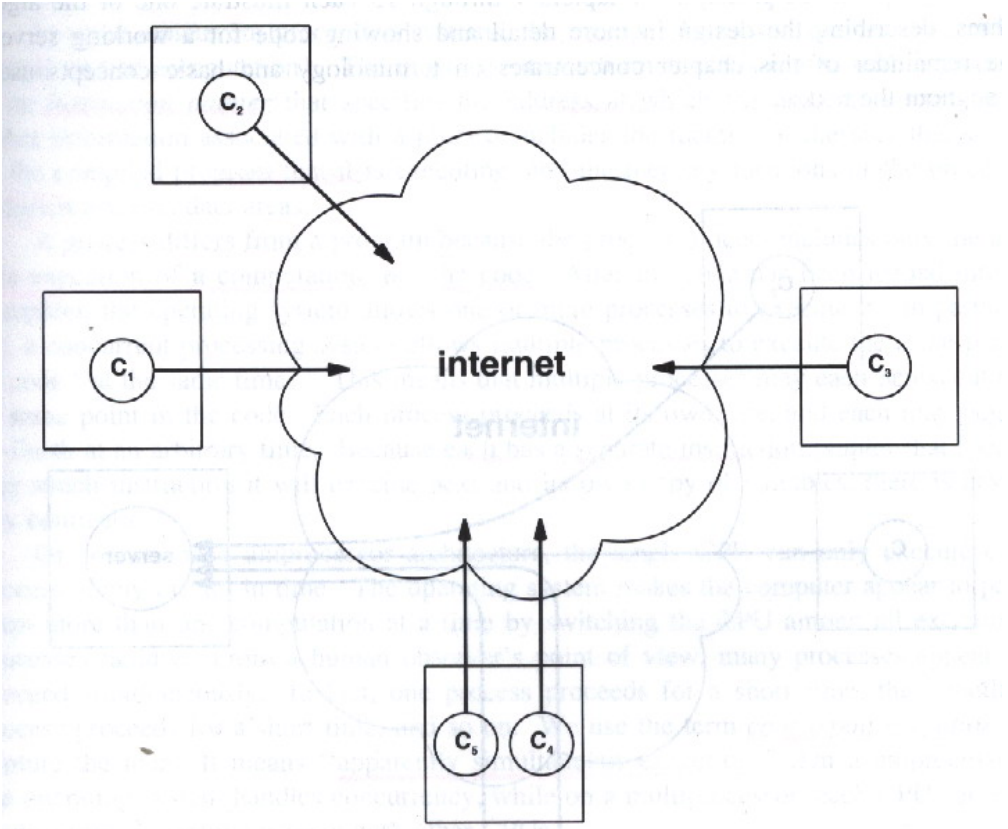


Figure 3.1 Concurrency among client programs occurs when users execute them on multiple machines simultaneously or when a multitasking operating system allows multiple copies to execute concurrently on a single computer.

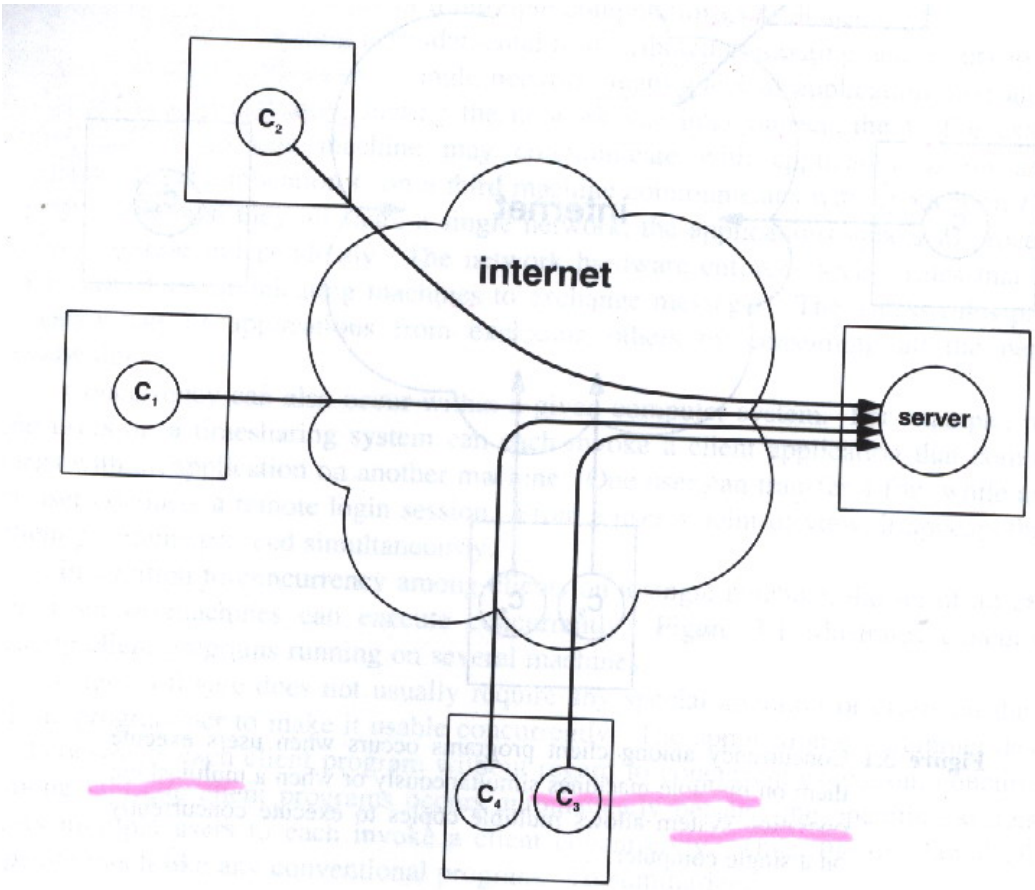


Figure 3.2 Server software must be explicitly programmed to handle concurrent requests because multiple clients contact a server using its single, well-known protocol port.

有鑑於此，於是由各公司組成的一群有志之士（包括 Microsoft、Sun Microsystems、HP、Informix、Novell、3Com ...等等）開始著手於整合定義出一套微軟視窗環境下的標準 TCP/IP 網路開發介面，並利用微軟視窗環境下「動態連結程式庫」(DLL, Dynamic Link Library）的特性，以便讓提供網路發展環境的廠商及開發網路應用程式的公司能夠各自分開發展而彼此不受限；換句話說，也就是你若遵循此一標準介面所開發出來的網路軟體應該可以在各家不同的環境上執行，而發展這些微軟視窗下 TCP/IP 網路環境的廠商亦須遵照此一標準提供符合這些介面的各項函式功能；這樣使用者就可以不受限地選擇不同廠商的軟體或是在不同家的系統環境上執行同一個自己開發的網路軟體了，甚至不必再經過修改、編譯等煩瑣的程序，真的是給使用者及開發者很大的方便。

這個標準的微軟視窗 TCP/IP 網路開發介面稱之為「Windows Sockets」(簡稱「Winsock」)，是以 UNIX 系統上 Berkeley Sockets 的函式為基礎，並加上了一些符合視窗環境「訊息驅動」(Message driven）特性的函式。目前的版本有 1993 年 1 月所定出的 1.1 版與 1995 年定義的 2.0 版。它總共定義了 46 個函式(參見表一)，包括了 30 個 BSD 函式(名稱、參數、回返值都和 UNIX 的 BSD 函

式相同)，及 16 個視窗特性的函式（WSA 開頭的函式)。Winsock 1.1 定義中並明白地表示提供此一網路開發環境的廠商一定要提供 Stream socket （TCP）及 Datagram socket （UDP）的功能。

這些 API 介面適用於 Internet Protocol Suite (IPS，通常稱之為 TCP/IP)，支援 Stream (TCP) 及 Datagram (UDP) Socket。

Stream (TCP) Socket 提供「雙向」、「可靠」、「有次序」、「不重覆」之資料傳送。Datagram (UDP) Socket 則提供「雙向」之溝通，但沒有「可靠」、「有次序」、「不重覆」等之保證； 所以使用者可能會收到無次序、重覆之資料，甚至資料在傳輸過程中也可能會遺漏。

（表一）(函式說明於附錄中）

1、BSD 函式

---

accept()	bind()	closesocket()	connect()
getpeername()	getsockname()	getsockopt()	htonl()
htons()	inet_addr()	inet_ntoa()	ioctlsocket()
listen()	ntohl()	ntohs()	recv()
recvfrom()	select()	send()	sendto()
setsockopt()	shutdown()	socket()	gethostname()
gethostbyaddr()	gethostbyname()		
getprotobyname()	getprotobynumber()		
getservbyname()	getservbyport()		

2、視窗特性函式

WSAAsyncGetHostByAddr()	WSAAsyncGetHostByName()
WSAAsyncGetProtoByName()	WSAAsyncGetProtoByNumber()
WSAAsyncGetServByName()	WSAAsyncGetServByPort()
WSAAsyncSelect()	WSACancelAsyncRequest()
WSACancelBlockingCall()	WSACleanup()
WSAGetLastError()	WSAIsBlocking()

WSASetBlockingHook() WSASetLastError()
WSAStartup() WSAUnhookBlockingHook()

二、連續（sequential）與同時（concurrent）處理

例子：一個印出從 1 到 5 的整數與總和的 C 程式

(a)連續處理

```
#include <stdlib.h>
#include <stdio.h>
#include <process.h>

int    addem(int);

int
main(int argc, char *argv[])
{
      addem(5);
      return 0;
}

int
addem(int count)
{
      int    i, sum;

      sum = 0;
      for (i=0; i<=count; ++i) {
            printf("The value of i is %d\n", i);
            fflush(stdout);
            sum += i;
      }
      printf("The sum is %d\n", sum);
      fflush(stdout);
      return 0;
}
```

執行結果＝>

(b)同時處理

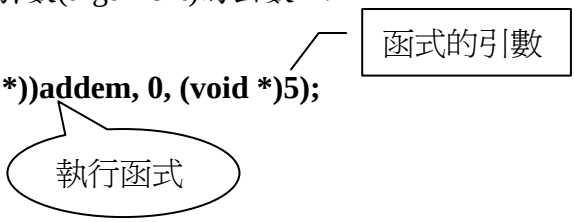
```
#include <stdlib.h>
#include <stdio.h>
#include <process.h>

int    addem(int);

int
main(int argc, char *argv[])
{
/*_beginthread() 產生一個新的thread，並讓原本的thread繼續執行下一行指令;而新的thread執行當作_beginthread引數(argument)的函數 */
    _beginthread((void (*)(void *))addem, 0, (void *)5);
    addem(5);
    return 0;
}

int
addem(int count)
{
    int    i, sum;

    sum = 0;
    for (i=0; i<=count; ++i) {
        printf("The value of i is %d\n", i);
        fflush(stdout);
        sum += i;
    }
    printf("The sum is %d\n", sum);
    fflush(stdout);
    return 0;
}
```



執行結果＝>

三、WinSock Application Programming Interface (API)

(a)架構

Windows 95 與 Windows NT 使用 Dynamic Linked Library---DLL，來執行 WinSock API。當需要時才載入記憶體，所有用到 WinSock 的程式共享一份 DLL。

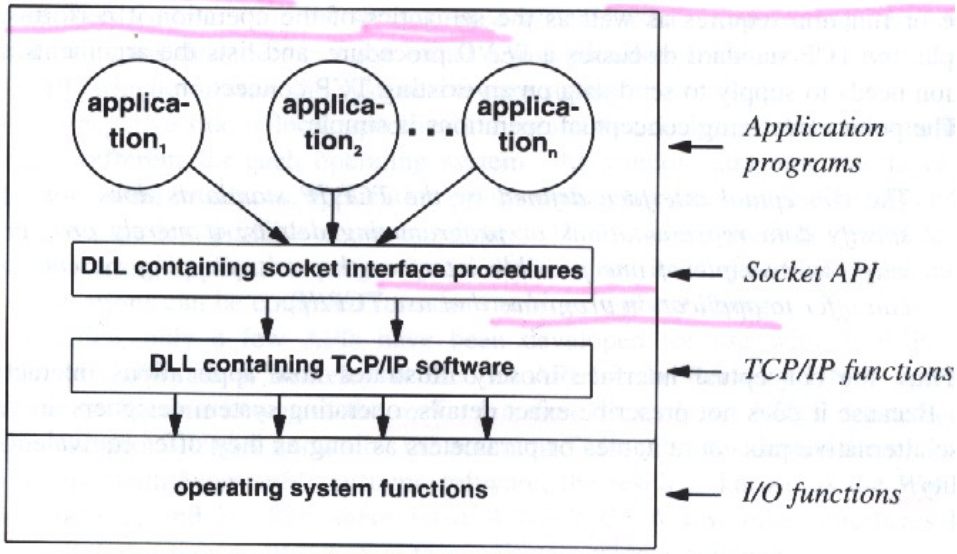


Figure 4.1 The organization of the socket API and TCP/IP code in a Dynamic Linked Library under Windows 95. One copy of a DLL is loaded into memory when needed; all applications share the copy.

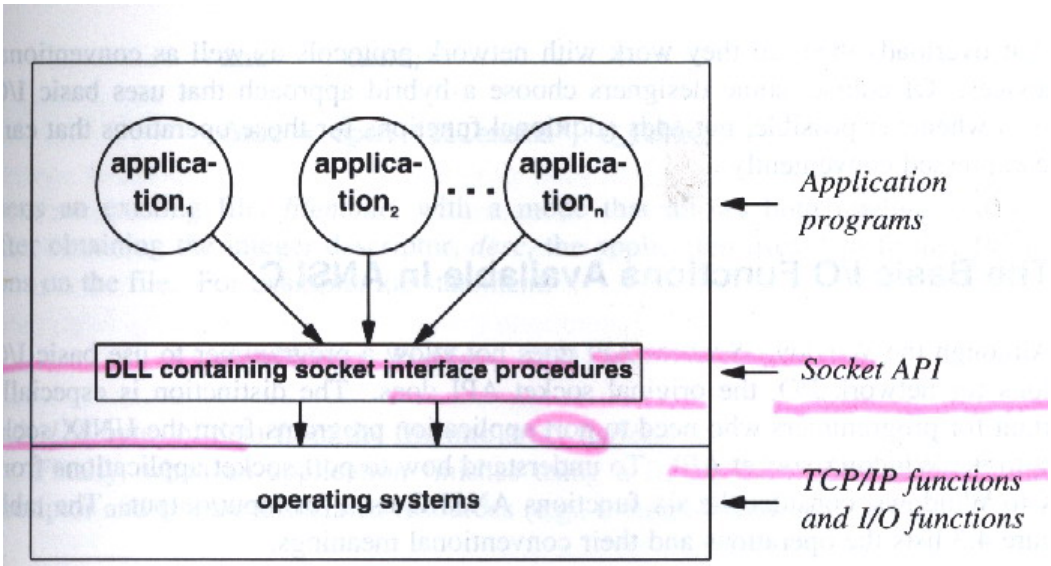


Figure 4.2 The organization of the socket API and TCP/IP code under Windows NT. Although code for TCP/IP is part of the operating system, procedures for the socket API are part of a DLL.

(b) Socket 的資料結構

使用一個 socket 描述子來表示某一個 socket。每一個處理有一個 socket 描述表。

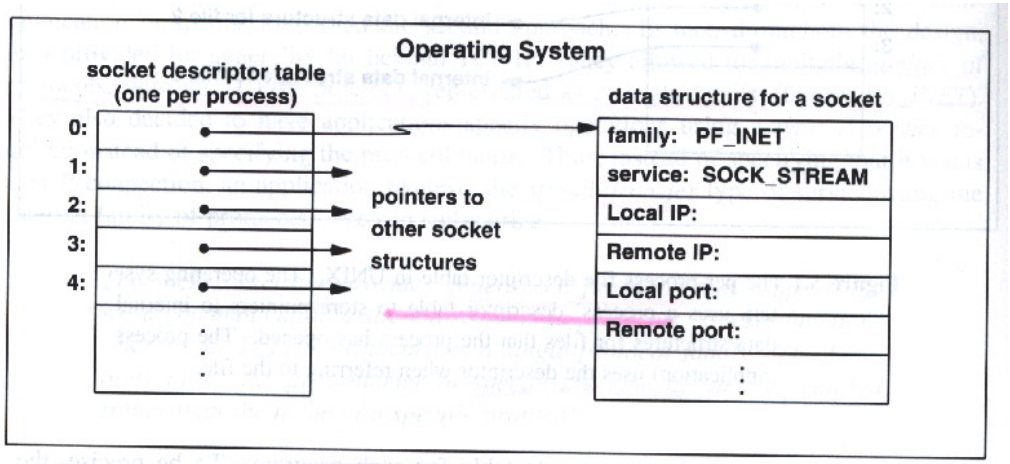


Figure 5.2 Conceptual operating system data structures after five calls to *socket*. The system keeps a separate socket descriptor table for each process; threads in the process share the table.

(c) 被動與主動 socket

伺服器(Server)所用的 socket，等著收到新的連線(connection)，稱為**被動**(passive)socket; 而**用戶端**(client)用來啟動連線建立的 socket，稱為**主動**(active)socket。

(d) 位址家族與指定端點位址

位址家族：**AF_INET** (TCP/IP 用)

端點位址：包含一個 IP 位址與協定埠(port)號碼

＊最一般化的結構

```
struct sockaddr {
    u_short    sa_family;
    char       sa_data[14];
};
```

＊TCP/IP 所用的結構

```
struct sockaddr_in {
    u_short    sin_family;          /* AF_INET */
    u_short    sin_port;            /*  協定埠(port)號碼  */
    struct     in_addr sin_addr;    /* IP 位址  */
    char       sin_zero[8];         /*  未用，設成 0*/
};
```

(e) Socket API 的函式

- WSAStartup()：第一個使用的函式
- WSACleanup()：最後一個使用的函式
- socket()：建立一個 socket，傳回一個 socket 描述子
- bind()：伺服器使用 bind()來指定本身的端點位址
- listen()：連接導向(connection-oriented)伺服器使用 listen()，使得 socket 成為被動模式，以接受新來的連結。
- accept()：伺服器使用 accept 來取得下一個連線要求。產生一個新的 socket，用來傳送連線的資料;原本的 socket 繼續接受新的連線要求。
- connect()：用戶端用來建立一條到遠端伺服器的連線
- send()：用戶端或伺服器在 TCP 連線或 UDP socket 上送資料 (blocking call)
- sendto()：將一個資料段(datagram)送到一個指定的端點
- recv()：用戶端或伺服器在 TCP 連線或 UDP socket 上收資料
- recvfrom()：收取下一個進來的資料段，並記錄傳送端的位址
- shutdown()：單或雙向的關閉一個 TCP 的連結
- closesocket()：關閉一個 socket
- htons(), htonl()：將本機的位元順序轉換成為網路上傳送的位元順序（最顯著位元優先）。分為 16(short)與 32(long)位元兩種
- ntohs(), ntohl()：將網路上傳送的位元順序轉換成為本機的位元順序

Function Name	Meaning
WSAStartup	Initialize the socket library (Windows only)
WSACleanup	Terminate use of socket library (Windows only)
socket	Create a descriptor for use in network communication
connect	Connect to a remote peer (client)
closesocket	Terminate communication and deallocate a descriptor
bind	Bind a local IP address and protocol port to a socket
listen	Place the socket in passive mode and set the number of incoming TCP connections the system will enqueue (server)
accept	Accept the next incoming connection (server)
recv	Acquire incoming data from a stream connection or the next incoming message
recvfrom	Receive the next incoming datagram and record its source endpoint address
select	Wait until the first of a specified set of sockets becomes ready for I/O
send	Send outgoing data or a message
sendto	Send an outgoing datagram to a specified endpoint address
shutdown	Terminate a TCP connection in one or both directions
getpeername	After a connection arrives, obtain the remote machine's endpoint address from a socket
getsockopt	Obtain the current options for a socket
setsockopt	Change the options for a socket

Figure 5.3 A summary of functions in the Windows Sockets API and the meaning of each.

(f) 使用 socket API 的順序

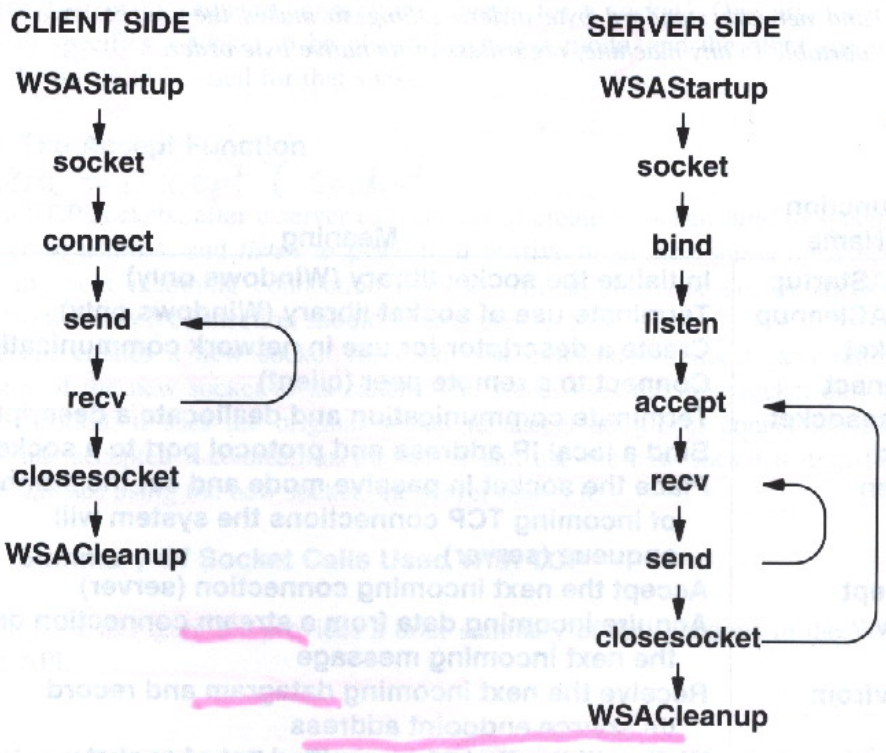


Figure 5.4 An example sequence of socket system calls made by a client and server using TCP. The server runs forever. It waits for a new connection on the well-known port, accepts the connection, interacts with the client, and then closes the connection.

(g) 使用 WinSock 定義的常數

```
#include <winsock.h>
```

四、用戶端的作法

(a) 如何取得伺服器的 IP 位址與協定埠號碼

1.在程式中指定： 簡單、但是伺服器不能改變位置。
2.程式執行時，由使用者輸入： 最常用的方式
3.由檔案或硬碟中取得： 需要每部機器都有這個檔案
4.使用特殊協定去找到伺服器： 只能是適於較小而簡單的網路環境，增加額外的資料傳送

(b) 尋找 Domain Name

兩種位址表示法：

1. Domain Name： 如 dec8.ncue.edu.tw
2. 點十進位 IP 位址： 如 163.23.1.1

需要將這兩種位址表示法轉換成為 sockaddr_in 結構中所需填入的二進位 32 位元的 IP 位址。

1. inet_addr()： 將點十進位 IP 位址的字串轉換成為二進位的 IP 位址。
2. gethostbyname()： 將機器 Domain Name 的字串傳入，取得一個含有二進位 IP 位址的 hostent 結構的位址（網路表示法）。

```
struct hostent {
    char FAR *          h_name;  /* 正式機器名稱 */
    char FAR * FAR *    h_aliases;/* 別名 */
    short               h_addrtype;   /* 位址型態 */
    short               h_length;     /* 位址長度 */
    char FAR * FAR *    h_addr_list;  /* 位址列表 */
}
#define h_addr      h_addr_list[0]     /* 位址列表的第一個 */
```

用戶端呼叫：

```
struct hostent *hptr;
char *examplenam = “dec8.cyut.edu.tw”;

if (hptr = gethostbyname( examplenam) ) {
    /* IP 位址現在在 hptr->h_addr */
} else {        /* error！傳回 NULL */
    /*處理的程式 */
}
```

(c) 由名稱尋找 well-known 埠

getservbyname()：公認的服務已經指定有公認(well-known)的協定埠。要使用他們，需先取得埠的號碼。傳回一個只到 servent 結構的指標。

```
struct servent {
    char FAR *          s_name;  /* 正式服務名稱 */
    char FAR * FAR *    s_aliases;/* 別名 */
    short               s_port;   /* 這個服務的埠 */
    char FAR *          s_proto;  /* 使用的協定 */
```

```
                    }
```

使用 SMTP 協定的 TCP 用戶端：

```
struct servent *sptr;

if (sptr = getservbyname("smtp", "tcp")) {
    /*  埠號碼在 sptr->s_port */
} else {        /* error！傳回 NULL */
    /*處理的程式 */
}
```

(d) 由名稱尋找協定

getprotobyname()：由一個協定名稱取得相對應的整數常數。傳回一個指到 protoent 結構的位址。

```
struct protoent {
        char FAR *          p_name;  /*  正式協定名稱 */
        char FAR * FAR *    p_aliases;/*  別名 */
        short               p_proto;  /*  正式協定號碼 */
}
```

用戶端尋找 UDP 的正式協定號碼：

```
struct protoent *pptr;

if (pptr = getprotobyname("udp")) {
    /*  協定號碼在 pptr->p_proto */
} else {        /* error！傳回 NULL */
    /*處理的程式 */
}
```

(e) TCP 用戶端演算法

1. **找到伺服器的 IP 位址與協定埠號碼**。
2. **配置 socket**。

```
#include <winsock.h>
```

SOCKET s;
/* 使用 TCP，需要指明協定家族：PF_INET; 服務：SOCK_STREAM; 傳回 socket 描述子 */
s = socket(PF_INET, SOCK_STREAM, 0);

3. **指明這個 TCP 連結需要選擇一個本機上尚未使用、任意的埠。**
4. **連接 socket 到伺服器。**

/* s ：socket 描述子; remaddr：表示遠端位址的 sockaddr_in 結構位址; remaddrlen：remaddr 的長度。直到 TCP 連結建立成功，才會傳回 0 ;或是失敗 */
retcode = connect(s, remaddr, remaddrlen)

5. **應用程式透過 socket 與伺服器通訊。**

『注意』：TCP 並不保留傳送資料的分界，而是將資料連接在一起。可以保證將資料送出，但接收端可能需要收好幾次，才能收滿送出的資料。

```
---------------------------------------------------------------------------
#define BLEN 120
char *req = “request of some sort”;
char buf[BLEN];
char *bptr;
int   n;
int   buflen;

bptr = buf;
buflen = BLEN;
/* send request */
send(s, req, strlen(req), 0);

/* read response (可能有許多次) */
n = recv(s, bptr, buflen, 0);
while (n != SOCKET_ERROR && n != 0) {
      bptr += n;
      buflen -= n;
      n = recv(s, bptr, buflen, 0);
}
```

---

6. **結束這個連結**。

『注意』: TCP 允許雙向通訊，有時並不知何時會送出或收到最後的資料，所以可以用 shutdown()來停止某一個方向的連接。當用戶端沒有資料要送了，他可以用 shutdown()，底下的協定會送一個 end-of-file 訊號，通知伺服器端。所以當伺服器端送完最後一個回應後，就可以關掉連接。

```
/* direction = 0 : no input;      = 1 : no output;       = 2 : no input/output */
                errcode = shutdown(s, direction)
```

範例：

```
/* conTCP.cpp - connectTCP */

#include <winsock.h>
SOCKET connectsock(const char *, const char *, const char *);
/*------------------------------------------------------------------------
 * connectTCP - connect to a specified TCP service on a specified host
 *------------------------------------------------------------------------
 */
SOCKET
connectTCP(const char *host, const char *service )
{
      return connectsock( host, service, "tcp");
}
**************************************************************************
/* consock.cpp - connectsock */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <winsock.h>

#ifndef    INADDR_NONE
#define    INADDR_NONE    0xffffffff
#endif     /* INADDR_NONE */
```

```
void errexit(const char *, ...);

/*------------------------------------------------------------------------
 * connectsock - allocate & connect a socket using TCP or UDP
 *------------------------------------------------------------------------
 */

SOCKET
connectsock(const char *host, const char *service, const char *transport )
{
        struct hostent  *phe;       /* pointer to host information entry       */
        struct servent  *pse;       /* pointer to service information entry    */
        struct protoent *ppe;       /* pointer to protocol information entry*/
        struct sockaddr_in sin;     /* an Internet endpoint address            */
        int    s, type;     /* socket descriptor and socket type       */

        memset(&sin, 0, sizeof(sin));
        sin.sin_family = AF_INET;

        /* Map service name to port number */
        if ( pse = getservbyname(service, transport) )
                sin.sin_port = pse->s_port;
        else if ( (sin.sin_port = htons((u_short)atoi(service))) == 0 )
                errexit("can't get \"%s\" service entry\n", service);

        /* Map host name to IP address, allowing for dotted decimal */
        if ( phe = gethostbyname(host) )
                memcpy(&sin.sin_addr, phe->h_addr, phe->h_length);
        else if ( (sin.sin_addr.s_addr = inet_addr(host)) == INADDR_NONE)
                errexit("can't get \"%s\" host entry\n", host);

        /* Map protocol name to protocol number */
        if ( (ppe = getprotobyname(transport)) == 0)
                errexit("can't get \"%s\" protocol entry\n", transport);
        /* Use protocol to choose a socket type */
        if (strcmp(transport, "udp") == 0)
                type = SOCK_DGRAM;
        else
```

```
        type = SOCK_STREAM;

    /* Allocate a socket */
    s = socket(PF_INET, type, ppe->p_proto);
    if (s == INVALID_SOCKET)
        errexit("can't create socket: %d\n", GetLastError());

    /* Connect the socket */
    if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) ==
        SOCKET_ERROR)
        errexit("can't connect to %s.%s: %d\n", host, service,
            GetLastError());
    return s;
}

************************************************************************
/* errexit.cpp - errexit */

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>

/*------------------------------------------------------------------------
 * errexit - print an error message and exit
 *------------------------------------------------------------------------
 */
/*VARARGS1*/
void
errexit(const char *format, ...)
{
    va_list     args;

    va_start(args, format);
    vfprintf(stderr, format, args);
    va_end(args);
    WSACleanup();
    exit(1);
```

```
}
**********************************************************************
```

[DAYTIME 的服務]

用戶端連上 DAYTIME 伺服器，取得伺服器的日期與時間。格式可以如：星期幾,幾月幾號,年份 時間-時區(Thursday, February 22, 1997 17:37:43-EST)。可以使用 TCP 或 UDP，都用埠 13。

（**TCP 版本**）：當伺服器收到一個新的連結，他將會取得當時的日期與時間的字串，送回給用戶端，然後結束連結。用戶不需送任何需求。

```
/* TCPdtc.cpp - main, TCPdaytime */

#include <stdlib.h>
#include <stdio.h>
#include <winsock.h>

void TCPdaytime(const char *, const char *);
void errexit(const char *, ...);
SOCKET connectTCP(const char *, const char *);

#define    LINELEN              128
#define WSVERS          MAKEWORD(2, 0)

/*------------------------------------------------------------------------
 * main - TCP client for DAYTIME service
 *------------------------------------------------------------------------
 */
int
main(int argc, char *argv[])
{
      char *host = "localhost";/* host to use if none supplied*/
      char *service = "daytime";     /* default service port          */
      WSADATA wsadata;

      switch (argc) {
      case 1:
            host = "localhost";
            break;
```

```
        case 3:
                service = argv[2];
                /* FALL THROUGH */
        case 2:
                host = argv[1];
                break;
        default:
                fprintf(stderr, "usage: TCPdaytime [host [port]]\n");
                exit(1);
        }

        if (WSAStartup(WSVERS, &wsadata) != 0)
                errexit("WSAStartup failed\n");
        TCPdaytime(host, service);
        WSACleanup();
        return 0;  /* exit */
}

/*------------------------------------------------------------------------
 * TCPdaytime - invoke Daytime on specified host and print results
 *------------------------------------------------------------------------
 */
void
TCPdaytime(const char *host, const char *service)
{
        char buf[LINELEN+1];        /* buffer for one line of text   */
        SOCKET s;                   /* socket descriptor        */
        int    cc;                  /* recv character count          */

        s = connectTCP(host, service);

        cc = recv(s, buf, LINELEN, 0);
        while( cc != SOCKET_ERROR && cc > 0) {
                buf[cc] = '\0';/* ensure null-termination      */
                (void) fputs(buf, stdout);
                cc = recv(s, buf, LINELEN, 0);
        }
        closesocket(s);
```

}

（UDP 版本）：用戶端需要送一個需求，包含隨意的資料段。當伺服器收到這個資料段，他會將日期與時間的字串送回用戶端。

****

[ECHO 的服務]：伺服器只傳回所有由用戶端所收到的資料
（TCP 版本）：

```
/* TCPecho.cpp - main, TCPecho */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <winsock.h>

void TCPecho(const char *, const char *);
void errexit(const char *, ...);
SOCKET connectTCP(const char *, const char *);

#define LINELEN 128
#define WSVERS MAKEWORD(2, 0)

/*------------------------------------------------------------------------
 * main - TCP client for ECHO service
 *------------------------------------------------------------------------
 */
void
main(int argc, char *argv[])
{
    char *host = "localhost";    /* host to use if none supplied */
    char *service = "echo"; /* default service name */
    WSADATA wsadata;

    switch (argc) {
    case 1:
        host = "localhost";
```

```
            break;
        case 3:
            service = argv[2];
            /* FALL THROUGH */
        case 2:
            host = argv[1];
            break;
        default:
            fprintf(stderr, "usage: TCPecho [host [port]]\n");
            exit(1);
        }
        if (WSAStartup(WSVERS, &wsadata) != 0)
            errexit("WSAStartup failed\n");
        TCPecho(host, service);
        WSACleanup();
        exit(0);
}

/*------------------------------------------------------------------------
 * TCPecho - send input to ECHO service on specified host and print reply
 *------------------------------------------------------------------------
 */
void
TCPecho(const char *host, const char *service)
{
        char buf[LINELEN+1];        /* buffer for one line of text    */
        SOCKET s;                   /* socket descriptor              */
        int   cc, outchars, inchars;  /* characters counts            */

        s = connectTCP(host, service);

        while (fgets(buf, sizeof(buf), stdin)) {
            buf[LINELEN] = '\0';  /* ensure line null-termination   */
            outchars = strlen(buf);
            (void) send(s, buf, outchars, 0);

            /* read it back */
            for (inchars = 0; inchars < outchars; inchars += cc) {
```

```
            cc = recv(s, &buf[inchars], outchars-inchars, 0);
            if (cc == SOCKET_ERROR)
                errexit("socket recv failed: %d\n",
                    GetLastError());
        }
        fputs(buf, stdout);
    }
    closesocket(s);
}
**********************************************************************
```

(f) UDP 用戶端演算法

1. **找到伺服器的 IP 位址與協定埠號碼**。
2. **配置 socket**。

   s = socket(PF_INET, SOCK_DGRAM, 0);
3. **指明這個 UDP 連結需要選擇一個本機上尚未使用、任意的埠**。

   UDP 的用戶端有兩種模式：

   （**連接式**）：使用 connect()來指定遠端位址。之後就像 TCP 一樣收送資料。

   （**未連接式**）：不連接遠端的位址。每次送一個訊息，就要指定遠端一次。

4. **指出資料要送到的伺服器**。
5. **應用程式與伺服器通訊**。

『注意』：UDP 的收送，都是以一個訊息為單位，不需呼叫多次 recv()。

6. **結束這個連結**。

『注意』：呼叫 shutdown()，只有做個紀錄，並不送訊號到遠端。

範例：

```
/* conUDP.cpp - connectUDP */

#include <winsock.h>

SOCKET connectsock(const char *, const char *, const char *);

/*------------------------------------------------------------------------
 * connectUDP - connect to a specified UDP service on a specified host
```

```
 *------------------------------------------------------------------------
 */
SOCKET
connectUDP(const char *host, const char *service )
{
      return connectsock(host, service, "udp");
}
```

[TIME 的服務]

用戶端連上 TIME 伺服器，取得伺服器的日期與時間。伺服器轉換當地時間，成為格林威治時間，用戶端收到後，再換成用戶端的時間，以避免時區的不同。結果是一 32 位元的整數，表示現在與 1900 年 1 月 1 日午夜的秒差。不是給人讀的，而是給程式所用。可以使用 TCP 或 UDP，都用埠 37。

（**TCP 版本**）：當伺服器收到一個新的連結，他將會取得當時時間的 整數，送回給用戶端，然後結束連結。用戶不需送任何需求。

（**UDP 版本**）：用戶端需要送一個需求，包含一個隨意的資料段。當伺服器收到這個資料段，他會取得傳送者的位址與埠號碼，將時間的整數放在資料段中送回用戶端。

```
/* UDPtime.cpp - main */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <winsock.h>

#define   BUFSIZE              64
#define   WSVERS            MAKEWORD(2, 0)

#define   WINEPOCH   2208988800   /* Windows epoch, in UCT secs    */
#define   MSG             "what time is it?\n"

SOCKET connectUDP(const char *, const char *);
void errexit(const char *, ...);

/*------------------------------------------------------------------------
 * main - UDP client for TIME service that prints the resulting time
 *------------------------------------------------------------------------
 */
```

```
int
main(int argc, char *argv[])
{
	char *host = "localhost";/* host to use if none supplied*/
	char *service = "time";   /* default service name          */
	time_t      now;                    /* 32-bit integer to hold time  */
	SOCKET s;                   /* socket descriptor        */
	int    n;                   /* recv count               */
	WSADATA     wsadata;

	switch (argc) {
	case 1:
		host = "localhost";
		break;
	case 3:
		service = argv[2];
		/* FALL THROUGH */
	case 2:
		host = argv[1];
		break;
	default:
		fprintf(stderr, "usage: UDPtime [host [port]]\n");
		exit(1);
	}
	if (WSAStartup(WSVERS, &wsadata))
		errexit("WSAStartup failed\n");

	s = connectUDP(host, service);

	(void) send(s, MSG, strlen(MSG), 0);

	/* Read the time */

	n = recv(s, (char *)&now, sizeof(now), 0);
	if (n == SOCKET_ERROR)
		errexit("recv failed: recv() error %d\n", GetLastError());
	WSACleanup();
	now = ntohl((u_long)now);    /* put in host byte order  */
```

```
    now -= WINEPOCH;            /* convert UCT to Windows epoch*/
    printf("%s", ctime(&now));  /* ctime()轉換成人們可以讀的格式 */
    return 0;  /* exit */
}
```

『注意』: Windows 是從 1970 年 1 月 1 日午夜開始計時。所以需要減掉與 1900 年 1 月 1 日午夜間的秒差：2208988800。

五、伺服器端的作法

(a) 反覆(iterative)與同時(concurrent)伺服器

1. **反覆伺服器**：每次只能處理一個需求。=> 設計簡單，但是可能要等很久。
2. **同時伺服器**：可以一次處理好幾個需求。=>複雜，但是效率較好！

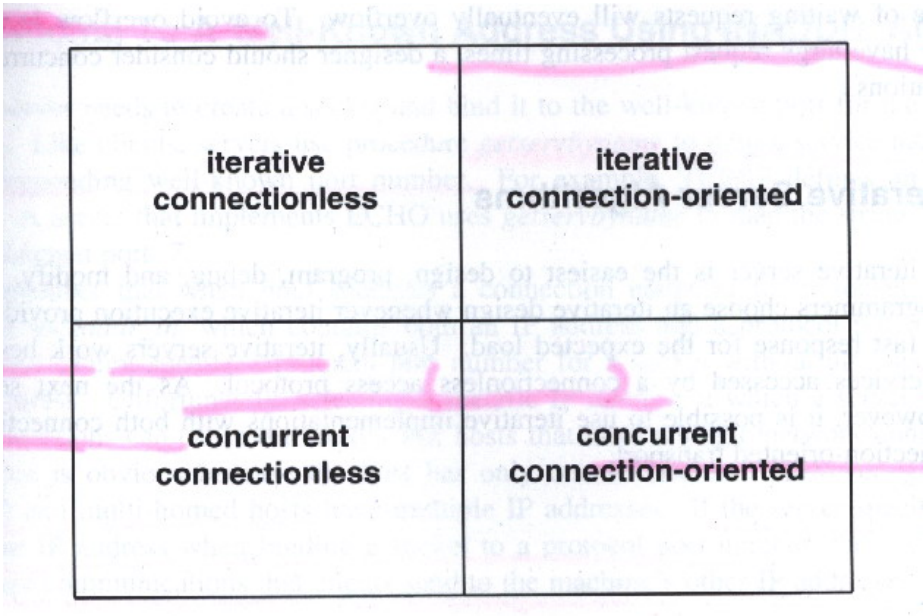


Figure 8.2 The four general server categories defined by whether they offer concurrency and whether they use connection-oriented transport.

(b) 連接導向(connection-oriented)與無連接(connectionless)伺服器

1. **連接導向伺服器**：使用 TCP（可靠），與每個用戶端都要建立一個連結（使用一個 socket）。缺點是：若沒有好好的處理，連結會耗盡大量資源！
2. **無連接伺服器**：使用 UDP（不可靠），可以用一個 socket 與多個用戶端通訊。不會耗盡資源，適用於廣播(broadcast)、群播(multicast)的應用。缺點是：要應用程式處理『可靠性』的問題！

(c) 四種伺服器種類

1. **反覆、連接導向伺服器**(TCP)

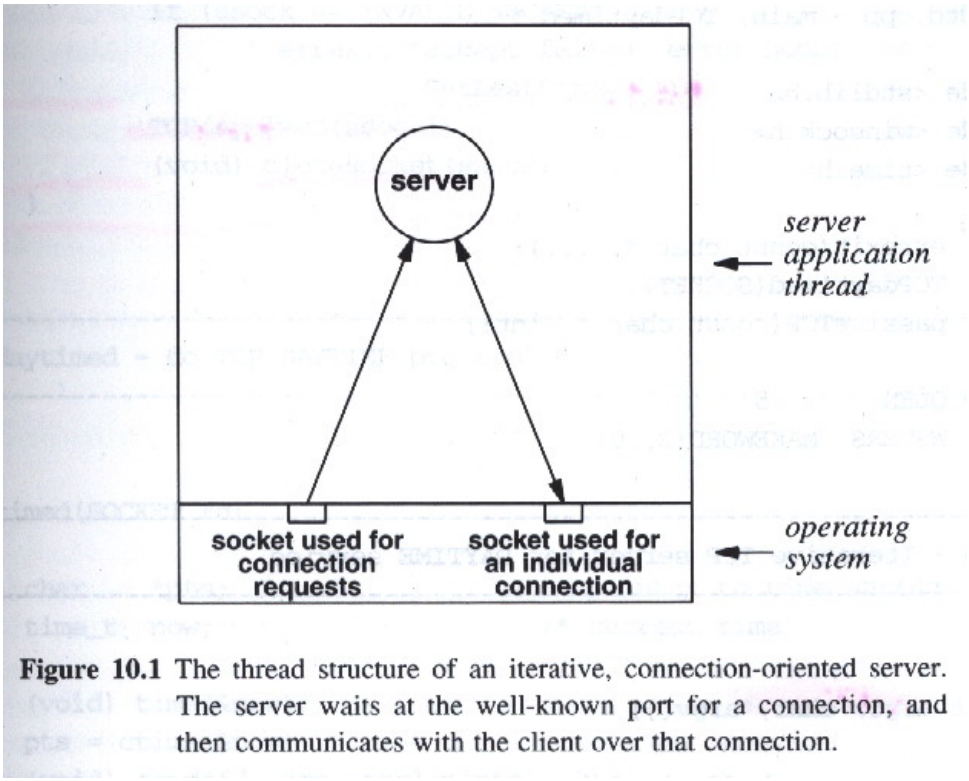


Figure 10.1 The thread structure of an iterative, connection-oriented server. The server waits at the well-known port for a connection, and then communicates with the client over that connection.

[**演算法**]

1) 建立一個 socket，將他連結(bind)到所用服務的公開位址上。=> socket(), bind()
   注意：對於有多個介面（IP 位址）的伺服器，使用『INADDR_ANY』常數，取代一個特別的 IP 位址，就可以從任何一個介面收取需求！

2) 將 socket 設成『被動』模式，準備讓伺服器使用。=> listen()
3) 從 socket 接收一個連結需求，產生一個新的 socket 讓這個連結使用。=> accept()
4) 根據應用程式，重複從用戶端收取需求，送回回應。=> send(), recv()
5) 當某個用戶端結束了，關閉與他的連結。回到第 3 步！

2. 反覆、非連接伺服器(UDP)

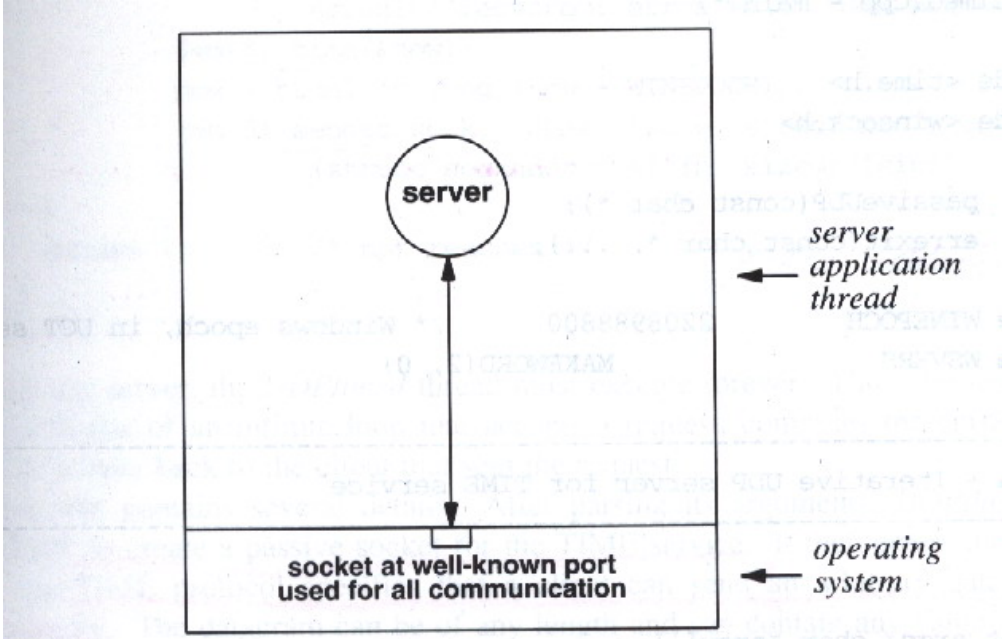


Figure 9.1 The thread structure for an iterative, connectionless server. A singly-threadeded server communicates with many clients using one socket.

[演算法]

1) 建立一個 socket，將他連結(bind)到所用服務的公開位址上。
2) 根據應用程式，重複從用戶端收取需求，送回回應。

/* s: socket; buf: 收到資料的緩衝區指標; len: 可用緩衝區長度; flags: 控制旗標; from: 指向傳送端點位置 sockaddr_in 的指標; fromlen: sockaddr_in 長度 */

retcode = recvfrom(s, buf, len, flags, from, fromlen);

/* s: socket; message: 要送資料的緩衝區指標; len: 資料長度; flags: 控制旗標; toaddr: 指向目的地端點位置 sockaddr_in 的指標（使用 recvfrom()中的 from 指標）; toaddrlen: sockaddr_in 長度 */

retcode = sendto(s, message, len, flags, toaddr, toaddrlen);

範例：

/* passUDP.cpp - passiveUDP */

#include <winsock.h>

SOCKET passivesock(const char *, const char *, int);

```
/*------------------------------------------------------------------------
 * passiveUDP - create a passive socket for use in a UDP server
 *------------------------------------------------------------------------
 */
SOCKET
passiveUDP(const char *service)
{
	return passivesock(service, "udp", 0);
}
************************************************************************
```

/* passivesock()的三個引數：服務名稱，協定名稱，所需連接需求 queue 長度 */

```
/*  passsock.cpp  -  passivesock  */

#include  <stdlib.h>
#include  <string.h>
#include  <winsock.h>

void errexit(const  char  *,  ...);
u_short   portbase  =  0;       /*  port  base,  for  test  servers          */

/*------------------------------------------------------------------------
 *  passivesock  -  allocate  &  bind  a  server  socket  using  TCP  or  UDP
 *------------------------------------------------------------------------
 */
SOCKET
passivesock(const  char  *service,  const  char  *transport,  int  qlen)
{
	struct  servent  *pse;     /*  pointer  to  service  information  entry     */
	struct  protoent  *ppe;    /*  pointer  to  protocol  information  entry*/
	struct  sockaddr_in  sin;  /*  an  Internet  endpoint  address         */
	SOCKET      s;    /*  socket  descriptor                  */
	int         type;/*  socket  type  (SOCK_STREAM,  SOCK_DGRAM)*/

	memset(&sin,  0,  sizeof(sin));
	sin.sin_family  =  AF_INET;
	sin.sin_addr.s_addr  =  INADDR_ANY;
```

```
    /* Map service name to port number */
    if ( pse = getservbyname(service, transport) )
        sin.sin_port = htons(ntohs((u_short)pse->s_port)
            + portbase);
    else if ( (sin.sin_port = htons((u_short)atoi(service))) == 0 )
        errexit("can't get \"%s\" service entry\n", service);

    /* Map protocol name to protocol number */
    if ( (ppe = getprotobyname(transport)) == 0)
        errexit("can't get \"%s\" protocol entry\n", transport);

    /* Use protocol to choose a socket type */
    if (strcmp(transport, "udp") == 0)
        type = SOCK_DGRAM;
    else
        type = SOCK_STREAM;

    /* Allocate a socket */
    s = socket(PF_INET, type, ppe->p_proto);
    if (s == INVALID_SOCKET)
        errexit("can't create socket: %d\n", GetLastError());

    /* Bind the socket */
    if (bind(s, (struct sockaddr *)&sin, sizeof(sin)) == SOCKET_ERROR)
        errexit("can't bind to %s port: %d\n", service,
            GetLastError());
    if (type == SOCK_STREAM && listen(s, qlen) == SOCKET_ERROR)
        errexit("can't listen on %s port: %d\n", service,
            GetLastError());
    return s;
}
************************************************************
```

[TIME 伺服器]：用戶端送一個任意資料段給伺服器，而回應時間

```
/* UDPtimed.cpp - main */

#include <time.h>
```

```
#include <winsock.h>

SOCKET passiveUDP(const char *);
void errexit(const char *, ...);

#define WINEPOCH 2208988800 /* Windows epoch, in UCT secs */
#define WSVERS MAKEWORD(2, 0)

/*------------------------------------------------------------------------
 * main - Iterative UDP server for TIME service
 *------------------------------------------------------------------------
 */
int
main(int argc, char *argv[])
{
    struct sockaddr_in fsin; /* the from address of a client */
    char *service = "time"; /* service name or port number */
    char buf[2048]; /* "input" buffer; any size > 1 packet */
    SOCKET sock; /* server socket */
    time_t now; /* current time */
    int alen; /* from-address length */
    WSADATA wsadata;

    switch (argc) {
    case 1:
        break;
    case 2:
        service = argv[1];
        break;
    default:
        errexit("usage: UDPtimed [port]\n");
    }

    if (WSAStartup(WSVERS, &wsadata))
        errexit("WSAStartup failed\n");
    sock = passiveUDP(service);

    while (1) {
```

```
        alen = sizeof(fsin);
        if (recvfrom(sock, buf, sizeof(buf), 0,
            (struct sockaddr *)&fsin, &alen) == SOCKET_ERROR)
            errexit("recvfrom: error %d\n", GetLastError());
        (void) time(&now);
        now = htonl((u_long)(now + WINEPOCH));
        (void) sendto(sock, (char *)&now, sizeof(now), 0,
            (struct sockaddr *)&fsin, sizeof(fsin));
    }
    return 1; /* not reached */
}
```

3. 同時、非連接伺服器

[**演算法**]

主要 1) 建立一個 socket，將他連結(bind)到所用服務的公開位址上。讓 socket 未連接！

主要 2) 反覆呼叫 recvfrom()，接收從用戶端來的下個需求，並建立一個新的『從屬線(slave thread)來處理這個回應。

從屬 1) 在建立時，收取這個特殊的需求與對 socket 的存取。

從屬 2) 根據應用程式，產生回應，並使用 sendto()送回回應。

從屬 3) 結束這個從屬線（只處理一個需求）。

4. 同時、連接導向伺服器(TCP)

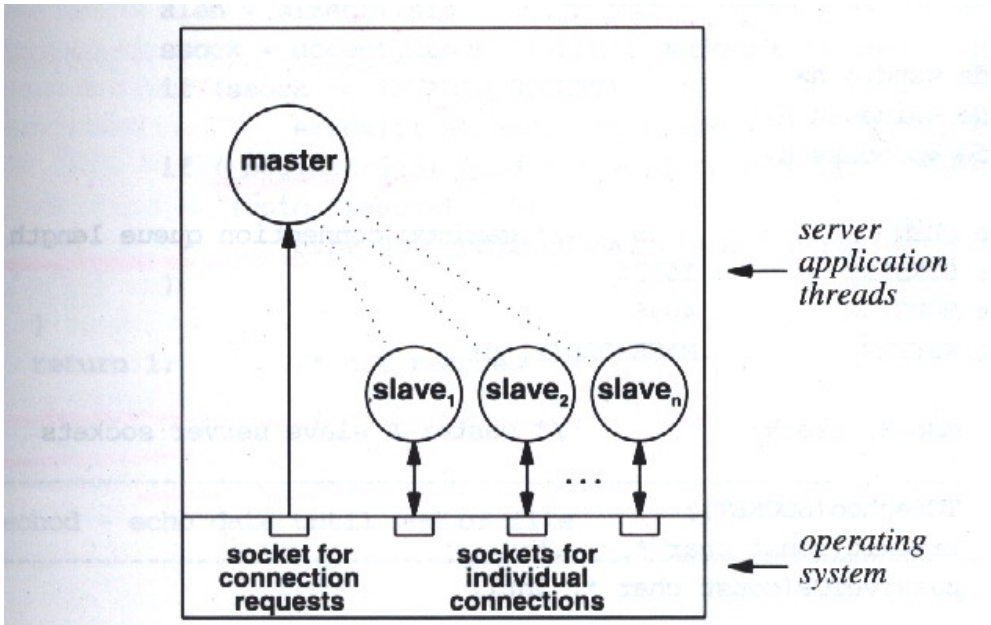


Figure 11.1 The thread structure of a concurrent, connection-oriented server. A master server thread accepts each incoming connection, and creates a slave thread to handle it.

[**演算法**]

主要 1) 建立一個 socket，將他連結(bind)到所用服務的公開位址上。讓 socket 未連接！

主要 2) 將 socket 設成『被動』模式，準備讓伺服器使用。

主要 3) 反覆呼叫 accept()，接收從用戶端來的下個需求，並建立一個新的『從屬線(slave thread)來處理這個回應。

從屬 1) 在建立時，收取這個連線需求。

從屬 2) 透過連結，與用戶端收送需求與回應。

從屬 3) 關閉這個連結，結束這個從屬線（處理完這個用戶端所有的需求）。

範例：ECHO 伺服器服務

```
/* TCPechod.cpp - main, TCPechod */

#include <stdio.h>
#include <winsock.h>
#include <process.h>

#define QLEN            5       /* maximum connection queue length      */
```

```
#define   STKSIZE         16536
#define   BUFSIZE         4096
#define   WSVERS          MAKEWORD(2, 0)

SOCKET msock, ssock;      /* master & slave server sockets    */

int   TCPechod(SOCKET);
void errexit(const char *, ...);
SOCKET passiveTCP(const char *, int);

/*------------------------------------------------------------------------
 * main - Concurrent TCP server for ECHO service
 *------------------------------------------------------------------------
 */
int
main(int argc, char *argv[])
{
      char *service = "echo"; /* service name or port number */
      struct    sockaddr_in fsin;  /* the address of a client   */
      int   alen;             /* length of client's address */
      WSADATA    wsadata;

      switch (argc) {
      case 1:
            break;
      case 2:
            service = argv[1];
            break;
      default:
            errexit("usage: TCPechod [port]\n");
      }

      if (WSAStartup(WSVERS, &wsadata) != 0)
            errexit("WSAStartup failed\n");
      msock = passiveTCP(service, QLEN);

      while (1) {
            alen = sizeof(fsin);
```

```
            ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
            if (ssock == INVALID_SOCKET)
                errexit("accept: error number\n", GetLastError());
            if (_beginthread((void (*)(void *))TCPechod, STKSIZE,
                (void *)ssock) < 0) {
                errexit("_beginthread: %s\n", strerror(errno));
            }
        }
        return 1; /* not reached */
}

/*------------------------------------------------------------------------
 * TCPechod - echo data until end of file
 *------------------------------------------------------------------------
 */
int
TCPechod(SOCKET fd)
{
        char buf[BUFSIZE];
        int   cc;

        cc = recv(fd, buf, sizeof buf, 0);
        while (cc != SOCKET_ERROR && cc > 0) {
            if (send(fd, buf, cc, 0) == SOCKET_ERROR) {
                fprintf(stderr, "echo send error: %d\n",
                    GetLastError());
                break;
            }
            cc = recv(fd, buf, sizeof buf, 0);
        }
        if (cc == SOCKET_ERROR)
            fprintf(stderr, "echo recv error: %d\n", GetLastError());
        closesocket(fd);
        return 0;
}
```

六、參考書目

[1] 林軍鼐, "Winsock API", bbs.ntu.edu.tw 之 Network board

[2] D. Comer, D. Stevens, "Internetworking with TCP/IP, Vol. III, Client-Server Programming and Applications: Windows Sockets Version", Prentice Hall, 1997.

[3] W. Stevens, "Unix Networking Programming-Networking APIs: Sockets and XTI", Vol. 1, the 2nd ed., Prentice Hall, 1998.

[4] 李孟書、黃鶴超譯, "Windows Sockets 網路程式設計經典", 基峰, 1997.

附錄---函式說明

一、BSD Socket 程式庫

(1) **accept()**：接受某一 Socket 的連接要求，以完成 Stream Socket 的連接。

格　式： SOCKET PASCAL FAR accept(SCOKET s,
struct sockaddr FAR *addr,
int FAR *addrlen);

參　數： s Socket 的識別碼
addr 存放來連接的彼端的位址
addrlen addr 的長度

傳回值：成功 - 新的 Socket 識別碼
失敗 - INVALID_SOCKET (呼叫 WSAGetLastError() 可得知原因)

說明： Server 端之應用程式呼叫此一函式來接受 Client 端要求之 Socket 連接動作；如果 Server 端之 Socket 是為 Blocking 模式，且沒有人要求連接動作，那麼此一函式會 Block 函式馬上回覆錯誤。accept() 函式的答覆值為一新的 Socket，此 Socket 不可再用來接受其它的連接要求；但是原先之 Socket 仍可接受其他人的連接要求。

(2) **bind()**：指定 Socket 的 Local 位址 (Address)。

格　式： int PASCAL FAR bind(SOCKET s,
const struct sockaddr FAR *name,
int namelen);

參　數：s Socket 的識別碼
name Socket 的位址值，其格式為

```
struct sockaddr {
    u_short    sa_family;
    char       sa_data[14];
};
```

namelen name 的長度

傳回值：成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明： 此一函式是指定 Local 位址及 Port 給某一未定名之 Socket。使用者若不在意位址或 Port 的值，那麼他可以設定位址為 INADDR_ANY，及 Port 為 0；那麼 Windows Sockets 會自動將其設定適當之位址及 Port(1024 到 5000

之間的值)，使用者可以在此 Socket 真正連接完成後，呼叫 getsockname() 來獲知其被設定的值。

(3) **closesocket()**：關閉某一 Socket。

格　式： int PASCAL FAR closesocket(SOCKET s);
參　數： s　Socket 的識別碼
傳回值：成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)
說明： 此一函式是用來關閉某一 Socket。若是使用者原先對要關閉之 Socket 設定 SO_DONTLINGER，則在呼叫此一函式後，會馬上回覆，但是此一 Sokcet 尚未傳送完畢的資料會繼續送完後才關閉。若是使用者原先設定此 Socket 為 SO_LINGER，則有兩種情況：

(a) Timeout 設為 0 的話，此一 Socket 馬上重新設定 (reset)，未傳完或未收到的資料全部遺失。
(b) Timeout 不為 0 的話，則會將資料送完，或是等到 Timeout 發生後才關閉。

(4) **connect()**：要求連接某一 Socket 到指定的對方。

格　式： int PASCAL FAR connect(SOCKET s,
const struct sockaddr
FAR *name,
int namelen);
參　數：s　Socket 的識別碼
name　此 Socket 想要連接的對方位址
namelen　name 的長度
傳回值：成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError()可得知原因)
說明： 此函式用來向對方要求建立連接。若是指定的對方位址為 0 的話，會傳回錯誤值。當連接建立完成後，使用者即可利用此一 Socket 來做傳送或接收資料之用了。

(5) **getpeername()**：獲取已連接成功之 Socket 的對方位址。

格　式： int PASCAL FAR getpeername(SOCKET s,
struct sockaddr FAR *name,
int FAR *namelen);

參　數：s Socket 的識別碼
name 此 Socket 連接的對方位址
namelen name 的長度
傳回值：成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)
說明： 此函式可用來取得已連接成功的 Socket 的彼端之位址資料。

(6) **getsockname()**：獲取 Socket 的 Local 位址資料。

格　式： int PASCAL FAR getsockname(SOCKET s,
struct sockaddr FAR *name,
int FAR *namelen);
參　數：s Socket 的識別碼
name 此 Socket 的 Local 位址
namelen name 的長度
傳回值：成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)
說明： 此函式是用來取得已設定位址或已連接之 Socket 的本端位址資料。若是此 Socket 被設定為 INADDR_ANY，則需等真正建立連接成功後才會傳回正確的位址。

(7) **getsockopt()**：要求某一 Socket 目前狀態設定的資料。

格式： int PASCAL FAR getsockopt(SOCKET s,
int level,
int optname,
char FAR *optval,
int FAR *optlen);
參　數：s Socket 的識別碼
level 選項設定的 level
optname 選項名稱
optval 選項的設定值
optlen 選項設定值的長度
傳回值：成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)
說明： 此函式用來獲取目前 Socket 的某些狀態設定值。WINSOCK 提供之 level 只有 SOL_SOCKET 及 IPPROTO_TCP。 optname 則有以下之選擇：

Value	Type	(* 表暫不提供此功能選項)
SO_ACCEPTCONN	BOOL	
SO_BROADCAST	BOOL	
*SO_DEBUG	BOOL	
SO_DONTLINGER	BOOL	
*SO_DONTROUTE	BOOL	
*SO_ERROR	int	
*SO_KEEPALIVE	BOOL	
SO_LINGER	struct linger FAR*	
SO_OOBINLINE	BOOL	
*SO_RCVBUF	int	
SO_REUSEADDR	BOOL	
*SO_SNDBUF	int	
SO_TYPE	int	
TCP_NODELAY	BOOL	

(8) **htonl()**：將一 32 位元 u_long 的值由 host 的排列方式轉換成 network 的排列方式。

格　式： u_long PASCAL FAR htonl(u_long hostlong);
參　數： hostlong 一個 32 位元 host 排列方式的數目
傳回值： 一個 32 位元 network 排列方式的數目
說明： 因為 network 的排列方式與 host 的排列方式可能不同，所以我們需要此一函式來做轉換。

(9) **htons()**：將一 16 位元 u_short 的值由 host 的排列方式轉換成 network 的排列方式。

格　式： u_short PASCAL FAR htons(u_short hostshort);
參　數： hostshort 一個 16 位元 host 排列方式的數目
傳回值： 一個 16 位元 network 排列方式的數目
說明： 因為 network 的排列方式與 host 的排列方式可能不同，所以我們需要此一函式來做轉換。

(10) **inet_addr()**：將字串格式的位址轉換成 32 位元 unsigned long 的格式。

格　式： unsigned long PASCAL FAR inet_addr(const char FAR *cp);

參　數： cp　　一個代表位址的「點格式」(dotted) 字串
傳回值：成功 - 一個代表 Internet 位址的 unsigned long
　　失敗 - INADDR_NONE
說明： 此函式將一「點格式」的位址字串轉換成適用之 Intenet 位址。「點格式」字串可為以下四種方式之任一：
(i) a.b.c.d　(ii) a.b.c　(iii) a.b　(iv) a

(11) **inet_ntoa()**：將一網路位址轉換成「點格式」字串。

格　式： char FAR * PASCAL FAR inet_ntoa(struct in_addr in);
參　數： in　　一個代表 Internet 位址的結構
傳回值：成功 - 一個代表位址的「點格式」(dotted) 字串
　　失敗 - NULL
說明： 此函式將一 Internet 位址轉換成「a.b.c.d」字串格式。

(12) **ioctlsocket()**：控制 Socket 的模式。

格　式： int PASCAL FAR ioctlsocket(SOCKET s,
　　long cmd,
　　u_long　FAR *argP);
參　數：s　　Socket 的識別碼
　　cmd　　指令名稱
　　argP　　指向 cmd 參數的指標
傳回值：成功 - 0
　　失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)
說明： 此函式用來獲取或設定 Socket 的運作參數。其所提供的指令有：
　　FIONBIO　　開關 non-blocking 模式
　　FIONREAD　　自 Socket 一次可讀取的資料量
　　SIOCATMARK　　OOB 資料是否已被讀取完 (*暫不提供此功能)

(13) **listen()**：設定 Socket 為監聽狀態，準備被連接。

格　式： int PASCAL FAR listen(SOCKET s, int backlog);
參　數：s　　Socket 的識別碼
　　backlog　　未真正完成連接前(尚未呼叫 accept() 前)彼端的連接要求的最大個數
傳回值：成功 - 0
　　失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明： 使用者可利用此函式來設定 Socket 進入監聽狀態，並設定最多可有多少個在未真正完成連接前的彼端的連接要求。(目前最大值限制為 5, 最小值為 1)

(14) **ntohl()**：將一 32 位元 u_long 的值由 network 排列方式轉換成 host 排列方式。

格　式： u_long PASCAL FAR ntohl(u_long netlong);
參　數： netlong　　一個 32 位元 network 排列方式的數目
傳回值： 一個 32 位元 host 排列方式的數目
說明： 因為 network 的排列方式與 host 的排列方式可能不同，所以我們需要此一函式來做轉換。

(15) **ntohs()**：將一 16 位元 u_short 的值由 network 排列方式轉換成 host 排列方式。

格　式： u_short PASCAL FAR ntohs(u_short netshort);
參　數： netshort　　一個 16 位元 network 排列方式的數目
傳回值： 一個 16 位元 host 排列方式的數目
說明： 因為 network 的排列方式與 host 的排列方式可能不同，所以我們需要此一函式來做轉換。

(16) **recv()**：自 Socket 接收資料。

格　式： int PASCAL FAR recv(SOCKET s,
　　char FAR *buf,
　　int len,
　　int flags);
參　數：s　　Socket 的識別碼
　　buf　　存放接收到的資料的暫存區
　　len　　buf 的長度
　　flags　　此函式被呼叫的方式
傳回值：成功 - 接收到的資料長度 (若對方 Socket 已關閉，則為 0)
　　失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)
說明： 此函式用來自連接式的 Datagram Socket 或 Stream Socket 接收資料。對 Stream Socket 言，我們可以接收到目前有效的 (available)資料，但其數量不超過 len 的大小。若是此 Socket 設定 SO_OOBINLINE，且有 out-of-band 的資料未被讀取，那麼只有 out-of-band 的資料被取出。對 Datagram Socket

言，只取出第一個 datagram；若是該 datagram 大於使用者提供的儲存空間，那麼只有該空間大小的資料被取出，多餘的資料將遺失，且回覆錯誤的訊息。 flags 的值可為 MSG_PEEK、MSG_OOB(*暫不提供此功能)的組合。

(17) **recvfrom()**：讀取一個 Datagram，並儲存資料來源的位址。

格　式： int PASCAL FAR recvfrom(SOCKET s,
char FAR *buf,
int len,
int flags,
struct socketaddr FAR *from,
int FAR *fromlen);

參　數：s Socket 的識別碼
buf 存放接收到的資料的暫存區
len buf 的長度
flags 此函式被呼叫的方式
from 資料來源的位址
fromlen from 的大小

傳回值：成功 - 接收到的資料長度 (若對方 Socket 已關閉，則為 0)
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明： 此函式用來讀取資料並記錄資料來源的位址。對 Stream Socket 言，其作用與 recv() 相同，參數 from 及 fromlen 將不被用到。

(18) **select()**：檢查一或多個 Sockets 是否處於可讀、可寫或錯誤的狀態。

格　式： int PASCAL FAR select(int nfds,
fd_set FAR *readfds,
fd_set FAR *writefds,
fd_set FAR *exceptfds,
const struct timeval FAR *timeout);

參　數：nfds 此參數在此並無作用
readfds 要被檢查是否可讀的 Sockets
writefds 要被檢查是否可寫的 Sockets
exceptfds 要被檢查是否有錯誤的 Sockets (*暫無作用)
timeout 此函式該等待的時間。若為 NULL 時，表示 blocking，此函式會等到有事件發生。

傳回值：成功 - 符合條件的 Sockets 總數 (若 Timeout 發生，則為 0)
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明： 使用者可利用此函式來檢查 Sockets 是否有資料可被讀取，或是有空間可以寫入，或是有錯誤發生。

(19) **send()**：使用連接式的 Socket 傳送資料。

格　式： int PASCAL FAR send(SOCKET s,
const char FAR *buf,
int len,
int flags);

參　數： s Socket 的識別碼
buf 存放要傳送的資料的暫存區
len buf 的長度
flags 此函式被呼叫的方式

傳回值： 成功 - 送出的資料長度
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明： 此函式適用於連接式的 Datagram 或 Stream Socket 來傳送資料。對 Datagram Socket 言，若是 datagram 的大小超過限制，則將不會送出任何資料，並會傳回錯誤值。若是傳送 (transport) 系統內之儲存空間不夠存放這些要傳送的資料，send() 將會被 block 住，除非該 Socket 被設定為 non-blocking 模式。使用者亦須注意 send()函式執行完成，並不表示資料已經成功地送抵對方了。flags 的值可設為 MSG_DONTROUTE(*暫不提供此功能)及 MSG_OOB 的組合。

(20) **sendto()**：將資料送到指定的目的地。

格　式： int PASCAL FAR sendto(SOCKET s,
const char FAR *buf,
int len,
int flags,
const struct sockaddr FAR *to,
int tolen);

參　數： s Socket 的識別碼
buf 存放要傳送的資料的暫存區
len buf 的長度
flags 此函式被呼叫的方式
to 資料要送達的位址
tolen to 的大小

傳回值： 成功 - 送出的資料長度
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明： 此函式適用於 Datagram 或 Stream Socket 來傳送資料到指定的位址。對 Datagram Socket 言，若是 datagram 的大小超過限制，則將不會送出任何資料，並會傳回錯誤值。對 Stream Socket 言，其作用與 send() 相同；參數 to 及 tolen 在此並無作用。 若是傳送 (transport) 系統內之儲存空間不夠存放這些要傳送的資料，sendto() 將會被 block 住，除非該 Socket 被設定為 non-blocking 模式。使用者亦須注意 sendto() 函式執行完成，並不表示資料已經成功地送抵對方了。 flags 的值可設為 MSG_DONTROUTE(*暫不提供此功能)及 MSG_OOB 的組合。

(21) **setsockopt()**：設定 Socket 的狀態。

格　式： int PASCAL FAR setsockopt(SOCKET s,
int level,
int optname,
const char FAR *optval,
int optlen);

參　數： s Socket 的識別碼
level 選項設定的 level
optname 選項名稱
optval 選項的設定值
optlen 選項設定值的長度

傳回值： 成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明： 此函式用來設定 Socket 的一些選項，藉以更改其動作。可更改的選項有：

Value	Type
SO_BROADCAST	BOOL
*SO_DEBUG	BOOL
SO_DONTLINGER	BOOL
*SO_DONTROUTE	BOOL
*SO_KEEPALIVE	BOOL
SO_LINGER	struct linger FAR*
SO_OOBINLINE	BOOL
*SO_RCVBUF	int
SO_REUSEADDR	BOOL
*SO_SNDBUF	int
TCP_NODELAY	BOOL

(* 表暫不提供此功能選項)

(22) **shutdown()**：停止 Socket 接收/傳送的功能。

格　式： int PASCAL FAR shutdown(SOCKET s, int how);
參　數：s　　Socket 的識別碼
　　how　　代表該停止那些動作的標幟
傳回值：成功 - 0
　　失敗 - SOCKET_ERROR (呼叫 WSAGetLastError()可得知原因)
說明： 此函式用來停止 Socket 的後續接收或傳送的功能。若 how 的值為 0，則不再接收資料。若 how 的值為 1，則不再允許傳送資料。若 how 的值為 2，則不再接收且不再傳送資料。shutdown() 函式並沒有將 Socket 關閉，所以該 Socket 所佔用之資源必須在呼叫 closesocket() 之後才會釋放。

(23) **socket()**：建立 Socket。

格　式： SOCKET PASCAL FAR socket(int af,
　　int type,
　　int protocol);
參　數：af　　目前只提供 PF_INET(AF_INET)
　　type　　Socket 的型態 (SOCK_STREAM、SOCK_DGRAM)
　　protocol 通訊協定(如果使用者不指定則設為 0)
傳回值：成功 - Socket 的識別碼
　　失敗 - INVALID_SOCKET(呼叫 WSAGetLastError() 可得知原因)
說明： 此函式用來建立一 Socket，並為此 Socket 建立其所使用的資源。Socket 的型態可為 Stream Socket 或 Datagram Socket。

(24) **gethostbyaddr()**：利用某一 host 的位址來獲取該 host 的資料。

格　式：struct hostent FAR * PASCAL FAR
　　gethostbyaddr(const char FAR *addr, int len, int type);
參　數：addr　　network 排列方式的位址
　　len　　addr 的長度
　　type　　PF_INET(AF_INET)
傳回值：成功 - 指向 struct hostent 的指標

```
struct hostent {
    char FAR *          h_name;
    char FAR * FAR *    h_aliases;
    short               h_addrtype;
```

```
        short           h_length;
        char FAR * FAR *  h_addr_list;
    }
```

失敗 - NULL (呼叫 WSAGetLastError() 可得知原因)

說明： 此函式是利用位址來獲取 host 的其他資料，如 host 的名稱、別名，位址的型態、長度等。

(25) **gethostbyname()**：利用某一 host 的名稱來獲取該 host 的資料。

格　式： struct hostent FAR * PASCAL FAR
gethostbyname(const char FAR *name);

參　數： name　　host 的名稱

傳回值：成功 - 指向 struct hostent 的指標

```
struct hostent {
    char FAR *          h_name;
    char FAR * FAR *    h_aliases;
    short               h_addrtype;
    short               h_length;
    char FAR * FAR *    h_addr_list;
}
```

失敗 - NULL (呼叫 WSAGetLastError() 可得知原因)

說明： 此函式是利用 host 名稱來獲取其他的資料，如 host 的位址、別名，位址的型態、長度等。

(26) **gethostname()**：獲取目前使用者使用的 host 的名稱。

格　式： int PASCAL FAR gethostname(char FAR *name, int namelen);

參　數：name　　用來存放 host 名稱的暫存區
namelen　name 的大小

傳回值：成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明： 此函式用來獲取 host 的名稱。

(27) **getprotobyname()**：依照通訊協定 (protocol) 的名稱來獲取該通訊協定的其他資料。

格　式： struct protoent FAR * PASCAL FAR
getprotobyname(const char FAR *name);

參　數： name　　通訊協定名稱

傳回值：成功 - 一指向 struct protoent 的指標

```
struct protoent {
    char FAR *          p_name;
    char FAR * FAR *    p_aliases;
    short               p_proto;
}
```

失敗 - NULL (呼叫 WSAGetLastError() 可得知原因)

說明： 利用通訊協定的名稱來得知該通訊協定的別名、編號等資料。

(28) **getprotobynumber()**：依照通訊協定的編號來獲取該通訊協定的其他資料。

格　式： struct protoent FAR * PASCAL FAR

```
getprotobynumber( int number );
```

參　數： number　　以 host 排列方式的通訊協定編號

傳回值：成功 - 一指向 struct protoent 的指標

```
struct protoent {
    char FAR *          p_name;
    char FAR * FAR *    p_aliases;
    short               p_proto;
}
```

失敗 - NULL (呼叫 WSAGetLastError() 可得知原因)

說明： 利用通訊協定的編號來得知該通訊協定的名稱、別名等資料。

(29) **getservbyname()**：依照服務 (service) 名稱及通訊協定來獲取該服務的其他資料。

格　式： struct servent * PASCAL FAR

```
getservbyname( const char FAR *name, const char FAR *proto );
```

參　數：name　　服務名稱

proto　　通訊協定名稱

傳回值： 成功 - 一指向 struct servent 的指標

```
struct servent {
    char FAR *          s_name;
    char FAR * FAR *    s_aliases;
    short               s_port;
    char FAR *          s_proto;
```

```
}
```

失敗 - NULL (呼叫 WSAGetLastError() 可得知原因)

說明：利用服務名稱及通訊協定來獲得該服務的別名、使用的 port 編號等。

(30) **getservbyport()**：依照服務 (service) 的 port 編號及通訊協定來獲取該服務的其他資料。

格　式： struct servent * PASCAL FAR
getservbyport(int port, const char FAR *proto);

參　數：port 服務的 port 編號
proto 通訊協定名稱

傳回值： 成功 - 一指向 struct servent 的指標

```
struct servent {
        char FAR *          s_name;
        char FAR * FAR *    s_aliases;
        short               s_port;
        char FAR *          s_proto;
}
```

失敗 - NULL (呼叫 WSAGetLastError() 可得知原因)

說明： 利用 port 編號及通訊協定來獲得該服務的名稱、別名等。

二、Microsoft Windows-specific Extensions

(1) **WSACleanup()**：結束 Windows Sockets DLL 的使用。

格　式： int PASCAL FAR WSACleanup(void);

參　數： 無

傳回值：成功 - 0
失敗 - SOCKET_ERROR (呼叫 WSAGetLastError() 可得知原因)

說明：應用程式在使用 Windows Sockets DLL 時必須先呼叫 WSAStartup() 來向 Windows Sockets DLL 註冊；當應用程式不再需要使用 Windows SocketsDLL 時，須呼叫此一函式來註銷使用，以便釋放其占用的資源。

(2) **WSAGetLastError()**：獲取最後一次錯誤發生時的訊息。

格　式： int PASCAL FAR WSAGetLastError(void);

參　數： 無

傳回值： Windows Sockets API 最後發生的錯誤碼

說明： 此函式用來獲取最後一次網路錯誤發生時的訊息。

(3) **WSAStartup()**：連結應用程式與 Windows Sockets DLL 的第一個函式。

格　式： int PASCAL FAR WSAStartup(WORD wVersionRequested,
LPWSADATA lpWSAData);

參　數： wVersionRequested　可使用的 Windows Sockets API 最高版本
lpWSAData　指向 WSADATA 資料的指標。系統傳回真正使用 WinSock 版本的資訊。

```
struct WSAData {
    WORD            wVersion;
    WORD            wHighVersion;
    char
szDescription[WSADESCRIPTION_LEN+1];
    char                szSystemStatus[WSASYSSTATUS_LEN+1];
    unsigned short      iMaxSockets;
    unsigned short      iMaxUdpDg;
    char FAR *          lpVendorInfo;
};
```

傳回值： 成功　0
失敗　WSASYSNOTREADY / WSAVERNOTSUPPORTED / WSAEINVAL

說明： 此函式「必須」是應用程式呼叫到 Windows Sockets DLL 函式中的第一個，也唯有此函式呼叫成功後，才可以再呼叫其他 Windows SocketsDLL 的函式。此函式亦讓使用者可以指定要使用的 Windows SocketsAPI 版本，及獲取設計者的一些資訊。