

Window Socket API

TS. Trần Quang Vinh
BM. Kỹ thuật Thông tin
Viện Điện tử - Viễn thông
Đại học Bách Khoa Hà Nội
vinh.tranquang1@hust. vn



Giới thiệu Winsock

- Windows Sockets API (Winsock API - WSA)
 - WSA là giao diện lập trình mạng, tương tự như Socket Linux với một số ngoại lệ như header files, các hàm của Winsock2 có thêm tiền tố WSA
 - Giao tiếp lập trình mạng cho phép phát triển ứng dụng giao tiếp trên cùng một máy hoặc nhiều máy khác nhau thông qua môi trường mạng
 - Winsock được hỗ trợ sẵn trong windows cho phép lập trình mạng với giao thức TCP/IP hoặc IPX
 - Lập trình Winsock trong windows sử dụng **winsock2.h** chứa các prototypes và thư viện **ws2_2.lib**

```
#include <stdio.h>
#include <winsock2.h>

int main()
{
    /**/
    return 0;
}
```

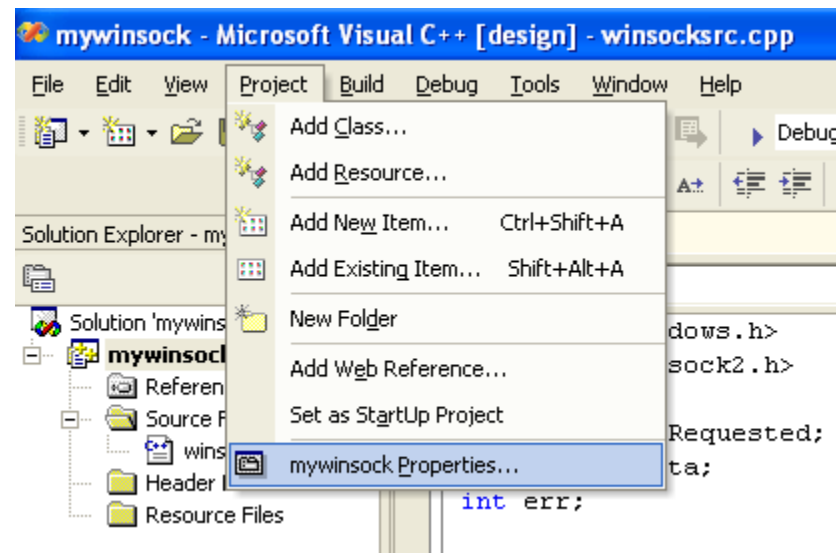
Giới thiệu Winsock

- Windows Sockets API (Winsock API - WSA)
 - Phiên bản winsock hỗ trợ cho các hệ điều hành Windows như sau:

Platform	Winsock Version
Windows 95	1.1 (2.2)
Windows 98	2.2
Windows Me	2.2
Windows NT 4.0	2.2
Windows 2000	2.2
Windows XP	2.2
Windows CE	1.1

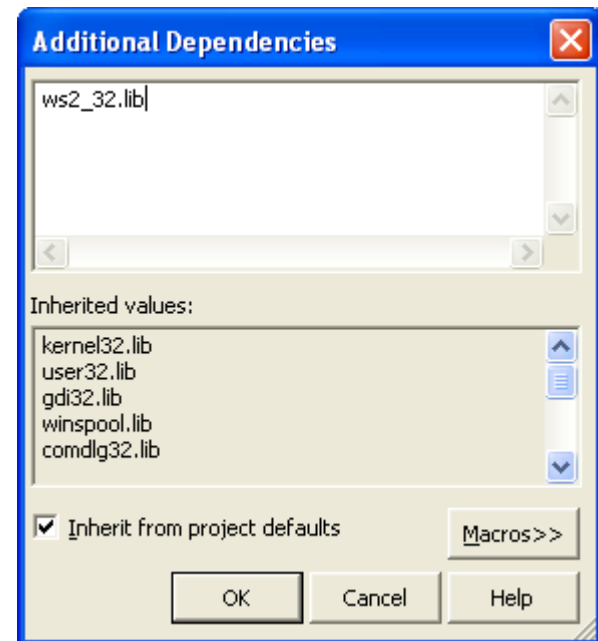
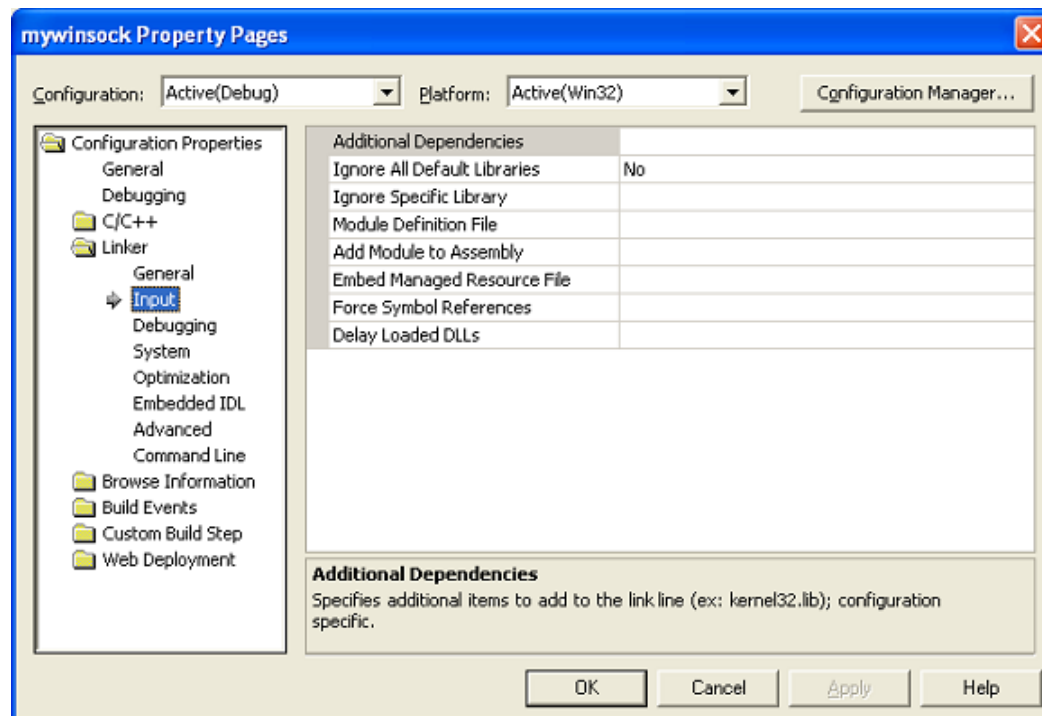
Compile and run Windows Sockets programs

- For Visual Studio .Net (Visual C++):
 - Select **Project** menu → **your_project_name Properties...** sub menu



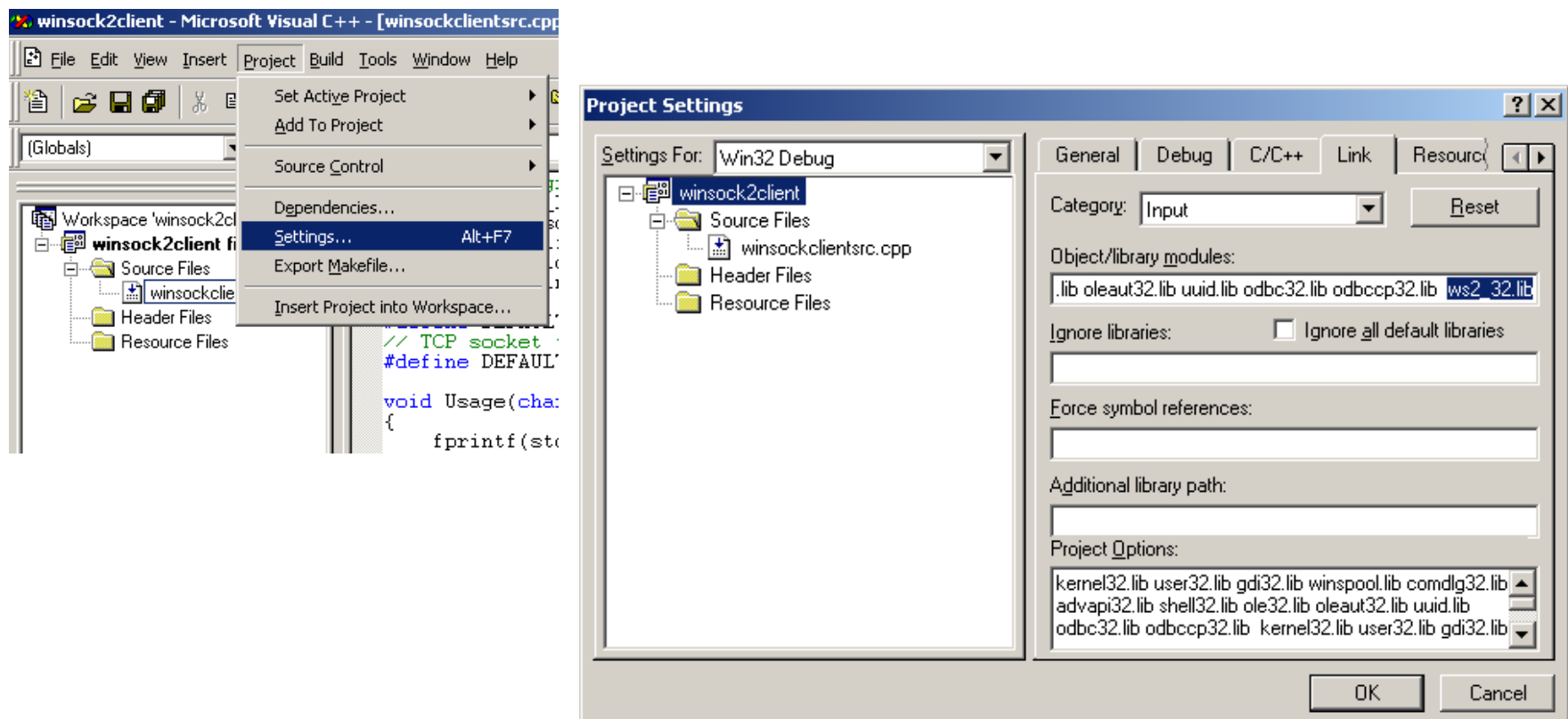
Compile and run Windows Sockets programs

- For Visual Studio .Net (Visual C++):
 - Expand **Linker** folder → select **Input** sub folder; For **Additional Dependencies**, select the right empty field and type the library name **ws2_32.lib**



Compile and run Windows Sockets programs

- Visual C++ 6.0
 - Select **Project** menu → **Settings...** sub menu



Quản lý địa chỉ Winsock

□ Địa chỉ IPv4

- Theo IPv4 , địa chỉ máy tính được thể hiện dưới dạng một số 32 bit
- Khi một client muốn kết nối với một server, nó phải biết địa chỉ của server và cổng kết nối
- Khi một server muốn nghe các yêu cầu từ client chuyển tới, chúng phải chỉ định 1 địa chỉ IP và 1 cổng kết nối cụ thể
- Trong Winsock, thông tin địa chỉ IP và cổng dịch vụ được chứa trong cấu trúc **sockaddr_in**

```
struct sockaddr_in {  
    short sin_family;          // họ địa chỉ Internet (AF_INET)  
    struct in_addr sin_addr;   // địa chỉ Internet  
    u_short sin_port;          // địa chỉ cổng, để nghe kết nối  
    char sin_zero[8];          // để dành 8 byte (không sử dụng)  
};
```

Quản lý địa chỉ Winsock

□ Cấu trúc `sockaddr_in`

□ `sin_addr`: là một structure để xác định địa chỉ IP

- ▶ Nếu `sin_addr.s_addr` = `INADDR_ANY`; → IP sẽ chính là IP nội bộ. Các Server thường dùng để chọn địa chỉ IP nội bộ và lắng nghe kết nối
- ▶ Nếu `sin_addr.s_addr` = `inet_addr("xx x.x.xx x.x.xx x.xxxx")`; → IP sẽ là IP chỉ định theo chuỗi. Client phải chỉ định được IP của Server mới kết nối được

□ `sin_port`: cổng (logic) để lắng nghe kết nối

- ▶ Card mạng sẽ thông qua số hiệu port này để nhận biết được gói tin đang nhận (hay gửi đi) của ứng dụng nào?
 - Các port dưới 1024 là những port dành riêng cho các dịch vụ như: 80 (Web), 20,21 (FTP), Mail (25 SMTP, 993 POP3), 53 (DNS), 23 (Telnet)...
- ▶ Dựa vào số port người quản trị có thể kiểm soát được mạng
 - ví dụ như chỉ có lướt WEB (mở port 80, 53) nhưng chặn hết các port khác → không thể chat và gameonline mặc dù có Internet
- ▶ Với số nguyên `u_short` (16bit) → có thể chọn tới $2^{16} = 65535$ port

□ `sin_zero`: Là vùng đệm để cấu trúc `SOCKADDR_IN` có cùng kích cỡ với `SOCKADDR`. Nó hữu dụng khi sử dụng hàm `inet_addr()` để convert một địa chỉ IP dạng chấm sang dạng `u_long` 32 bit

Các bước tạo một streaming TCP/IP

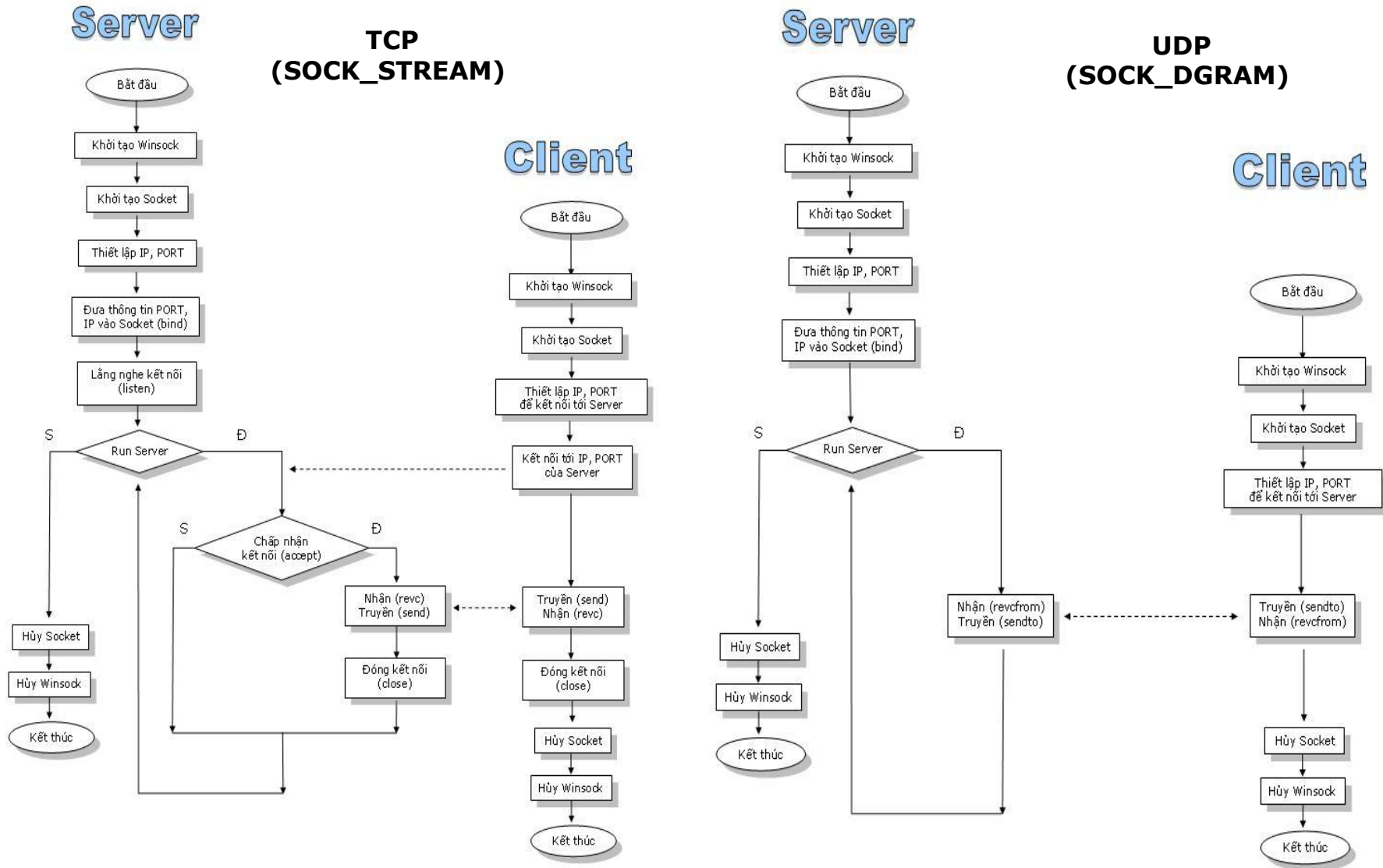
□ Server side programs

1. Initialize WSA – `WSAStartup()`
2. Create a socket – `socket()`
3. Bind the socket – `bind()`
4. Listen on the socket – `listen()`
5. Accept a connection – `accept()`, `connect()`
6. Send and receive data – `recv()`, `send()`, `recvfrom()`, `sendto()`
7. Disconnect – `closesocket()`

□ Client side programs

1. Initialize WSA – `WSAStartup()`
2. Create a socket – `socket()`
3. Connect to the server – `connect()`
4. Send and receive data – `recv()`, `send()`, `recvfrom()`, `sendto()`
5. Disconnect – `closesocket()`

Flowchart



Khởi động Winsock

(1) Tạo một object WSADATA

```
WSADATA wsaData;
```

- **WSADATA** là một structure chứa thông tin về việc thực thi Windows Sockets:

```
typedef struct WSADATA {  
    WORD        wVersion;           // Phiên bản hiện tại  
    WORD        wHighVersion;      // Phiên bản cao nhất có thể hỗ trợ  
    char        szDescription[WSADESCRIPTION_LEN + 1]; // Ghi chú  
    char        szSystemStatus[WSASYS_STATUS_LEN + 1]; // Trạng thái HT  
    unsigned short iMaxSockets;     // Không sử dụng từ Version 2 trở đi  
    unsigned short iMaxUdpDg;      // Không sử dụng từ Version 2 trở đi  
    char FAR*    lpVendorInfo;      // Không sử dụng từ Version 2 trở đi  
} WSADATA, *LPWSADATA;
```

Khởi động Winsock

(2) Gọi hàm **WSAStartup()** để khởi động thư viện **WS2_32.lib**

```
int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSADATA);
```

□ Tham số:

- **wVersionRequested** là phiên bản thư viện được sử dụng. Ở đây sẽ là giá trị 0x0202 có nghĩa là phiên bản 2.2. Có thể dùng macro MAKEWORD(2,2) để trả về giá trị 0x0202
- **lpWSADATA** là một số thông tin bổ sung **sẽ được trả về** sau khi gọi khởi tạo Winsock

□ Ví dụ:

```
wVersionRequested = MAKEWORD(2,2);  
int wsaerr = WSAStartup(wVersionRequested, &wsaData);  
if (wsaerr != 0){  
    printf("The Winsock dll not found!\n");  
}
```

□ Hủy Winsock

```
int WSACleanup (void);
```

Khởi động Winsock

```
#include <stdio.h>
#include <winsock2.h>

/* Create a Winsoc */
int main()
{
    WORD wVersionRequested = MAKEWORD(2,2);
    WSADATA wsaData;

    int wsaerr = WSASStartup(wVersionRequested, &wsaData);
    if (wsaerr != 0){
        printf("The Winsock dll not found!\n");
        return 0;
    }
    else {
        printf("The Winsock dll found!\n");
        printf("The status: %s.\n", wsaData.szSystemStatus);
    }
    WSACleanup();
    return 0;
}
```

Tạo một socket

(1) Khai báo một object kiểu SOCKET

```
SOCKET m_socket;
```

(2) Gọi hàm socket(), gán giá trị trả về cho m_socket

```
m_socket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
```

□ SOCKET là một cấu trúc để lưu giữ 1 Socket

```
SOCKET socket (  
    int af,  
    int type,  
    int protocol  
);
```

- ▶ **af**: họ địa chỉ giao thức, thiết lập là AF_INET nếu sử dụng IPv4, AF_NETBIOS: NetBIOS (Giao thức dùng tên máy để truyền dữ liệu), AF_APPLETALK: AppleTalk, AF_ATM: ATM
- ▶ **type**: kiểu giao thức của socket, thiết lập là SOCK_STREAM cho TCP/IP, SOCK_DGRAM cho UDP/IP, SOCK_RAW → protocol có thể là: IPPROTO_RAW hay IPPROTO_ICMP
- ▶ **protocol**: thiết lập là IPPROTO_TCP đối với TCP, IPPROTO_UDP đối với UDP

Tạo một socket

```
int main()
{
    /* Create a Winsoc */

    /* Create a SOCKET object called m_socket */
    SOCKET m_socket;

    // Call the socket function and return its value to the m_socket variable.
    // using AF_INET family, TCP socket type and protocol of the AF_INET - IPv4
    m_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    // Check for errors to ensure that the socket is a valid socket.
    if (m_socket == INVALID_SOCKET){
        printf("Error at socket(): %ld\n", WSAGetLastError());
        WSACleanup();
        return 0;
    }
    else{
        printf("socket() is OK!\n");
    }
    return 0;
}
```

Server: Binding a socket

□ Hàm bind()

- giúp cho SOCKET của SERVER biết rằng nó sẽ chờ đợi kết nối và nhận dữ liệu trên IP nào và PORT bao nhiêu?

```
int bind(  
    SOCKET s,                                // Socket được thiết lập  
    const struct sockaddr* name,             // Kích thước của cấu trúc sockaddr  
    int namelen  
);
```

- sockaddr* name: Cấu trúc ADDR bao gồm địa chỉ IP và PORT
- PORT ở đây nên ở trong khoảng nào?
 - ▶ 0 -1023: Là những PORT đã được sử dụng bởi các dịch vụ như WEB, FTP
 - ▶ 1024-49151: Là PORT dành cho SERVER lắng nghe. SERVER nên chọn trong khoảng này.
 - ▶ 49152-65535: Là PORT khởi tạo ngẫu nhiên dành cho CLIENT kết nối tới Server

Server: Binding a socket

(1) Tạo và thiết lập giá trị cho một object kiểu `sockaddr_in`

```
sockaddr_in service;  
  
service.sin_family = AF_INET;  
service.sin_addr.s_addr = inet_addr("127.0.0.1");  
service.sin_port = htons(55555);
```

(2) Gọi hàm `bind()` với tham số là `socket` và `sockaddr_in` vừa tạo

```
if (bind(m_socket, (SOCKADDR*)&service, sizeof(service)) == SOCKET_ERROR){  
    printf("bind() failed: %ld.\n", WSAGetLastError());  
    closesocket(m_socket);  
    return 0;  
}  
else {  
    printf("bind() is OK!\n");  
}
```

- `AF_INET`: họ địa chỉ Internet
- `"127.0.0.1"`: địa chỉ IP nội bộ gắn với socket
- `55555`: số hiệu cổng gắn với socket

Server: Binding a socket

```
int main()
{
    /* Create a Winsoc*/
    /* Create a SOCKET */

    /* Create a sockaddr_in object and set its values*/
    sockaddr_in service;
    service.sin_family = AF_INET;
    service.sin_addr.s_addr = inet_addr("127.0.1.1");
    service.sin_port = htons(55555);

    // Call the bind function, passing the created socket and the sockaddr_in
    // structure as parameters, Check for general errors.
    if (bind(m_socket, (SOCKADDR*)&service, sizeof(service)) == SOCKET_ERROR){
        printf("bind() failed: %ld.\n", WSAGetLastError());
        closesocket(m_socket);
        return 0;
    }
    else{
        printf("bind() is OK!\n");
    }
    return 0;
}
```

Server: Listening on a Socket

□ Hàm listen()

- Sau khi socket được bind đến một địa chỉ IP và cổng trên hệ thống, server bắt đầu lắng nghe trên địa chỉ IP và cổng cho các yêu cầu kết nối đến

```
int listen(  
    SOCKET s,          // Socket đã được thiết lập IP và PORT  
    int backlog  
);
```

- **Backlog:** Số kết nối cho phép chờ trong hàng đợi khi Server chưa chấp nhận kết nối. (vì đôi lúc có thể có tới 2 hay 3 client kết nối tới cùng 1 lúc). Giá trị tốt nhất là khoảng từ 5 – 10

□ Ví dụ

```
if ( listen( m_socket, 1 ) == SOCKET_ERROR )  
    printf( "Error listening on socket.\n");
```

Server: Listening on a Socket

```
int main()
{
    /* Create a Winsoc */
    /* Create a SOCKET */
    /* Call the bind function*/

    /* Call the listen function, passing the created socket and the maximum number of
    allowed, connections to accept as parameters. Check for general errors*/

    if (listen( m_socket, 1) == SOCKET_ERROR)
        printf("listen(): Error listening on socket %ld.\n", WSAGetLastError());
    else
    {
        printf("listen() is OK, I'm waiting for connections...\n");
    }
    return 0;
}
```

Server: Accepting a Connection

□ Hàm accept()

- Khi Client kết nối tới Server, nó phải chờ Server chấp nhận kết nối (nếu ở giao thức TCP) bằng hàm accept()

```
SOCKET accept(  
    SOCKET s,  
    struct sockaddr* addr,  
    int* addrlen  
);
```

- SOCKET s: Socket lắng nghe của SERVER.
- struct sockaddr addr: Là cấu trúc sockaddr lưu địa chỉ IP và PORT của CLIENT kết nối tới SERVER.
- int addrlen: Kích thước cấu trúc địa chỉ IP này.
- Hàm ACCEPT trả về 1 SOCKET mới, Socket mới được tạo này đại diện cho 1 Connection (kết nối) mới giữa Server và Client.
- Sau khi đã truyền dữ liệu, phải đóng SOCKET này lại bằng hàm closesocket(connect)

Server: Accepting a Connection

(1) Tạo một socket tạm thời để chấp nhận kết nối

```
SOCKET AcceptSocket;
```

(2) Tạo một loop để kiểm tra các yêu cầu kết nối, nếu một y/c kết nối đến, gọi hàm `accept()` để xử lý y/c này

```
printf( "Waiting for a client to connect...\n" );  
while (1) {  
    AcceptSocket = SOCKET_ERROR;  
    while ( AcceptSocket == SOCKET_ERROR ) {  
        AcceptSocket = accept( m_socket, NULL, NULL );  
    }  
}
```

(3) Khi kết nối với client được chấp nhận, chuyển điều khiển từ socket tạm thời sang socket ban đầu và dừng việc kiểm tra các kết nối mới

```
printf( "Client Connected.\n" );  
m_socket = AcceptSocket;  
break;  
}
```

Server: Accepting a Connection

```
int main()
{
    ...
    // Create a temporary SOCKET object called AcceptSocket for accepting connections
    SOCKET AcceptSocket;

    // Create a continuous loop that checks for connections requests. If a connection
    // request occurs, call the accept function to handle the request.
    printf("Server: Waiting for a client to connect...\n");
    printf("***Hint: Server is ready...run your client program...***\n");

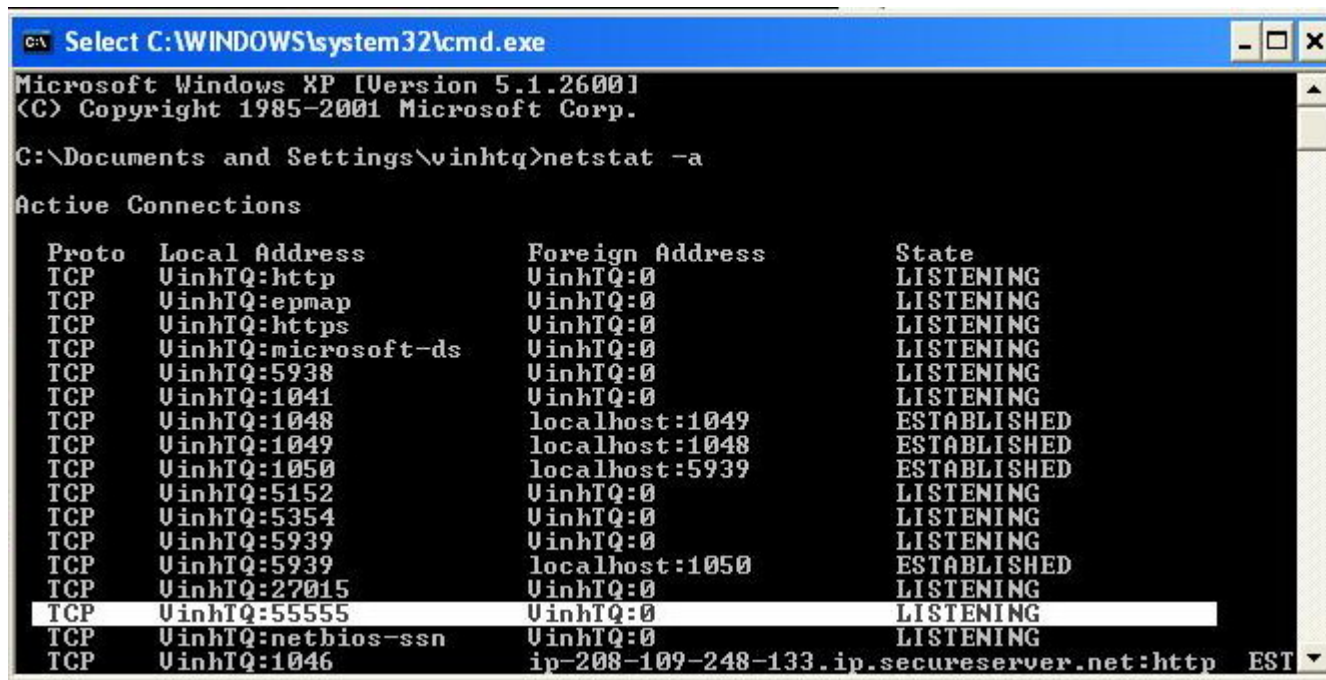
    // Do some verification...
    while (1){
        AcceptSocket = SOCKET_ERROR;
        while (AcceptSocket == SOCKET_ERROR){
            AcceptSocket = accept(m_socket, NULL, NULL);
        }

        // else, accept the connection...
        // When the client connection has been accepted, transfer control from the
        // temporary socket to the original socket and stop checking for new connections.
        printf("Server: Client Connected!\n");
        m_socket = AcceptSocket;
        break;
    }
    return 0;
}
```

Next Step: **Connecting to a Socket**

Server: Accepting a Connection

- We can verify the connection listening and waiting using netstat command on Windows console



```
C:\ Select C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\vinhtq>netstat -a

Active Connections

Proto Local Address           Foreign Address         State
TCP   UinhTQ:http              UinhTQ:0                LISTENING
TCP   UinhTQ:epmap             UinhTQ:0                LISTENING
TCP   UinhTQ:https             UinhTQ:0                LISTENING
TCP   UinhTQ:microsoft-ds     UinhTQ:0                LISTENING
TCP   UinhTQ:5938              UinhTQ:0                LISTENING
TCP   UinhTQ:1041              UinhTQ:0                LISTENING
TCP   UinhTQ:1048              localhost:1049           ESTABLISHED
TCP   UinhTQ:1049              localhost:1048           ESTABLISHED
TCP   UinhTQ:1050              localhost:5939           ESTABLISHED
TCP   UinhTQ:5152              UinhTQ:0                LISTENING
TCP   UinhTQ:5354              UinhTQ:0                LISTENING
TCP   UinhTQ:5939              UinhTQ:0                LISTENING
TCP   UinhTQ:5939              localhost:1050           ESTABLISHED
TCP   UinhTQ:27015             UinhTQ:0                LISTENING
TCP   UinhTQ:55555             UinhTQ:0                LISTENING
TCP   UinhTQ:netbios-ssn      UinhTQ:0                LISTENING
TCP   UinhTQ:1046              ip-208-109-248-133.ip.secureserver.net:http EST
```


Client: Connecting to a Socket

□ Hàm connect()

- Client muốn truyền thông trên mạng thì phải kết nối với một server

```
int connect(  
    SOCKET s,                // socket đã được khởi tạo  
    struct sockaddr *serv_addr, // IP và PORT của Server  
    int addrlen );           // Sizeof của cấu trúc sockaddr
```

Client: Connecting to a Socket

(1) Khai báo và thiết lập giá trị cho một object kiểu `sockaddr_in`

```
sockaddr_in clientService;  
  
clientService.sin_family = AF_INET;  
clientService.sin_addr.s_addr = inet_addr( "127.0.0.1" );  
/* 127.0.0.1 là địa chỉ của server mà client muốn kết nối */  
clientService.sin_port = htons( 27015 );  
/* 27015 là địa chỉ cổng gắn với server mà client muốn kết nối */
```

(2) Gọi hàm `connect()`, tham số là `socket` và `sockaddr_in` vừa tạo

```
if ( connect( m_socket, (SOCKADDR*) &clientService,  
sizeof(clientService) ) == SOCKET_ERROR)  
{  
    printf( "Failed to connect.\n" );  
    WSACleanup();  
    return;  
}
```

Sending and Receiving Data

□ Nhận dữ liệu trên giao thức TCP

```
int recv(  
    SOCKET s,  
    char * buf,  
    int len,  
    int flags  
);
```

□ Gửi dữ liệu giao thức TCP

```
int send(  
    SOCKET s,  
    const char * buf,  
    int len,  
    int flags  
);
```

- **SOCKET s**: Là SOCKET được tạo ra khi Server chấp nhận kết nối từ CLIENT
- **char * buf**: Là dữ liệu (dạng BYTE – char) nhận hay gửi
- **int len**: Kích thước của dữ liệu
- **int flags**: Một số cờ hiệu đi kèm (thông thường là 0)

Sending and Receiving Data

□ Nhận dữ liệu trên giao thức UDP

```
int recvfrom(  
    SOCKET s,                // SOCKET được tạo ra ban đầu  
    char * buf,              // dữ liệu (dạng BYTE - char) nhận  
    int len,                 // Kích thước của dữ liệu nhận  
    int flags,               // Một số cờ hiệu đi kèm (thông thường là 0)  
    struct sockaddr *from,   // IP và PORT từ bên gửi  
    int *fromlen             // Sizeof cấu trúc addr  
);
```

□ Gửi dữ liệu giao thức UDP

```
int sendto(  
    SOCKET s,                // SOCKET được tạo ra ban đầu  
    const char *buf,         // dữ liệu (dạng BYTE - char) gửi  
    int len,                 // Kích thước của dữ liệu gửi  
    int flags,               // Một số cờ hiệu đi kèm  
    const struct sockaddr* to, // IP và PORT từ bên gửi  
    int tolen                // Sizeof cấu trúc addr  
);
```

Sending and Receiving Data

- Demonstrates the send() and recv() functions (TCP packet)

- Server

```
int bytesSent;
int bytesRecv = SOCKET_ERROR;
char sendbuf[200] = "Hello! I'm server, sending some test data.";
char recvbuf[200] = "";

bytesRecv = recv(m_socket, recvbuf, 32, 0);
printf("Bytes Recv: %ld\n", bytesRecv);

bytesSent = send(m_socket, sendbuf, strlen(sendbuf), 0);
printf("Bytes Sent: %ld\n", bytesSent);
```

Sending and Receiving Data

- Demonstrates the send() and recv() functions (TCP packet)

- Client

```
int bytesSent;
int bytesRecv = SOCKET_ERROR;
char sendbuf[200] = "Hello! I'm client, sending some test data.";
char recvbuf[200] = "";

bytesSent = send(m_socket, sendbuf, strlen(sendbuf), 0);
printf("Bytes Sent: %ld\n", bytesSent);

while(bytesRecv == SOCKET_ERROR)
{
    bytesRecv = recv(m_socket, recvbuf, 32, 0);
    if (bytesRecv == 0 || bytesRecv == WSAECONNRESET)
    {
        printf("Connection Closed.\n");
        break;
    }
    if (bytesRecv < 0)
        return;
    printf("Bytes Recv: %ld\n", bytesRecv);
}
```

Close/Shutdown

- ❑ Hủy socket sau một kết nối hoặc kết thúc chương trình

```
int closesocket (SOCKET s);
```

```
int shutdown(  
    SOCKET s,  
    int how  
);
```

- ❑ Tham số **how** của hàm **shutdown()**:
 - ❑ **SD_RECEIVE**: Đóng SOCKET, không cho phép NHẬN nhưng cho phép GỬI
 - ❑ **SD_SEND**: Đóng SOCKET, không cho phép GỬI nhưng cho phép NHẬN
 - ❑ **SD_BOTH**: Không cho GỬI và NHẬN (giống gọi hàm closesocket)