Chương 3. Windows Socket

Lương Ánh Hoàng

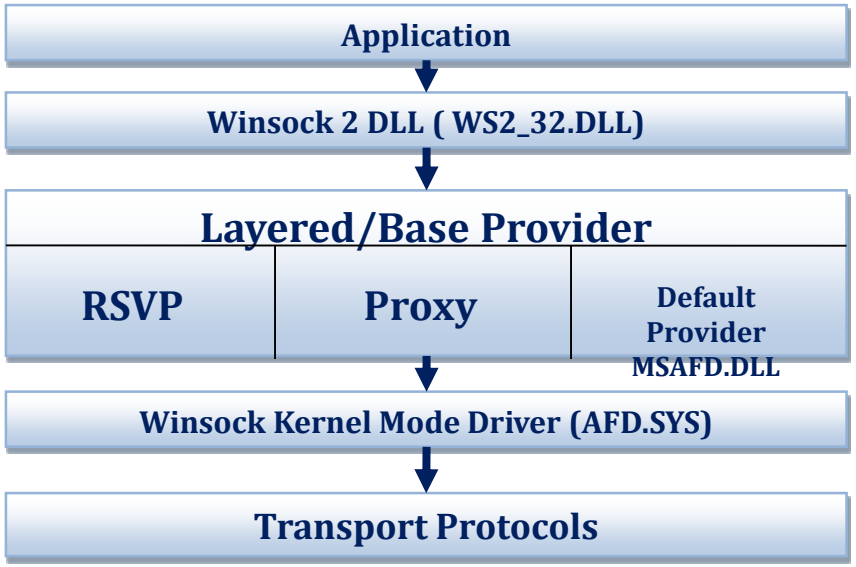
hoangla@soict.hut.edu.vn

Chương 3. Windows Socket

- 3.1. Kiến trúc
- 3.2. Đặc tính
- 3.3. Lập trình WinSock
- 3.4. Các phương pháp vào ra

3.1 Kiến trúc

- Windows Socket (WinSock)
  - Bộ thư viện liên kết động của Microsoft.
  - Cung cấp các API dùng để xây dựng ứng dụng mạng hiệu năng cao.



3.1 Kiến trúc

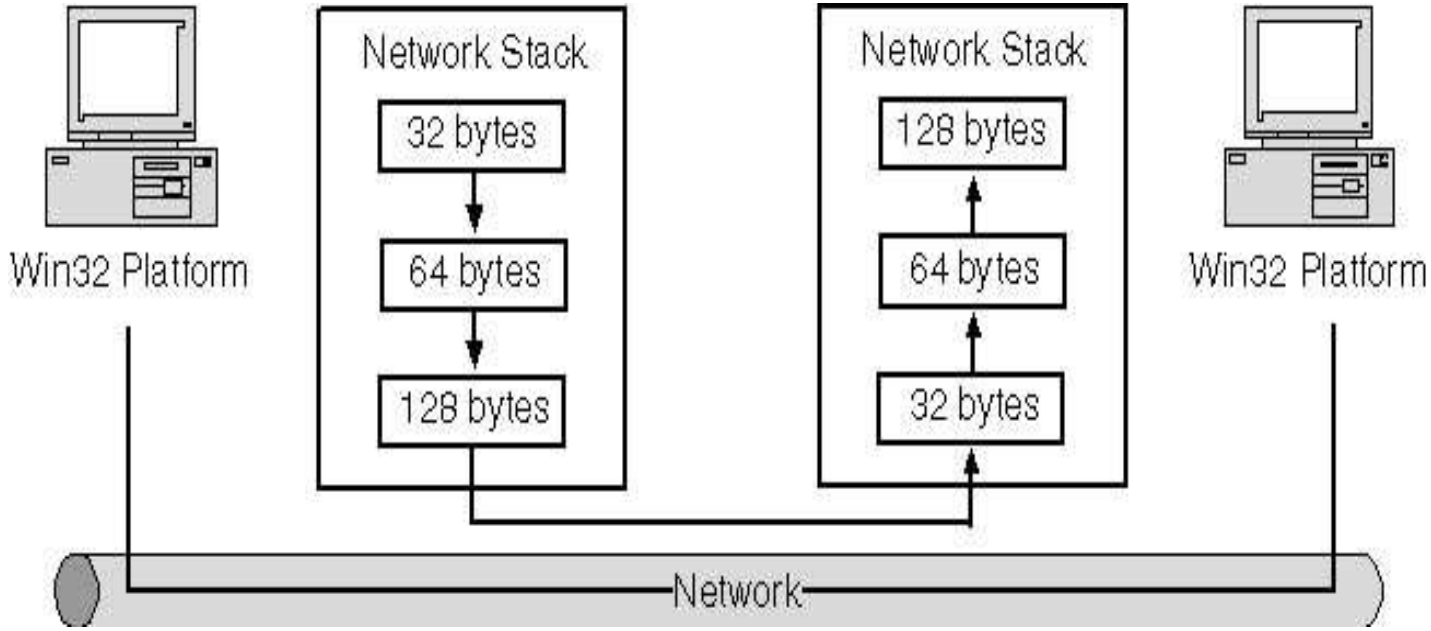
- Windows Socket (WinSock)
  - Phiên bản hiện tại là WinSock 2.2
  - Các ứng dụng sẽ giao tiếp với thư viện liên kết động ở tầng trên cùng: **WS2_32.DLL.**
  - **Provider** do nhà sản xuất của các giao thức cung cấp. Tầng này bổ sung giao thức của các tầng mạng khác nhau cho WinSock như TCP/IP, IPX/SPX, AppleTalk, NetBIOS...tầng này vẫn chạy ở **UserMode.**
  - **WinSock Kernel Mode Driver (AFD.SYS)** là driver chạy ở KernelMode, nhận dữ liệu từ tầng trên, quản lý kết nối, bộ đệm, tài nguyên liên quan đến socket và giao tiếp với driver điều khiển thiết bị.

3.1 Kiến trúc

- Windows Socket (WinSock)
  - **Transport Protocols** là các driver ở tầng thấp nhất, điều khiển trực tiếp thiết bị. Các driver này do nhà sản xuất phần cứng xây dựng, và giao tiếp với **AFD.SYS** thông qua giao diện TDI (Transport Driver Interface)
  - Việc lập trình Socket sẽ chỉ thao tác với đối tượng SOCKET.
  - Mỗi ứng dụng cần có một SOCKET trước khi muốn trao đổi dữ liệu với ứng dụng khác.
  - Đường dây ảo nối giữa các SOCKET sẽ là kênh truyền dữ liệu của hai ứng dụng.

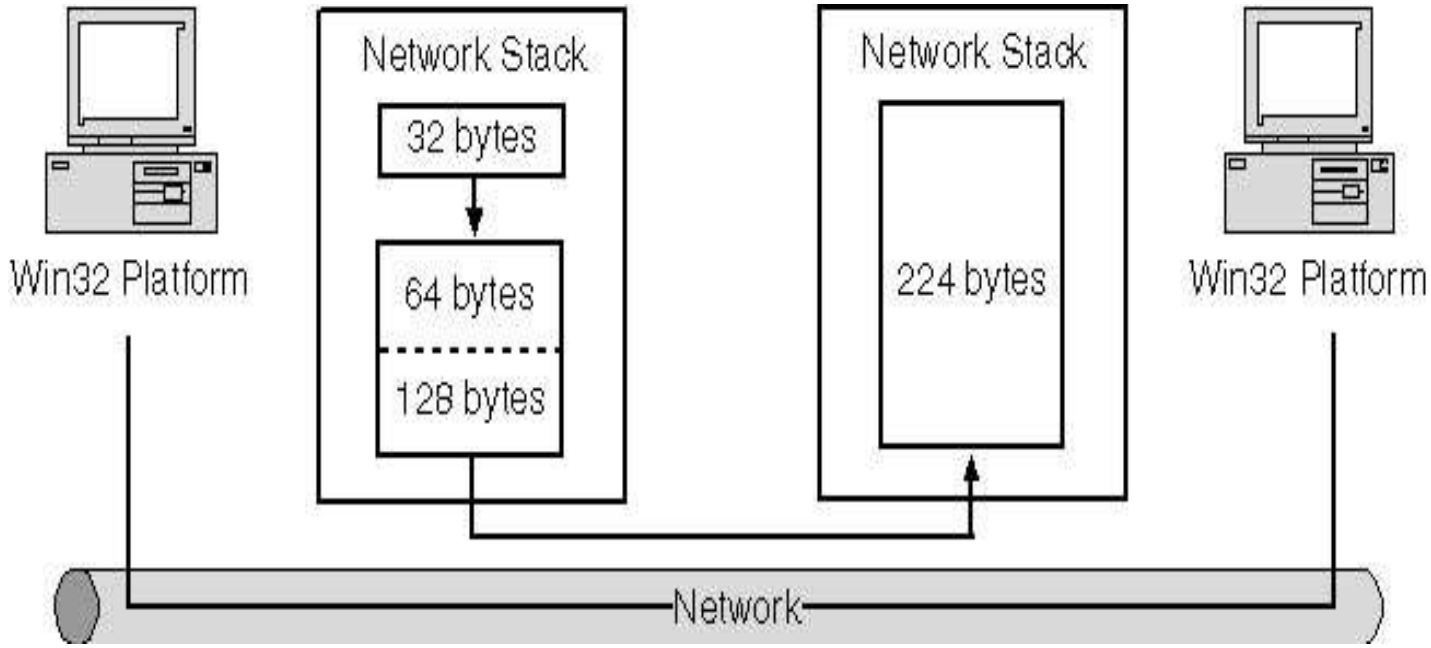
3.2 Đặc tính

- Hỗ trợ các giao thức hướng thông điệp (message oriented)
  - Thông điệp truyền đi được tái tạo nguyên vẹn cả về kích thước và biên ở bên nhận



3.2 Đặc tính

- Hỗ trợ các giao thức hướng dòng (stream oriented)
  - Biên của thông điệp không được bảo toàn khi truyền đi



3.2 Đặc tính

- Hỗ trợ các giao thức hướng kết nối và không kết nối
  - Giao thức hướng kết nối (connection oriented) thực hiện thiết lập kênh truyền trước khi truyền thông tin. Thí dụ: TCP
  - Giao thức không kết nối (connectionless) không cần thiết lập kênh truyền trước khi truyền. Thí dụ: UDP

3.2 Đặc tính

- Hỗ trợ các giao thức hướng kết nối và không kết nối
  - Giao thức hướng kết nối (connection oriented) thực hiện thiết lập kênh truyền trước khi truyền thông tin. Thí dụ: TCP
  - Giao thức không kết nối (connection less) không cần thiết lập kênh truyền trước khi truyền. Thí dụ: UDP

3.2 Đặc tính

- Hỗ trợ các giao thức tin cậy và trật tự
  - Tin cậy (reliability): đảm bảo chính xác từng byte được gửi đến đích.
  - Trật tự (ordering): đảm bảo chính xác trật tự từng byte dữ liệu. Byte nào gửi trước sẽ được nhận trước, byte gửi sau sẽ được nhận sau.

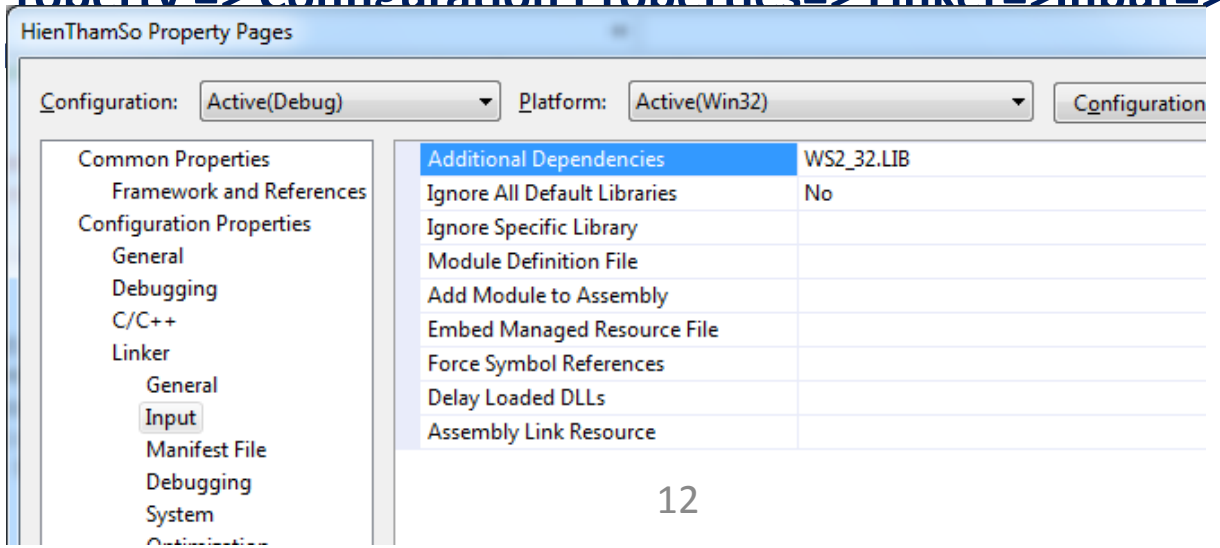
3.2 Đặc tính

- Multicast
  - WinSock hỗ trợ các giao thức Multicast: gửi dữ liệu đến một hoặc nhiều máy trong mạng.
- Chất lượng dịch vụ - Quality of Service (QoS)
  - Cho phép ứng dụng yêu cầu một phần băng thông dành riêng cho mục đích nào đó. Thí dụ: truyền hình thời gian thực.

3.3 Lập trình WinSock

- Chuẩn bị môi trường
  - Hệ điều hành Windows XP/2003/Vista/7.
  - Visual Studio C++
  - Thư viện trực tuyến MSDN
  - Thêm tiêu đề WINSOCK2.H vào đầu mỗi tệp mã nguồn.
  - Thêm thư viện WS2_32.LIB vào mỗi Project bằng cách

Project => Property => Configuration Properties=> Linker=>Input=>Additional Depende



3.3 Lập trình WinSock

- Khởi tạo WinSock
  - WinSock cần được khởi tạo ở đầu mỗi ứng dụng trước khi có thể sử dụng
  - Hàm WSAStartup sẽ làm nhiệm khởi tạo

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

    - wVersionRequested: [IN] phiên bản WinSock cần dùng.
    - lpWSAData: [OUT] con trỏ chứa thông tin về WinSock cài đặt trong hệ thống.
    - Giá trị trả về:
      - Thành công: 0
      - Thất bại: SOCKET_ERROR

3.3 Lập trình WinSock

- Khởi tạo WinSock
  - Thí dụ

```
WSADATA             wsaData;
WORD    wVersion = MAKEWORD(2,2); // Khởi tạo phiên bản 2.2
if (WSAStartup(wVersion,&wsaData))
{
            printf("Version not supported");
}
```

3.3 Lập trình WinSock

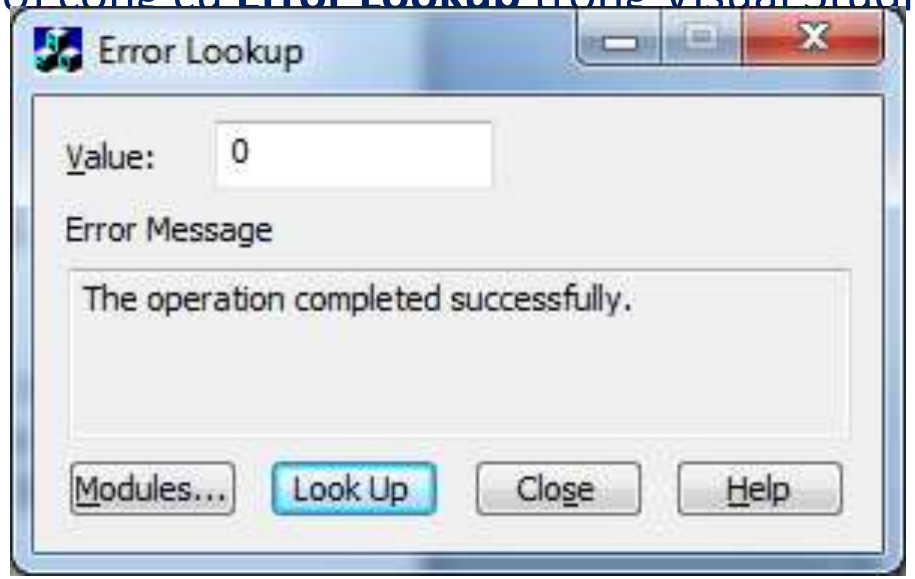
- Giải phóng WinSock
  - Ứng dụng khi kết thúc sử dụng WinSock có thể gọi hàm sau để giải phóng tài nguyên về cho hệ thống

    int WSACleanup(void);
  - Giá trị trả về:
    - Thành công: 0
    - Thất bại: SOCKET_ERROR

3.3 Lập trình WinSock

- Xác định lỗi
  - Phần lớn các hàm của WinSock nếu thành công đều trả về 0.
  - Nếu thất bại, giá trị trả về của hàm là SOCKET_ERROR.
  - Ứng dụng có thể lấy mã lỗi gần nhất bằng hàm

    int WSAGetLastError(void);
  - Tra cứu lỗi với công cụ **Error Lookup** trong Visual Studio



3.3 Lập trình WinSock

- Tạo SOCKET
  - SOCKET là một số nguyên trừu tượng hóa kết nối mạng của ứng dụng.
  - Ứng dụng phải tạo SOCKET trước khi có thể gửi nhận dữ liệu.
  - Hàm **socket** được sử dụng để tạo SOCKET

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

Trong đó:

- **af**: [IN] Address Family, họ giao thức sẽ sử dụng, thường là AF_INET, AF_INET6.
- **type**: [IN] Kiểu socket, SOCK_STREAM cho TCP/IP và SOCK_DGRAM cho UDP/IP.
- **protocol**: [IN] Giao thức tầng giao vận, IPPROTO_TCP hoặc IPPROTO_UDP

3.3 Lập trình WinSock

- Tạo SOCKET
  - Thí dụ

```
SOCKET      s1,s2; // Khai báo socket s1,s2

// Tạo socket TCP
s1 = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

// Tạo socket UDP
s2 = socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP);
```

3.3 Lập trình WinSock

- Xác định địa chỉ
  - WinSock sử dụng **sockaddr_in** để lưu địa chỉ của ứng dụng đích cần nối đến.
  - Ứng dụng cần khởi tạo thông tin trong cấu trúc này

```
struct sockaddr_in{
    short          sin_family; // Họ giao thức, thường là AF_INET
    u_short        sin_port;  // Cổng, dạng big-endian
    struct in_addr  sin_addr; // Địa chỉ IP
    char           sin_zero[8];     // Không sử dụng với IPv4
};
```

3.3 Lập trình WinSock

- Xác định địa chỉ
  - Sử dụng các hàm hỗ trợ :
    - Chuyển đổi địa chỉ IP dạng xâu sang số nguyên 32 bit
      unsigned long inet_addr(const char FAR *cp);
    - Chuyển đổi địa chỉ từ dạng **in_addr** sang dạng xâu
      char FAR *inet_ntoa(struct in_addr in);
    - Chuyển đổi little-endian => big-endian (network order)
      // Chuyển đổi 4 byte từ little-endian=>big-endian
      u_long htonl(u_long hostlong)
      // Chuyển đổi 2 byte từ little-endian=>big-endian
      u_short htons(u_short hostshort)
    - Chuyển đổi big-endian => little-endian (host order)
      // Chuyển 4 byte từ big-endian=>little-endian
      u_long ntohl(u_long netlong)
      // Chuyển 2 byte từ big-endian=>little-endian
      u_short ntohs(u_short netshort)

3.3 Lập trình WinSock

- Xác định địa chỉ
  - Thí dụ: điền địa chỉ 192.168.0.1:80 vào cấu trúc sockaddr_in

```
SOCKADDR_IN    InternetAddr; // Khai báo biến lưu địa chỉ

InternetAddr.sin_family = AF_INET;// Họ địa chỉ Internet

//Chuyển xâu địa chỉ 192.168.0.1 sang số 4 byte dang network-byte
// order và gán cho trường sin_addr
InternetAddr.sin_addr.s_addr = inet_addr("192.168.0.1");

//Chuyển đổi cổng sang dạng network-byte order và gán cho
trường // sin_port
InternetAddr.sin_port = htons(80);
```

3.3 Lập trình WinSock

- Phân giải tên miền
  - Đôi khi địa chỉ của máy đích được cho dưới dạng tên miền
  - Ứng dụng cần thực hiện phân giải tên miền để có địa chỉ thích hợp
  - Hàm **getnameinfo** và **getaddrinfo** sử dụng để phân giải tên miền
  - Cần thêm tệp tiêu đề WS2TCPIP.H

```
int getaddrinfo(
        const char  *nodename, // Tên miền hoặc địa chỉ cần phân giải
        const char  *servname,  //  Dịch vụ hoặc cổng
        const struct addrinfo  *hints, // Cấu trúc gợi ý
        struct addrinfo  **res  // Kết quả
);
```

    - Giá trị trả về
      - Thành công: 0
      - Thất bại: mã lỗi
    - Giải phóng: **freeaddrinfo()**

3.3 Lập trình WinSock

- Phân giải tên miền
  - Cấu trúc **addrinfo**: danh sách liên kết đơn chứa thông tin về tên miền tương ứng

```
struct addrinfo {
        int             ai_flags; // Thường là AI_CANONNAME
        int             ai_family; // Thường là AF_INET
        int             ai_socktype; // Loại socket
        int             ai_protocol; // Giao thứ giao vận
        size_t          ai_addrlen; // Chiều dài của ai_addr
        char            *ai_canonname; // Tên miền
        struct sockaddr *ai_addr;       // Địa chỉ socket đã phân giải
        struct addrinfo *ai_next;    // Con trỏ tới cấu trúc tiếp theo
};
```

3.3 Lập trình WinSock

- Phân giải tên miền
  - Đoạn chương trình sau sẽ thực hiện phân giải địa chỉ cho tên miền www.hut.edu.vn

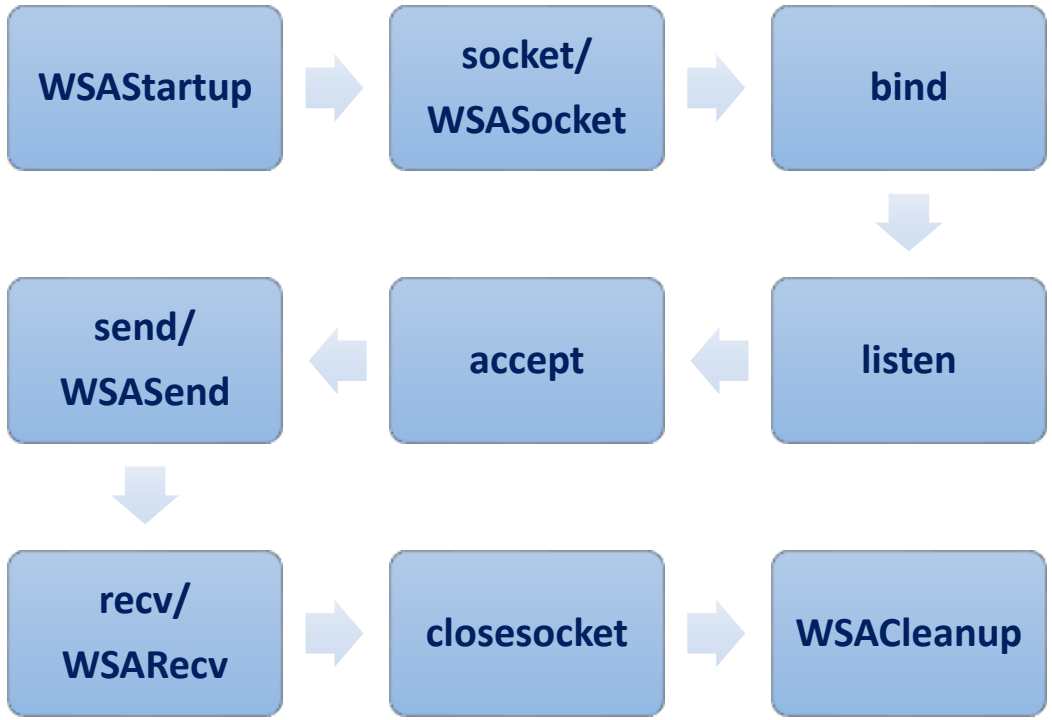
```
addrinfo   * result; // Lưu kết quả phân giải
int              rc;       // Lưu mã trả về
sockaddr_in     address; // Lưu địa chỉ phân giải được
rc = getaddrinfo("www.hut.edu.vn",  "http", NULL,  &result);

// Một tên miền có thể có nhiều địa chỉ IP tương ứng
// Lấy kết quả đầu tiên
if (rc==0)
      memcpy(&address,result->ai_addr,result->ai_addrlen);

// Xử lý với address...
```

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Việc truyền nhận dữ liệu sử dụng giao thức TCP sẽ bao gồm hai phần: ứng dụng phía client và phía server.
  - Ứng dụng phía server:
    - Khởi tạo WinSock qua hàm **WSAStartup**
    - Tạo SOCKET qua hàm **socket** hoặc **WSASocket**
    - Gắn SOCKET vào một giao diện mạng thông qua hàm **bind**
    - Chuyển SOCKET sang trạng thái đợi kết nối qua hàm **listen**
    - Chấp nhận kết nối từ client thông qua hàm **accept**
    - Gửi dữ liệu tới client thông qua hàm **send** hoặc **WSASend**
    - Nhận dữ liệu từ client thông qua hàm **recv** hoặc **WSARecv**
    - Đóng SOCKET khi việc truyền nhận kết thúc bằng hàm **closesocket**
    - Giải phóng WinSock bằng hàm **WSACleanup**

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía server (tiếp)



3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía server (tiếp)
    - Hàm **bind**: gắn SOCKET vào một giao diện mạng của máy

      int bind(SOCKET s, const struct sockaddr FAR* name, int namelen);

      Trong đó
      - **s: [IN]** SOCKET vừa được tạo bằng hàm socket
      - **name: [IN]** địa chỉ của giao diện mạng cục bộ
      - **namelen: [IN]** chiều dài của cấu trúc name

      Thí dụ

```
SOCKADDR_IN                 tcpaddr;
short                       port = 8888;
tcpaddr.sin_family = AF_INET;// Socket IPv4
tcpaddr.sin_port = htons(port); // host order => net order
tcpaddr.sin_addr.s_addr = htonl(INADDR_ANY); //Giao diện bất kỳ
 bind(s, (SOCKADDR *)&tcpaddr, sizeof(tcpaddr)); // Bind socket
```

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía server (tiếp)
    - Hàm **listen**: chuyển SOCKET sang trạng thái đợi kết nối
      int listen(SOCKET s, int backlog);

      Trong đó
      - **s**: [IN] SOCKET đã được tạo trước đó bằng socket/WSASocket
      - **backlog**: [IN] chiều dài hàng đợi chấp nhận kết nối

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía server (tiếp)
    - Hàm **accept**: chấp nhận kết nối

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

      Trong đó
      - **s**: **[IN]** SOCKET hợp lệ, đã được bind và listen trước đó
      - **addr: [OUT]** địa chỉ của client kết nối đến
      - **addrlen: [IN/OUT]** con trỏ tới chiều dài của cấu trúc addr. Ứng dụng cần khởi tạo addrlen trỏ tới một số nguyên chứa chiều dài của addr

      Giá trị trả về là một SOCKET mới, sẵn sàng cho việc gửi nhận dữ liệu trên đó. Ứng với mỗi kết nối của client sẽ có một SOCKET riêng.

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía server (tiếp)
    - Hàm **send**: gửi dữ liệu trên SOCKET

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

      Trong đó
      - **s**: **[IN]** SOCKET hợp lệ, đã được accept trước đó
      - **buf: [IN]** địa chỉ của bộ đệm chứa dữ liệu cần gửi
      - **len: [IN]** số byte cần gửi
      - **flags:[IN]** cờ quy định cách thức gửi, có thể là 0,MSG_OOB,MSG_DONTROUTE

      Giá trị trả về
      - Thành công: số byte gửi được, có thể nhỏ hơn **len**
      - Thất bại: SOCKET_ERROR

      Thí dụ

      char szHello[]=”Hello Network Programming”;
      send(s,szHello,strlen(szHello),0);

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía server (tiếp)
    - Hàm **recv**: nhận dữ liệu trên SOCKET

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

      Trong đó
      - **s**: **[IN]** SOCKET hợp lệ, đã được accept trước đó
      - **buf: [OUT]** địa chỉ của bộ đệm nhận dữ liệu
      - **len: [IN]** kích thước bộ đệm
      - **flags:[IN]** cờ quy định cách thức nhận, có thể là 0, MSG_PEEK, MSG_OOB, MSG_WAITALL

      Giá trị trả về
      - Thành công: số byte nhận được, có thể nhỏ hơn **len**
      - Thất bại: SOCKET_ERROR

      Thí dụ

      char buf[100];
      int len = 0;
      len = recv(s,buf,100,0);

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía server (tiếp)
    - Hàm **closesocket**: đóng kết nối trên một socket

      int closesocket(SOCKET *s*)

      Trong đó
      - **s**: **[IN]** SOCKET hợp lệ, đã kết nối

      Giá trị trả về
      - Thành công: 0
      - Thất bại: SOCKET_ERROR

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Đoạn chương trình minh họa

```
#include <winsock2.h>                    //Thu vien Winsock
void main(void)
{
  WSADATA          wsaData;
  SOCKET           ListeningSocket;
  SOCKET           NewConnection;
  SOCKADDR_IN      ServerAddr;
  SOCKADDR_IN      ClientAddr;
  int                                     ClientAddrLen;
  int              Port = 8888;
  // Khoi tao Winsock 2.2
  WSAStartup(MAKEWORD(2,2), &wsaData);
  // Tao socket lang nghe ket noi tu client.
   ListeningSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  // Khoi tao cau truc SOCKADDR_IN cua server
  // doi ket noi o cong 8888
    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_port = htons(Port);
    ServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Đoạn chương trình minh họa (tiếp)

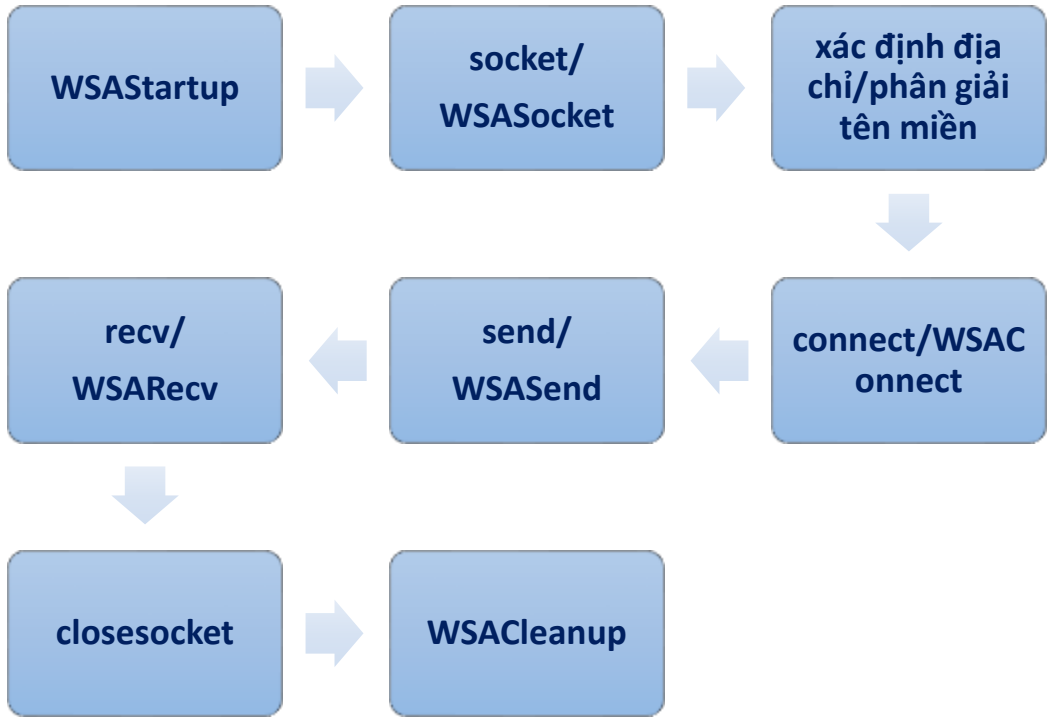
```
// Bind socket cua server.
    bind(ListeningSocket, (SOCKADDR *)&ServerAddr, sizeof(ServerAddr));
  // Chuyen sang trang thai doi ket noi
    listen(ListeningSocket, 5);
  // Chap nhan ket noi moi.
    ClientAddrLen = sizeof(ClientAddr);
    NewConnection = accept(ListeningSocket, (SOCKADDR *)
                  &ClientAddr,&ClientAddrLen);
  // Sau khi chap nhan ket noi, server co the tiep tuc chap nhan them cac ket noi khac,
  //  hoac gui nhan du lieu voi cac client thong qua cac socket duoc accept voi client
  // Dong socket
    closesocket(NewConnection);
    closesocket(ListeningSocket);
  // Giai phong Winsock
    WSACleanup();
}
```

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía client
    - Khởi tạo WinSock qua hàm **WSAStartup**
    - Tạo SOCKET qua hàm **socket** hoặc **WSASocket**
    - Điền thông tin về server vào cấu trúc **sockaddr_in**
    - Kết nối tới server qua hàm **connect** hoặc **WSAConnect**
    - Gửi dữ liệu tới server thông qua hàm **send** hoặc **WSASend**
    - Nhận dữ liệu từ server thông qua hàm **recv** hoặc **WSARecv**
    - Đóng SOCKET khi việc truyền nhận kết thúc bằng hàm **closesocket**
    - Giải phóng WinSock bằng hàm **WSACleanup**

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía client (tiếp)



3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Ứng dụng phía client (tiếp)
    - Địa chỉ của server xác định trong cấu trúc **sockaddr_in** nhờ hàm **inet_addr** hoặc theo **getaddrinfo**
    - Hàm **connect**: kết nối đến server
      int connect(SOCKET s,const struct sockaddr FAR* name,int namelen);

      Trong đó
      - **s: [IN]** SOCKET đã được tạo bằng **socket** hoặc **WSASocket** trước đó
      - **name:[IN]** địa chỉ của server
      - **namelen:[IN]** chiều dài cấu trúc **name**

      Giá trị trả về
      - Thành công: 0
      - Thất bại: SOCKET_ERROR

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Chương trình minh họa

```
#include <winsock2.h>
 void main(void)
{
  WSADATA              wsaData;
  SOCKET               s;
  SOCKADDR_IN          ServerAddr;
  int                  Port = 8888;
  // Khoi tao Winsock 2.2
   WSAStartup(MAKEWORD(2,2), &wsaData);
   // Tao socket client
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

  // Khoi tao cau truc SOCKADDR_IN co dia chi server la 202.191.56.69 va cong 8888

    ServerAddr.sin_family = AF_INET;
    ServerAddr.sin_port = htons(Port);
    ServerAddr.sin_addr.s_addr = inet_addr("202.191.56.69");
```

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng TCP
  - Chương trình minh họa (tiếp)

```
// Ket noi den server thong qua socket s.
    connect(s, (SOCKADDR *) &ServerAddr, sizeof(ServerAddr));

 // Bat dau gui nhan du lieu

 // Ket thuc gui nhan du lieu
 //  Dong socket
   closesocket(s);

 // Giai phong Winsock

   WSACleanup();
}
```

3.3 Lập trình WinSock

- Bài tập

  1. Viết chương trình TCPClient, kết nối đến một máy chủ xác định bởi tên miền hoặc địa chỉ IP. Sau đó nhận dữ liệu từ bàn phím và gửi đến Server. Tham số được truyền vào từ dòng lệnh có dạng

     TCPClient.exe <Địa chỉ IP/Tên miền> <Cổng>

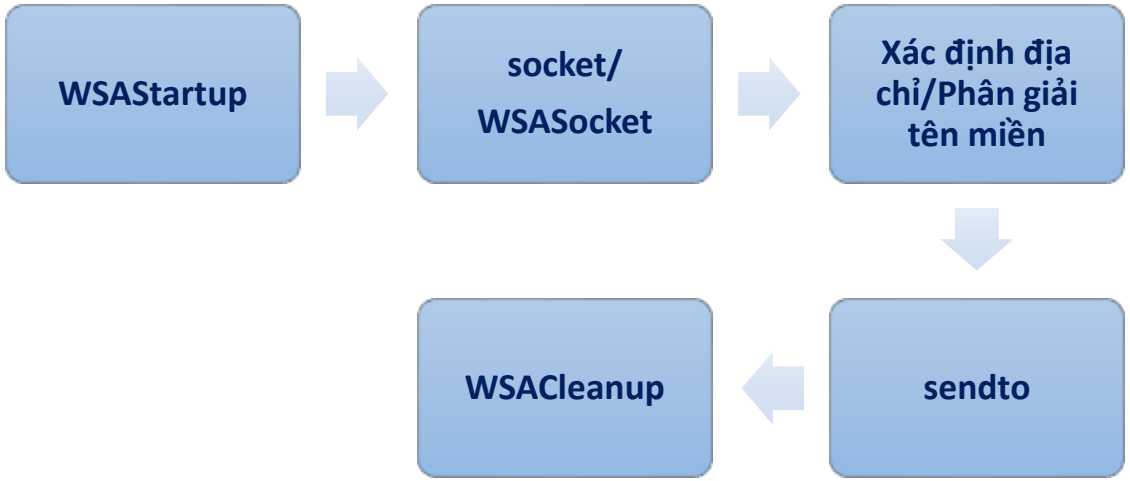
  2. Viết chương trình TCPServer, đợi kết nối ở cổng xác định bởi tham số dòng lệnh. Mỗi khi có client kết nối đến, thì gửi xâu chào được chỉ ra trong một tệp tin xác định, sau đó ghi toàn bộ nội dung client gửi đến vào một tệp tin khác được chỉ ra trong tham số dòng lệnh

     TCPServer.exe <Cổng> <Tệp tin chứa câu chào> <Tệp tin lưu nội dung client gửi đến>

     VD: TCPServer.exe 8888 chao.txt client.txt

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
  - Giao thức UDP là giao thức không kết nối (Connectionless)
  - Ứng dụng không cần phải thiết lập kết nối trước khi gửi tin.
  - Ứng dụng có thể nhận được tin từ bất kỳ máy tính nào trong mạng.
  - Trình tự gửi thông tin ở bên gửi như sau



3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
  - Ứng dụng bên gửi
    - Hàm **sendto**: gửi dữ liệu đến một máy tính bất kỳ

```
int sendto(
   SOCKET s,              // [IN] socket đã tạo bằng hàm socket/WSASocket
   const char FAR * buf,  // [IN] bộ đệm chứa dữ liệu cần gửi
   int len,               // [IN] số byte cần gửi
   int flags,             // [IN] cờ, tương tự như hàm send
   const struct sockaddr FAR * to,   // [IN] địa chỉ đích
   int tolen              // [IN] chiều dài địa chỉ đích
);
```

Giá trị trả về

- Thành công: số byte gửi được, có thể nhỏ hơn **len**
- Thất bại: SOCKET_ERROR

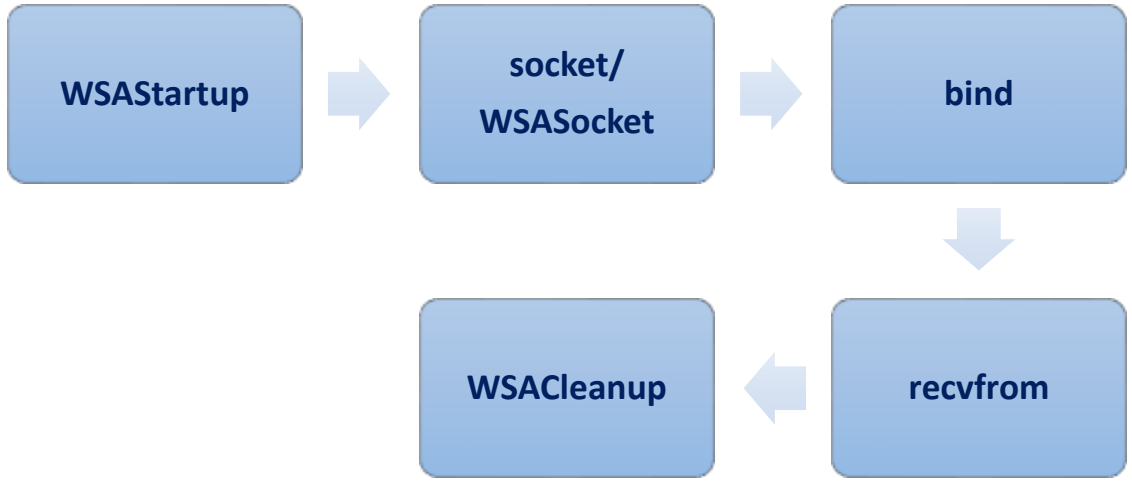
3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
  - Đoạn chương trình sau sẽ gửi một xâu tới địa chỉ 202.191.56.69:8888

```
char        buf[]=”Hello Network Programming”; // Xâu cần gửi
SOCKET    sender; // SOCKET để gửi
SOCKADDR_IN receiverAddr; // Địa chỉ nhận
// Tạo socket để gửi tin
sender = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
 // Điền địa chỉ đích
receiverAddr.sin_family = AF_INET;
receiverAddr.sin_port = htons(8888);
receiverAddr.sin_addr.s_addr = inet_addr("202.191.56.69");
// Thực hiện gửi tin
sendto(sender, buf, strlen(buf), 0,
        (SOCKADDR *)&receiverAddr, sizeof(receiverAddr));
```

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
  - Trình tự nhận thông tin ở bên nhận như sau



3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
  - Ứng dụng bên nhận
    - Hàm **recvfrom**: nhận dữ liệu từ một socket

```
int recvfrom(
   SOCKET s,          // [IN] SOCKET sẽ nhận dữ liệu
   char FAR* buf,     // [IN] địa chỉ bộ đệm chứa dữ liệu sẽ nhận được
   int len,           // [IN] kích thước bộ đệm
   int flags,         // [IN] cờ, tương tự như hàm recv
   struct sockaddr FAR* from,// [OUT] địa chỉ của bên gửi
   int FAR* fromlen // [IN/OUT] chiều dài cấu trúc địa chỉ của bên
                    //  gửi, khởi tạo là chiều dài của from
);
```

Giá trị trả về

- Thành công: số byte nhận được
- Thất bại: SOCKET_ERROR

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
  - Đoạn chương trình sau sẽ nhận đữ liệu datagram từ cổng 8888 và hiển thị ra màn hình

```
SOCKET              receiver;
SOCKADDR_IN         addr, source;
int                 len = sizeof(source);
// Tạo socket UDP
receiver = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
// Khởi tạo địa chỉ và cổng 8888
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(8888); // Đợi UDP datagram ở cổng 8888

// Bind socket vào tất cả các giao diện và cổng 8888
bind(receiver,(sockaddr*)&addr,sizeof(SOCKADDR_IN));
```

3.3 Lập trình WinSock

- Truyền dữ liệu sử dụng UDP
  - Đoạn chương trình (tiếp)

```
// Lặp đợi gói tin
while (1)
{
        // Nhận dữ liệu từ mạng
        datalen = recvfrom(receiver,buf,100,0,(sockaddr*)&source,
                                &len);
        // Kiểm tra chiều dài
        if (datalen>0)
        {
                buf[datalen]=0;
                printf("Data:%s",buf); // Hiển thị ra màn hình
        }
}
```

3.3 Lập trình WinSock

- Sử dụng Netcat để gửi nhận dữ liệu đơn giản
  - Netcat là một tiện ích mạng rất đa năng.
  - Có thể sử dụng như TCP server: nc.exe -v -l -p *<cổng đợi kết nối>*
    Thí dụ: nc.exe -l -p 8888
  - Có thể sử dụng như TCP client: nc -v *<ip/tên miền>* *<cổng>*
    Thí dụ: nc.exe 127.0.0.1 80
  - Sử dụng như UDP receiver: nc -v -l -u -p *<cổng đợi kết nối>*
    Thí dụ: nc.exe -v -l -u -p 8888
  - Sử dụng như UDP sender: nc -v -u *<ip/tên miền>* *<cổng>*
  - Thí dụ: nc.exe -v -u 192.168.0.1 80

3.3 Lập trình WinSock

- Một số hàm khác
  - getpeername: lấy địa chỉ đầu kia mà SOCKET kết nối đến

```
int getpeername(
   SOCKET s,                          // [IN] SOCKET cần lấy địa chỉ
   struct sockaddr FAR* name,  // [OUT] địa chỉ lấy được
   int FAR* namelen                   // [OUT] chiều dài địa chỉ
);
```

  - getsockname: lấy địa chỉ cục bộ của SOCKET

```
int getsockname(
   SOCKET s,                          // [IN] SOCKET cần lấy địa chỉ
   struct sockaddr FAR* name, // [OUT] địa chỉ lấy được
    int FAR* namelen    //  [OUT] chiều dài địa chỉ
);
```

3.3 Lập trình WinSock

- Bài tập

  1. Viết chương trình **clientinfo** thực hiện kết nối đến một máy chủ xác định và gửi thông tin về tên máy, danh sách các ổ đĩa có trong máy, kích thước các ổ đĩa. Địa chỉ (tên miền) và cổng nhận vào từ tham số dòng lệnh.

     VD: clientinfo abc.com 1234

  2. Viết chương trình **serverinfo** đợi kết nối từ **các clientinfo** và thu nhận thông tin từ client, hiện ra màn hình. Tham số dòng lệnh truyền vào là cổng mà serverinfo sẽ đợi kết nối

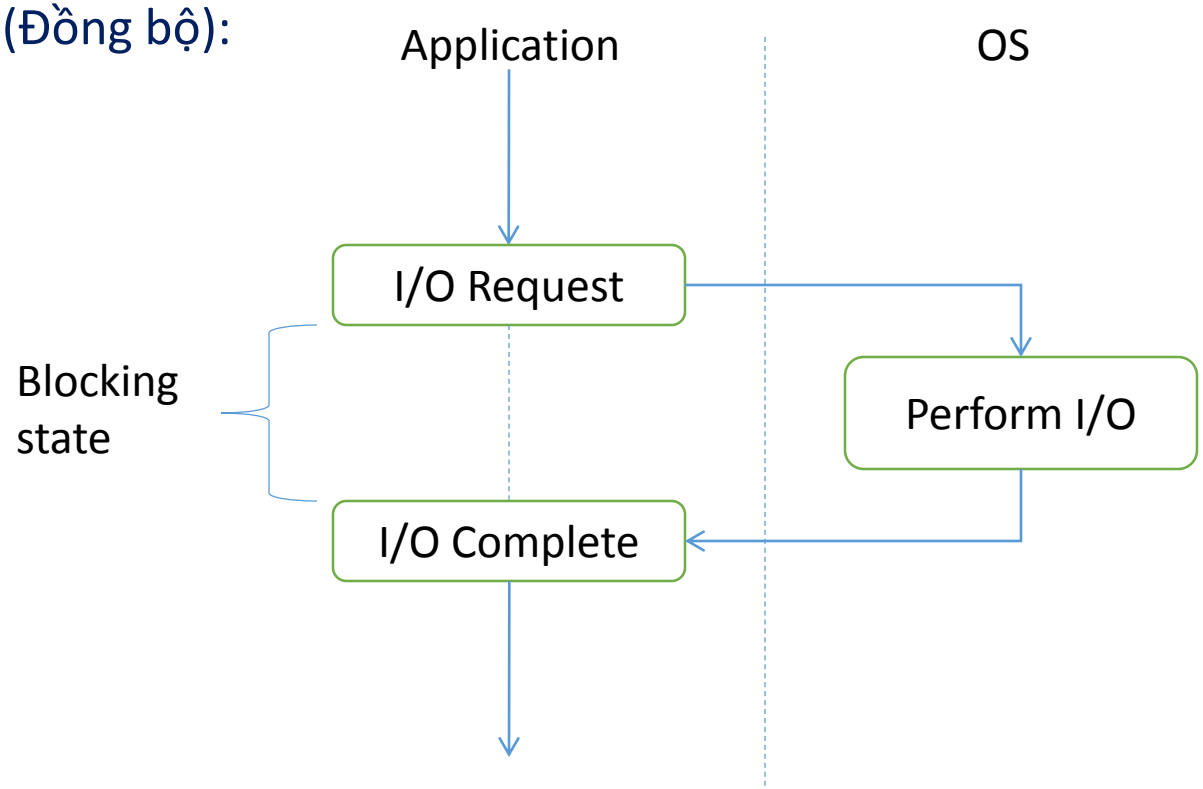
     VD: serverinfo 1234

3.4 Các phương pháp vào ra

- Các chế độ hoạt động của WinSock
  - Thread(Luồng):
    - Là đơn vị thực thi độc lập và tuần tự của chương trình.
    - Mỗi chương trình có ít nhất một thread chính là thread bắt đầu thực hiện tại hàm **main**
  - Blocking (Đồng bộ):
    - Là chế độ mà các hàm vào ra sẽ chặn thread đến khi thao tác vào ra hoàn tất (các hàm vào ra sẽ không trở về cho đến khi thao tác hoàn tất).
    - Là chế độ mặc định của SOCKET
    - Các hàm ảnh hưởng:
      - **accept**
      - **connect**
      - **send**
      - **recv**
      - **...**

3.4 Các phương pháp vào ra

- Các chế độ hoạt động của WinSock
  - Blocking (Đồng bộ):



3.4 Các phương pháp vào ra

- Các chế độ hoạt động của WinSock
  - Blocking (Đồng bộ):
    - Thích hợp với các ứng dụng xử lý tuần tự. Không nên gọi các hàm blocking khi ở thread xử lý giao diện (GUI Thread).
    - Thí dụ:
      - Thread bị chặn bởi hàm **recv** thì không thể gửi dữ liệu

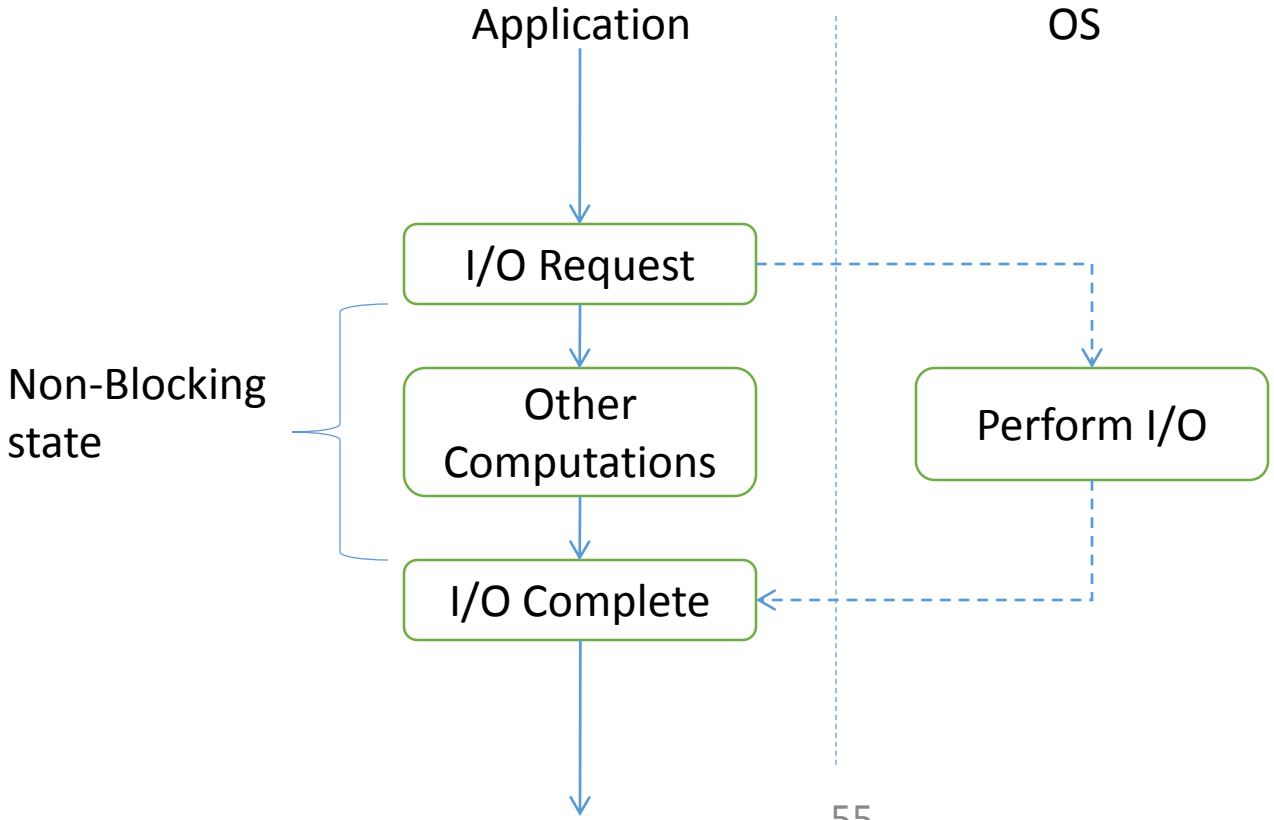
```
do
{
        // Thread sẽ bị chặn lại khi gọi hàm recvfrom
        // Trong lúc đợi dữ liệu thì không thể gửi dữ liệu
        rc = recvfrom(receiver,szXau,128,0,
                (sockaddr*)&senderAddress,&senderLen);
        //      ..
}while
...
```

3.4 Các phương pháp vào ra

- Các chế độ hoạt động của WinSock
  - Non-Blocking (Bất đồng bộ):
    - Là chế độ mà các thao tác vào ra sẽ trở về nơi gọi ngay lập tức và tiếp tục thực thi thread. Kết quả của thao tác vào ra sẽ được thông báo cho chương trình dưới một cơ chế đồng bộ nào đó.
    - Các hàm vào ra bất đồng bộ sẽ trả về mã lỗi **WSAWOULDBLOCK** nếu thao tác đó không thể hoàn tất ngay và mất thời gian đáng kể(chấp nhận kết nối, nhận dữ liệu, gửi dữ liệu...). Đây là điều hoàn toàn bình thường.
    - Có thể sử dụng trong thread xử lý giao diện của ứng dụng.
    - Thích hợp với các ứng dụng hướng sự kiện.

3.4 Các phương pháp vào ra

- Các chế độ hoạt động của WinSock
  - Non-Blocking (Bất đồng bộ):



3.4 Các phương pháp vào ra

- Các chế độ hoạt động của WinSock
  - Non-Blocking (Bất đồng bộ):
    - Socket cần chuyển sang chế độ này bằng hàm **ioctlsocket**

```
SOCKET      s;
unsigned long ul = 1;
int         nRet;

s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
// Chuyển sang chế độ non-blocking
nRet = ioctlsocket(s, FIONBIO, (unsigned long *) &ul);
if (nRet == SOCKET_ERROR)
{
   // Thất bại
}
```

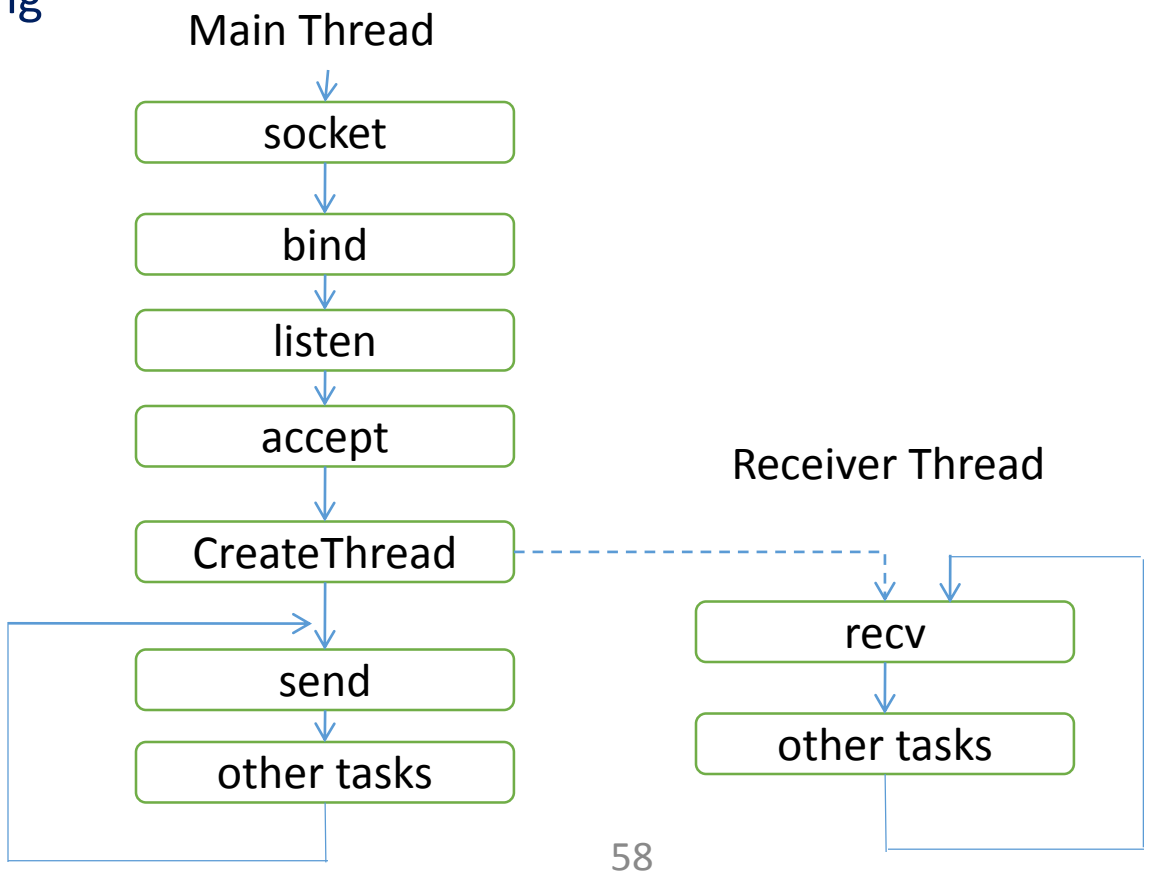
3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Blocking
    - Mô hình mặc định, đơn giản nhất.
    - Không thể gửi nhận dữ liệu đồng thời trong cùng một luồng.
    - Chỉ nên áp dụng trong các ứng dụng đơn giản, xử lý tuần tự, ít kết nối.
    - Giải quyết vấn đề xử lý song song bằng việc tạo thêm các thread chuyên biệt: thread gửi dữ liệu, thread nhận dữ liệu
    - Hàm API **CreateThread** được sử dụng để tạo một luồng mới

      HANDLE WINAPI CreateThread(
      __in LPSECURITY_ATTRIBUTES lpThreadAttributes,
      __in SIZE_T dwStackSize,
      __in LPTHREAD_START_ROUTINE lpStartAddress,
      __in LPVOID lpParameter,
      __in DWORD dwCreationFlags,
      __out LPDWORD lpThreadId);
    - **TerminateThread**
    - BOOL WINAPI TerminateThread(__in_out HANDLE *hThread*,
      __in DWORD *dwExitCode*);
  - Trên unix/linux/posix: **pthread_create** và **pthread_kill**

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Blocking



3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Blocking
    - Đoạn chương trình sau sẽ minh họa việc gửi và nhận dữ liệu đồng thời trong TCP Client

```
// Khai báo luồng xử lý việc nhận dữ liệu
DWORD WINAPI ReceiverThread(LPVOID lpParameter);
...
// Khai báo các biến toàn cục
SOCKADDR_IN address;
SOCKET                  client;
char                    szXau[128];
...
rc = connect(client,(sockaddr*)&address,sizeof(address));
// Tạo luồng xử lý việc nhận dữ liệu
CreateThread(0,0,ReceiverThread,0,0,0);
while (strlen(gets(szXau))>=2)
{
        rc = send(client,szXau,strlen(szXau),0);
}
...
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Blocking
    - Đoạn chương trình (tiếp)

```
DWORD WINAPI ReceiverThread(LPVOID lpParameter)
{
        char    szBuf[128];
        int     len = 0;
        do
        {
                len = recv(client,szBuf,128,0);
                if (len>=2)
                {
                        szBuf[len] = 0;
                        printf("%s\n",szBuf);
                }
                else
                        break;
        }while (len>=2);
}
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình **Select**
    - Là mô hình được sử dụng phổ biến.
    - Sử dụng hàm **select** để thăm dò các sự kiện trên socket (gửi dữ liệu, nhận dữ liệu, kết nối thành công, yêu cầu kết nối...).
    - Hỗ trợ nhiều kết nối cùng một lúc.
    - Có thể xử lý tập trung tất cả các socket trong cùng một thread (tối đa 1024).
    - Nguyên mẫu hàm như sau

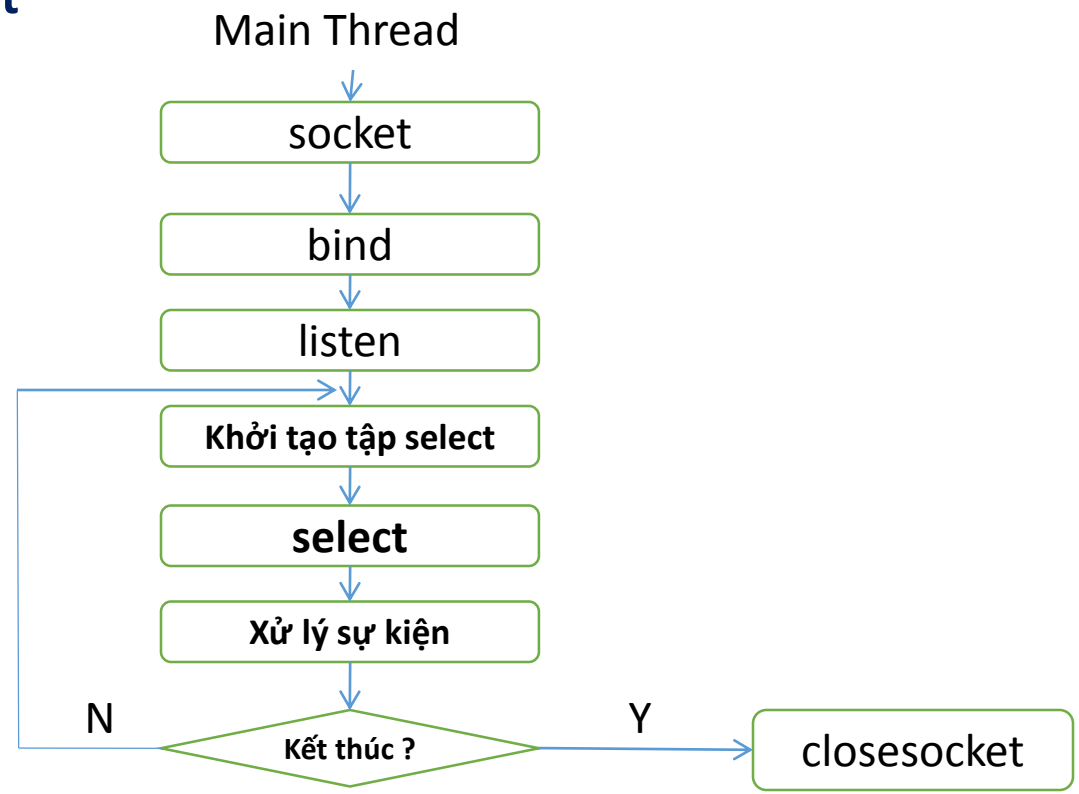
```
int select(
  int nfds,                 // Không sử dụng
  fd_set FAR * readfds,     //  Tập các socket hàm sẽ thăm dò cho sự kiện read
  fd_set FAR * writefds,    //  Tập các socket hàm sẽ thăm dò cho sự kiện write
  fd_set FAR * exceptfds,   //  Tập các socket hàm sẽ thăm dò cho sự kiện except
  const struct timeval FAR * timeout // Thời gian thăm dò tối đa
);
```

Giá trị trả về:

- Thành công: số lượng socket có sự kiện xảy ra
- Hết giờ: 0
- Thất bại: SOCKET_ERROR

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình **Select**



3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Select
    - Điều kiện thành công của **select**
      - Một trong các socket của tập readfds nhận được dữ liệu hoặc kết nối bị đóng, reset, hủy, hoặc hàm accept thành công.
      - Một trong các socket của tập writefds có thể gửi dữ liệu, hoặc hàm connect thành công trên socket non-blocking.
      - Một trong các socket của tập exceptfds nhận được dữ liệu OOB, hoặc connect thất bại.
    - Các tập readfds, writefds, exceptfds có thể NULL, nhưng không thể cả ba cùng NULL.
    - Các MACRO FD_CLR, FD_ZERO, FD_ISSET, FD_SET sử dụng để thao tác với các cấu trúc fdset.

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Select
    - Đoạn chương trình sau sẽ thăm dò trạng thái của socket s khi nào có dữ liệu

```
SOCKET s;
fd_set  fdread;
int     ret;
// Khởi tạo socket s và tạo kết nối
 ...
// Thao tác vào ra trên socket s
while(TRUE)
{
   // Xóa tập fdread
   FD_ZERO(&fdread);
   // Thêm s vào tập fdread
   FD_SET(s, &fdread);
   ret = select(0, &fdread, NULL, NULL, NULL); // Đợi sự kiện trên socket
   if (ret == SOCKET_ERROR) {
            // Xử lý lỗi
      }
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Select
    - Đoạn chương trình (tiếp)

```
if (ret > 0)
    {
      // Kiểm tra xem s có được thiết lập hay không
      if (FD_ISSET(s, &fdread)) {
      // Đọc dữ liệu từ s
      }
    }
}
```

3.4 Các phương pháp vào ra

Bài tập: Simple Telnet Server

- Sử dụng mô hình select viết hai chương trình server/client thực hiện yêu cầu sau:
  - Viết chương trình server đợi kết nối ở cổng 12345.
  - Với mỗi client kết nối đến, yêu cầu nhập tên và mật khẩu. So sánh tên và mật khẩu trong tệp tin passwd gồm nhiều dòng có dạng:

    user1:pass1

    user2:pass2

    …
  - Nếu đăng nhập thành công thì đợi lệnh từ client và chuyển tiếp lệnh cho hệ thống xử lý, sau đó gửi trả kết quả cho client.

    Ví dụ:

    Hello guest, please authenticate yourself

    Username: noname

    Password: nopass

    Welcome to noname server

    C:\Program files>dir /ah

    <Kết quả>

    **C:\Program files>cd **

    C:

- Viết chương trình SelectTCPServer có thể thực hiện những công việc sau:
  - Đáp ứng được tối đa 255 kết nối.
  - Sử dụng mô hình Select.
  - Định danh các client bằng nickname
  - Chuyển tiếp thông điệp từ một nickname đến tất cả các nickname khác đang đăng nhập vào hệ thống.
  - Cú pháp đăng nhập của một client:
    - “Hello <nickname>\r\n”
  - Cú pháp gửi thông điệp của một nickname đến hệ thống
    - “Chat <nội dung>\r\n”
  - Cú pháp gửi thông điệp của hệ thống đến một client
    - “Chat <nickname> <nội dung>\r\n”

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAAsyncSelect
    - Cơ chế xử lý sự kiện dựa trên thông điệp của Windows
    - Ứng dụng GUI có thể nhận được các thông điệp từ WinSock qua cửa sổ của ứng dụng.
    - Hàm **WSAAsyncSelect** được sử dụng để chuyển socket sang chế độ bất đồng bộ và thiết lập tham số cho việc xử lý sự kiện

```
int WSAAsyncSelect(
   SOCKET s,              //  [IN] Socket sẽ xử lý sự kiện
   HWND hWnd,             //  [IN] Handle cửa sổ nhận sự kiện
   unsigned int wMsg, // [IN] Mã thông điệp, tùy chọn, thường>=WM_USER
   long lEvent            // [IN] Mặt nạ chứa các sự kiện ứng dụng muốn nhận
                          // bao gồm FD_READ,
                          //FD_WRITE,FD_ACCEPT,FD_CONNECT,FD_CLOSE
);
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAAsyncSelect
    - Thí dụ: **WSAAsyncSelect(s, hwnd, WM_SOCKET, FD_CONNECT | FD_READ | FD_WRITE | FD_CLOSE);**
    - Tất cả các cửa sổ đều có hàm callback để nhận sự kiện từ Windows. Khi ứng dụng đã đăng ký socket với cửa sổ nào, thì cửa sổ đó sẽ nhận được các sự kiện của socket.
    - Nguyên mẫu của hàm callback của cửa số:

```
LRESULT CALLBACK WindowProc(
                    HWND hWnd,
                    UINT uMsg,
                    WPARAM wParam,
                    LPARAM lParam );
```

    - Khi cửa sổ nhận được các sự kiện liên quan đến WinSock:
      - uMsg sẽ chứa mã thông điệp mà ứng dụng đã đăng ký bằng WSAAsyncSelect
      - wParam chứa bản thân socket xảy ra sự kiện
      - Nửa cao của lParam chứa mã lỗi nếu có, nửa thấp chứa mã sự kiện có thể là FD_READ, FD_WRITE, FD_CONNECT, FD_ACCEPT, FD_CLOSE

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAAsyncSelect
    - Ứng dụng sẽ dùng hai MACRO: WSAGETSELECTERROR và WSAGETSELECTEVENT để kiểm tra lỗi và sự kiện xảy ra trên socket.
    - Thí dụ:

```
BOOL CALLBACK WinProc(HWND hDlg,UINT wMsg,
   WPARAM wParam, LPARAM lParam)
{
   SOCKET Accept;
    switch(wMsg)
   {
      case WM_PAINT:      // Xử lý sự kiện khác
                  break;
       case WM_SOCKET:  // Sự kiện WinSock
                  if (WSAGETSELECTERROR(lParam)) // Kiểm tra có lỗi hay không
                  {
                              closesocket( (SOCKET) wParam); // Đóng socket
                              break;
                   }
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAAsyncSelect
    - Thí dụ (tiếp):

```
switch(WSAGETSELECTEVENT(lParam)) // Xác định sự kiện
        {
          case FD_ACCEPT:    // Chấp nhận kết nối
               Accept = accept(wParam, NULL, NULL);
               ....
            break;
          case FD_READ:   // Có dữ liệu từ socket wParam
                    ...
            break;
          case FD_WRITE:  // Có thể gửi dữ liệu đến socket wParam
               break;
          case FD_CLOSE:   // Đóng kết nối
               closesocket( (SOCKET)wParam);
            break;
        }
        break;
    }
    return TRUE;
}
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAAsyncSelect
    - Ưu điểm: xử lý hiệu quả nhiều sự kiện trong cùng một luồng.
    - Nhược điểm: ứng dụng phải có ít nhất một cửa sổ, không nên dồn quá nhiều socket vào cùng một cửa sổ vì sẽ dẫn tới đình trệ trong việc xử lý giao diện.

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Xử lý dựa trên cơ chế đồng bộ đối tượng sự kiện của Windows: **WSAEVENT**
    - Mỗi đối tượng có hai trạng thái: Báo hiệu (signaled) và chưa báo hiệu (non-signaled).
    - Hàm **WSACreateEvent** sẽ tạo một đối tượng sự kiện ở trạng thái chưa báo hiệu và có chế độ hoạt động là thiết lập thủ công (manual reset).

      WSAEVENT WSACreateEvent(void);
    - Hàm **WSAResetEvent** sẽ chuyển đối tượng sự kiện về trạng thái chưa báo hiệu

      BOOL WSAResetEvent(WSAEVENT hEvent);
    - Hàm **WSACloseEvent** sẽ giải phóng một đối tượng sự kiện

      BOOL WSACloseEvent(WSAEVENT hEvent);

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Hàm **WSAEventSelect** sẽ tự động chuyển socket sang chế độ non-blocking và gắn các sự kiện của socket với đối tượng sự kiện truyền vào theo tham số

      int WSAEventSelect(

      SOCKET s, // [IN] Socket cần xử lý sự kiện

      WSAEVENT hEventObject,// [IN] Đối tượng sự kiện đã tạo trước đó

      long lNetworkEvents // [IN] Các sự kiện ứng dụng muốn nhận

      // từ WinSock

      );
    - Thí dụ: **rc = WSAEventSelect(s, hEventObject, FD_READ|FD_WRITE);**

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Hàm **WaitForMultipleEvent** sẽ đợi sự kiện trên một mảng các đối tượng sự kiện cho đến khi một trong các đối tượng chuyển sang trạng thái báo hiệu.

```
DWORD WSAWaitForMultipleEvents(
   DWORD cEvents,  // [IN] Số lượng sự kiện cần đợi
   const WSAEVENT FAR * lphEvents,// [IN] Mảng các sự kiện, max 64
   BOOL fWaitAll,                 //[IN] Có đợi tất cả các sự kiện không ?
   DWORD dwTimeout,               //[IN] Thời gian đợi tối đa
   BOOL fAlertable   //[IN]  Thiết lập là FALSE
);
```

Giá trị trả về

- Thành công: Số thứ tự của sự kiện xảy ra + WSA_WAIT_EVENT_0.
- Hết giờ: WSA_WAIT_TIMEOUT.
- Thất bại: WSA_WAIT_FAILED.

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Xác định mã của sự kiện gắn với một đối tượng sự kiện cụ thể bằng hàm **WSAEnumNetworkEvents.**

```
int WSAEnumNetworkEvents(
   SOCKET s,                          // [IN] Socket muốn thăm dò
   WSAEVENT hEventObject,  // [IN] Đối tượng sự kiện tương ứng
   LPWSANETWORKEVENTS lpNetworkEvents// [OUT] Cấu trúc chứa mã sự kiện
);
```

    - Mã sự kiện lại nằm trong cấu trúc WSANETWORKEVENTS có khai báo như sau

```
typedef struct _WSANETWORKEVENTS
{
        long lNetworkEvents; // Mặt nạ chứa sự kiện được kích ho
        int  iErrorCode[FD_MAX_EVENTS]; // Mảng các mã sự kiện
} WSANETWORKEVENTS, FAR * LPWSANETWORKEVENTS;
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Thí dụ

```
#include <winsock2.h>
#define MAX_EVENTS 64
int _tmain(int argc, _TCHAR* argv[])
{
SOCKET SocketArray [MAX_EVENTS];
WSAEVENT EventArray [MAX_EVENTS],NewEvent;
SOCKADDR_IN InternetAddr;
SOCKET Accept, Listen;
DWORD EventTotal = 0;
DWORD Index, i;
WSADATA    wsaData;
WORD       wVersion = MAKEWORD(2,2);
int                rc = WSAStartup(wVersion,&wsaData);
// Thiết lập TCP socket đợi kết nối ở 8888
Listen = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
InternetAddr.sin_family = AF_INET;
InternetAddr.sin_addr.s_addr = htonl(INADDR_ANY);
InternetAddr.sin_port = htons(8888);
rc = bind(Listen, (PSOCKADDR) &InternetAddr,sizeof(InternetAddr));
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Thí dụ (tiếp)

```
SOCKET Accept, Listen;
…
NewEvent = WSACreateEvent();
 WSAEventSelect(Listen, NewEvent,FD_ACCEPT | FD_CLOSE);
rc = listen(Listen, 5);
WSANETWORKEVENTS   NetworkEvents;
SocketArray[EventTotal] = Listen;
EventArray[EventTotal] = NewEvent;
EventTotal++;
char  buffer[1024];
int               len;
while(TRUE)
{
   // Đợi tất cả các sự kiện
   Index = WSAWaitForMultipleEvents(EventTotal,EventArray, FALSE,
                                    WSA_INFINITE, FALSE);
   Index = Index - WSA_WAIT_EVENT_0;
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Thí dụ (tiếp)

```
// Duyệt để tìm ra sự kiện nào được báo hiệu
for(i=Index; i < EventTotal ;i++)
{
 Index = WSAWaitForMultipleEvents(1, &EventArray[i], TRUE, 1000,
                                  FALSE);
 if ((Index == WSA_WAIT_FAILED) || (Index == WSA_WAIT_TIMEOUT))
    continue;
 else
 {
    Index = i;
   WSAResetEvent(EventArray[Index]);
    WSAEnumNetworkEvents(
       SocketArray[Index],
       EventArray[Index],
      &NetworkEvents);
```

3.4 Các phương pháp vào ra

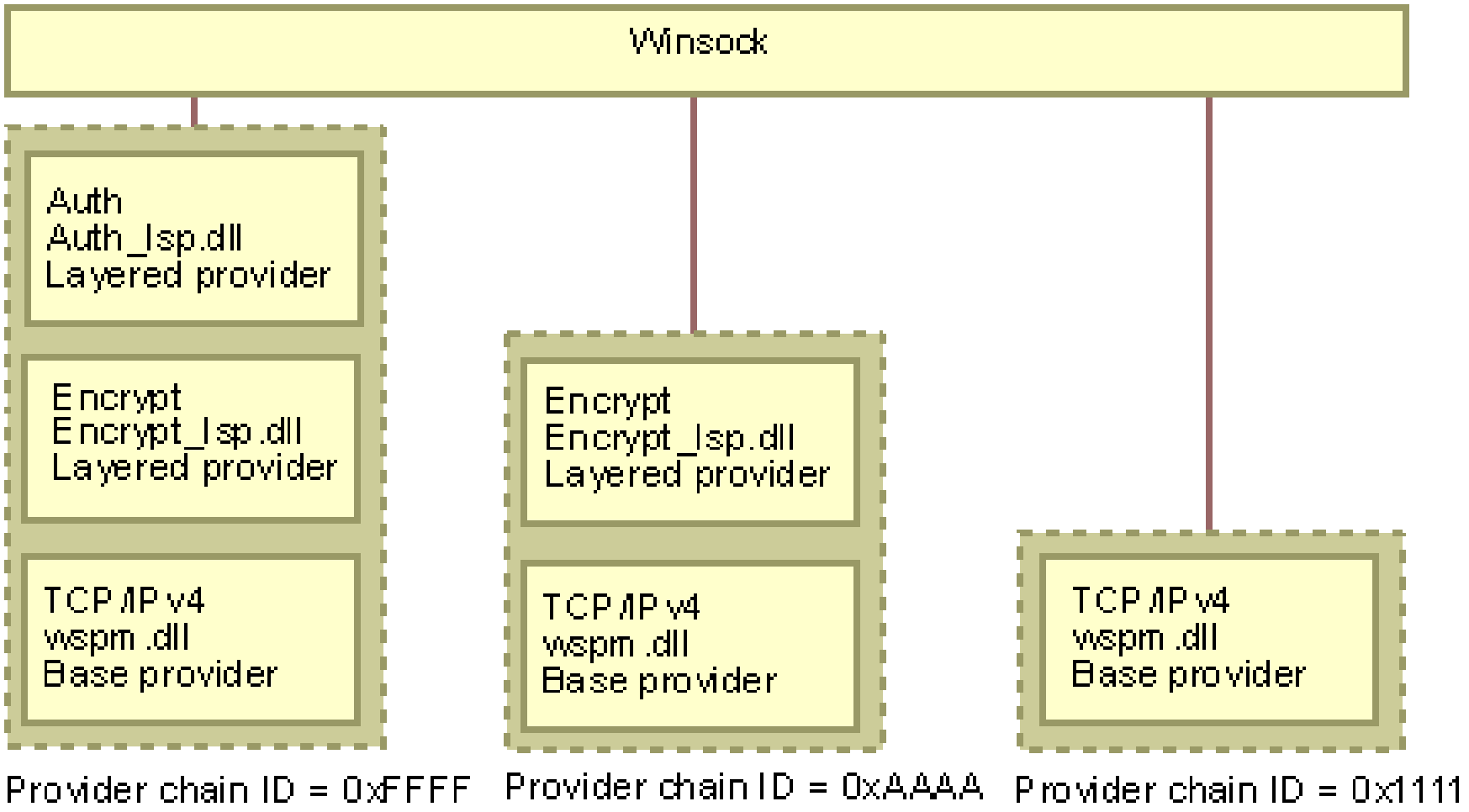
- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Thí dụ (tiếp)

```
// Kiểm tra sự kiện FD_ACCEPT
 if (NetworkEvents.lNetworkEvents & FD_ACCEPT)
 {
    if (NetworkEvents.iErrorCode[FD_ACCEPT_BIT] != 0)
    {
       printf("FD_ACCEPT failed with error %d\n",
          NetworkEvents.iErrorCode[FD_ACCEPT_BIT]);
       break;
    }

    // Chấp nhận kết nối mới
    // cho vào danh sách socket và sự kiện
    Accept = accept(
       SocketArray[Index],
       NULL, NULL);
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình WSAEventSelect
    - Thí dụ (tiếp)

```
if (EventTotal > WSA_MAXIMUM_WAIT_EVENTS)
        {
          printf("Too many connections");
          closesocket(Accept);
          break;
        }

        NewEvent = WSACreateEvent();

        WSAEventSelect(Accept, NewEvent,
          FD_READ | FD_WRITE | FD_CLOSE);

        EventArray[EventTotal] = NewEvent;
        SocketArray[EventTotal] = Accept;
        EventTotal++;
        printf("Socket %d connected\n", Accept);
      }
...
```


Winsock
Auth
Auth_lsp.dll
Layered provider
Encrypt
Encrypt_lsp.dll
Layered provider
TCP/IP v4
wspm .dll
Base provider
Encrypt
Encrypt_lsp.dll
Layered provider
TCP/IP v4
wspm .dll
Base provider
TCP/IP v4
wspm .dll
Base provider
Provider chain ID = 0xFFFF
Provider chain ID = 0xAAAA
Provider chain ID = 0x1111

Bài tập

- Viết chương trình chat+gửi file đơn giản (client +server) sử dụng mô hình WSAEventSelect. Có thể nhập và hiển thị tiếng Việt. Có quản lý username, password.

Nội dung lưu trong xâu có kiểu wchar_t. Số lượng byte gửi đi = chiều dài xâu * 2.

Bài tập

- Thiết kế và cài đặt một giao thức cho phép gửi file và văn bản đồng thời trên **một** kết nối TCP.
- Thiết kế và cài đặt một giao thức cho phép gửi file và văn bản đồng thời trên **một** socket UDP.

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped
    - Sử dụng cấu trúc OVERLAPPED chứa thông tin về thao tác vào ra.
    - Các thao tác vào ra sẽ trở về ngay lập tức và thông báo lại cho ứng dụng theo một trong hai cách sau:
      - **Event** được chỉ ra trong cấu trúc OVERLAPPED.
      - **Completion routine** được chỉ ra trong tham số của lời gọi vào ra.
    - Các hàm vào ra sử dụng mô hình này:
      - WSASend
      - WSASendTo
      - WSARecv
      - WSARecvFrom
      - WSAIoctl
      - WSARecvMsg
      - AcceptEx
      - ConnectEx
      - TransmitFile
      - TransmitPackets
      - DisconnectEx
      - WSANSPIoctl

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped– Xử lý qua **event**
    - Cấu trúc OVERLAPPED

```
typedef struct WSAOVERLAPPED
{
    DWORD    Internal;
    DWORD    InternalHigh;
    DWORD    Offset;
    DWORD    OffsetHigh;
    WSAEVENT hEvent;
} WSAOVERLAPPED, FAR * LPWSAOVERLAPPED
```

Internal, InternalHigh,Offset,OffsetHigh được sử dụng nội bộ trong WinSock

hEvent là đối tượng **event** sẽ được báo hiệu khi thao tác vào ra hoàn tất, chương trình cần khởi tạo cấu trúc với một đối tượng sự kiện hợp lệ.

Khi thao tác vào ra hoàn tất, chương trình cần lấy kết quả vào ra thông qua hàm **WSAGetOverlappedResult**

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped– Xử lý qua **event**
    - Hàm **WSAGetOverlappedResult**

```
BOOL WSAGetOverlappedResult(
  SOCKET s,
  LPWSAOVERLAPPED lpOverlapped,
  LPDWORD lpcbTransfer,
  BOOL fWait,
  LPDWORD lpdwFlags
);
```

s là socket muốn kiểm tra kết quả

lpOverlapped là con trỏ đến cấu trúc OVERLAPPED

lpcbTransfer là con trỏ đến biến sẽ lưu số byte trao đổi được

fWait là biến báo cho hàm đợi cho đến khi thao tác vào ra hoàn tất

lpdwFlags : cờ kết quả của thao tác

Hàm trả về TRUE nếu thao tác hoàn tất hoặc FALSE nếu thao tác chưa hoàn tất, có lỗi hoặc không thể xác định.

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped – Xử lý qua **event**
    - Tạo đối tượng event với **WSACreateEvent**.
    - Khởi tạo cấu trúc OVERLAPPED với event vừa tạo.
    - Gửi yêu cầu vào ra với tham số là cấu trúc OVERLAPPED vừa tạo, **tham số liên quan đến CompletionRoutine phải luôn bằng NULL**.
    - Đợi thao tác kết thúc qua hàm **WSAWaitForMultipleEvents.**
    - Nhận kết quả vào ra qua hàm WSAGetOverlappedResult

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped – Thí dụ xử lý qua **event**

```
// Khởi tạo WinSock và kết nối đến 127.0.0.1:8888
…
OVERLAPPED              overlapped; // Khai báo cấu trúc OVERLAPPED
WSAEVENT  receiveEvent = WSACreateEvent(); // Tạo event
memset(&overlapped,0,sizeof(overlapped));
overlapped.hEvent = receiveEvent;

char          buff[1024];  // Bộ đệm nhận dữ liệu
WSABUF        databuff;                 // Cấu trúc mô tả bộ đệm
databuff.buf = buff;
databuff.len = 1024;
DWORD         bytesReceived = 0;        // Số byte nhận được
DWORD         flags = 0;                / Cờ quy định cách nhận, bắt buộc phải có

while (1)
{
              DWORD       flags = 0;
              // Gửi yêu cầu nhận dữ liệu
              rc = WSARecv(s,&databuff,1,&bytesReceived,&flags,&overlapped,0);
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped – Thí dụ xử lý qua **event**

```
if (rc == SOCKET_ERROR)
{
        rc = WSAGetLastError();
        if (rc != WSA_IO_PENDING)
        {
                printf("Loi %d !\n",rc);
                continue;
        }
};
rc = WSAWaitForMultipleEvents(1,&receiveEvent,TRUE,WSA_INFINITE,FALSE);
if ((rc == WSA_WAIT_FAILED)||(rc==WSA_WAIT_TIMEOUT)) continue;
WSAResetEvent(receiveEvent);
rc = WSAGetOverlappedResult(s,&overlapped,&bytesReceived,FALSE,&flags);
// Kiểm tra lỗi
…
// Hiển thị
buff[bytesReceived] = 0;
printf(buff);
}
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped – Xử lý Completion Routine
    - Hệ thống sẽ thông báo cho ứng dụng biết thao tác vào ra kết thúc thông qua một hàm callback gọi là **Completion Routine**
    - Nguyên mẫu của hàm như sau

```
void CALLBACK CompletionROUTINE(
        IN DWORD dwError,   // Mã lỗi
        IN DWORD cbTransferred,         // Số byte trao đổi
        IN LPWSAOVERLAPPED lpOverlapped,  // Cấu trúc lpOverlapped
                                          // tương ứng
        IN DWORD dwFlags );// Cờ kết quả thao tác vào ra
```

    - WinSock sẽ bỏ qua trường **event** trong cấu trúc OVERLAPPED, việc tạo đối tượng event và thăm dò là không cần thiết nữa.

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped – Xử lý Completion Routine
    - Ứng dụng cần chuyển luồng sang trạng thái **alertable** ngay sau khi gửi yêu cầu vào ra.
    - Các hàm có thể chuyển luồng sang trạng thái **alertable**: **WSAWaitForMultipleEvents, SleepEx**
    - Nếu ứng dụng không có đối tượng event nào thì có thể sử dụng SleepEx

```
DWORD SleepEx(DWORD     dwMilliseconds, // Thời gian đợi
              BOOL      bAlertable      //  Trạng thái alertable
              );
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped – Thí dụ Completion Routine

```
// Khai báo các cấu trúc cần thiết
SOCKET      s;
OVERLAPPED              overlapped;
char        buff[1024];
WSABUF      databuff;
DWORD       flags;
DWORD       bytesReceived = 0;
Int         rc = 0;

void CALLBACK CompletionRoutine(  IN DWORD dwError,
                                  IN DWORD cbTransferred,
                                  IN LPWSAOVERLAPPED lpOverlapped,
                                  IN DWORD dwFlags)
{
            if (dwError != 0||cbTransferred==0) // Xử lý lỗi
            {
                        closesocket(s);
                        return;
            };
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped – Thí dụ Completion Routine

```
// Hiển thị xâu ra màn hình
buff[cbTransferred]=0;
printf(buff);
// Khởi tạo lại cấu trúc overlapped và lại gửi tiếp yêu cầu nhận dữ liệu
memset(&overlapped,0,sizeof(overlapped));
flags = 0;
rc = WSARecv(s, &databuff, 1, &bytesReceived, &flags, &overlapped,
                                                    CompletionRoutine);
if (rc == SOCKET_ERROR)
{
        rc = WSAGetLastError();
        if (rc != WSA_IO_PENDING)
                printf("Loi %d !\n",rc);
};
return;
}
```

3.4 Các phương pháp vào ra

- Các mô hình vào ra của WinSock
  - Mô hình Overlapped – Thí dụ Completion Routine

```
int _tmain(int argc, _TCHAR* argv[])
{
        // Khởi tạo và kết nối đến 127.0.0.1:8888
        …
        // Khởi tạo cấu trúc overlapped
        memset(&overlapped,0,sizeof(overlapped));
        // Khởi tạo bộ đệm dữ liệu
        databuff.buf = buff;
        databuff.len = 1024;
        // Gửi yêu cầu vào ra
        rc = WSARecv(s, &databuff,1,&bytesReceived,&flags,&overlapped,
                                                        CompletionRoutine);
        // Xử lý lỗi…
        // Chuyển luồng sang trạng thái  alertable
        while (1)      SleepEx(1000,TRUE);
        getch();
        closesocket(s);
        WSACleanup();
        return 0;
}
```

3.4 Các phương pháp vào ra

- Bài tập
  - Viết chương trình Server sử dụng mô hình Overlapped – Completion Routine có thể nhận nhiều kết nối từ client và chuyển tiếp dữ liệu từ một client đến các client còn lại.