LẬP TRÌNH ỨNG DỤNG BẰNG WINDOWS SOCKET

Mục tiêu:

Sau bài thực hành này, sinh viên có thể:

- Sử dụng được các hàm cơ bản trong Socket
- Lập trình một ứng dụng mạng đơn giản bằng socket

Nội dung chính

- Giới thiệu socket
- Hướng dẫn viết một ứng dụng mạng Server-Client bằng socket
- Giới thiệu một số hàm cơ bản của lớp CSocket.
- Minh họa ứng dụng chat đơn giản giữa client và server

1 Socket

Sockets cung cấp một interface để lập trình mạng tại tầng Transport. Một socket là một end-point của một liên kết giữa hai ứng dụng. Ngày nay, Socket được hỗ trợ trong hầu hết các hệ điều hành như MS Windows (WinSock), Linux và được sử dụng trong nhiều ngôn ngữ lập trình khác nhau: như C, C++, Java, Visual Basic, C#, . . .

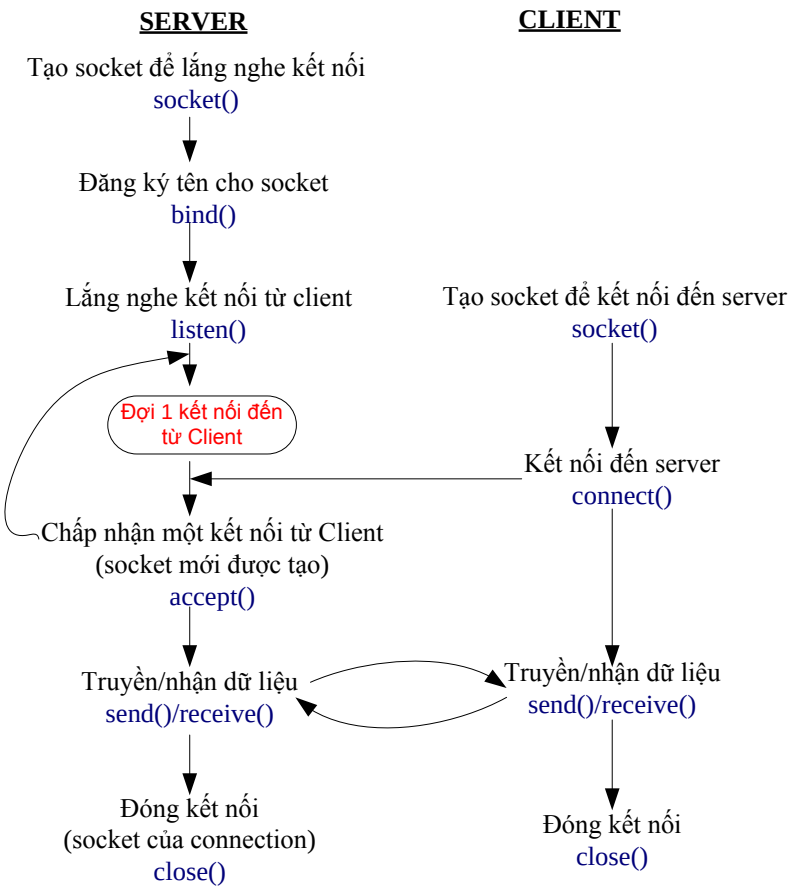
Windows Socket Application Programming Interface (Winsock API) là một thư viện các hàm socket. Winsock hỗ trợ các lập trình viên xây dựng các ứng dụng mạng trên nền TCP/IP.

2 Xây dựng ứng dụng Client-Server với Socket

Khi xây dựng một ứng dụng mạng, chúng ta thực hiện các bước sau:

1. Xác định kiến trúc mạng: Client – Server, Peer-to-Peer
2. Giao thức sử dụng tầng Transport: TCP, UDP
3. Các port sử dụng ở Server và Client
4. Giao thức tầng ứng dụng khi trao đổi dữ liệu giữa hai end-host
5. Lập trình

Phần này trình bày các bước cơ bản trong việc xây dựng các ứng dụng mạng theo kiến trúc Client-Server và giao thức sử dụng ở tầng Transport là TCP bằng Socket.



Hình 1: Sơ đồ tương tác giữa Server-Client theo giao thức TCP

Trong giai đoạn truyền nhận dữ liệu, việc trao đổi dữ liệu giữa Client và Server phải tuân thủ theo giao thức của ứng dụng.

Ghi chú:

- nếu chúng ta phát triển ứng dụng theo các giao thức đã định nghĩa sẵn, chúng ta phải tham khảo và tuân thủ đúng những qui định của giao thức (tham khảo trong các tài liệu RFC (Request For Comments)).
- Nếu xây dựng ứng dụng dạng Peer-to-Peer, thì một ứng dụng phải có cả chức năng client và server trong mô hình trên.

3 Một số hàm cơ bản trong lớp CSocket

CSocket (một lớp được hỗ trợ trong MFC) là một lớp kế thừa từ lớp CAsynSocket dùng

để quản lý việc truyền và nhận dữ liệu thông qua socket. Trong phần này, chúng ta sẽ khảo sát một số hàm cơ bản trong lớp CSocket. Các hàm khác các bạn tham khảo trong MSDN: http://msdn.microsoft.com/en-US/library/65bbyctt%28v=VS.80%29.aspx

Khởi tạo Socket

```
BOOL AfxSocketInit(
   WSADATA* lpwsaData = NULL
);
```

Trước khi sử dụng các hàm của lớp CSocket, chúng ta phải gọi hàm này để khởi tạo Windows Socket với tham số lpwsaData gán bằng NULL. Nếu lpwsaData không bằng NULL thì địa chỉ của tham số này phải lấy từ hàm WSAStartup.

Tạo socket

```
BOOL Create(
   UINT nSocketPort = 0,
   int nSocketType = SOCK_STREAM,
   LPCTSTR lpszSocketAddress = NULL
);
```

- nSocketPort: port của socket; nếu bằng 0 thì port sẽ được MFC chọn ngẫu nhiên.
- nSocketType là SOCK_STREAM (TCP) hay SOCK_DGRAM (UDP).
- lpszSocketAddress: địa chỉ IP của host dùng socket. Nếu dùng NULL, socket sẽ lắng nghe hoạt động của client trên tất cả các card mạng.

Giá trị trả về: khác 0 nếu thành công; ngược lại thì bằng 0 và mã lỗi cụ thể sẽ được cho khi gọi hàm GetLastError.

Ghi chú:

- giải thích của các thành phần sẽ được sử dụng trong các hàm sau
- hàm GetLastError dùng để lấy mã lỗi cụ thể khi xảy ra lỗi trong quá trình thực hiện các hàm. Do đó, các hàm về sau, nếu có lỗi thì dùng hàm này để lấy mã lỗi

chi tiết.

Đăng ký tên cục bộ cho socket

```
BOOL Bind(
   UINT nSocketPort,
   LPCTSTR lpszSocketAddress = NULL
);
BOOL Bind (
   const SOCKADDR* lpSockAddr,
   int nSockAddrLen
);
```

- lpSockAddr: trỏ đến cấu trúc SOCKADDR chứa địa chỉ IP của host dùng socket.

- nSockAddrLen: chiều dài của địa chỉ lpSockAddr được tính theo byte.

Giá trị trả về: khác 0 nếu thành công; ngược lại bằng 0.

Giải phóng/Đóng socket

```
Close()
```

Kết nối đến server

```
BOOL Connect(
   LPCTSTR lpszHostAddress,
   UINT nHostPort
);
BOOL Connect(
   const SOCKADDR* lpSockAddr,
   int nSockAddrLen );
```

- lpszHostAddress: địa chỉ IP của Server.

- nHostPort: Port của socket lắng nghe kết nối trên Server.

- lpSockAddr: trỏ đến cấu trúc SOCKADDR chứa địa chỉ IP của Server.

- nSockAddrLen: chiều dài của địa chỉ trong lpSockAddr được tính theo byte.

Giá trị trả về: khác 0 nếu thành công; ngược lại, thất bại = 0.

Lắng nghe các yêu cầu kết nối

```
BOOL Listen(
   int nConnectionBacklog = 5
);
```

- Hàm này chỉ hỗ trợ cho socket dạng SOCK_STREAM.
- nConnectionBacklog: chiều dài tối đa mà hàng đợi kết nối chưa được chấp nhận có thể tăng. Miền giá trị từ 1 đến 5.

Giá trị trả về: khác 0 = thành công, bằng 0 = thất bại.

Chấp nhận một kết nối

```
virtual BOOL Accept(
   CAsyncSocket& rConnectedSocket,
   SOCKADDR* lpSockAddr = NULL,
   int* lpSockAddrLen = NULL );
```

- rConnectedSocket: tham chiếu định danh socket của kết nối được chấp nhận.
- lpSockAddr trỏ đến cấu trúc SOCKADDR nhận địa chỉ IP của socket kết nối đến.
- Nếu lpSockAddr hay lpSockAddrLen lấy giá trị mặc định NULL thì sẽ không có thông tin từ socket (trên client) được chấp nhận được trả về.
- lpSockAddrlen chứa chiều dài thực sự của lpSockAddr khi trả về theo byte.

Giá trị trả về: khác 0 nếu thành công và bằng 0 nếu thất bại.

Nhận dữ liệu

```
virtual int Receive(
   void* lpBuf,
   int nBufLen,
   int nFlags = 0
);
```

- lpBuf: vùng đệm chứa dữ liệu.
- nBuffLen: kích thước của vùng đệm tính theo byte.
- nFlag: cách nhận dữ liệu, sử dụng giá trị mặc định là 0.
- Giá trị trả về là số byte nhận được, nếu socket đóng thì giá trị trả về là 0, ngoài ra giá trị sẽ trả về là SOCKET_ERROR.

Gửi dữ liệu

```
virtual int Send(
   const void* lpBuf,
   int nBufLen,
   int nFlags = 0
);
```

- lpBuf: vùng đệm chứa dữ liệu để truyền đi.
- nBufLen: chiều dài vùng đệm.
- nFlag: cách truyền dữ liệu, sử dụng giá trị mặc định là 0.
- Giá trị trả về là số ký tự được gửi, nếu thất bại giá trị trả về là SOCKET_ERROR.

4 Minh họa ứng dụng chat đơn giản

Bài toán: viết một ứng dụng chat tuần tự giữa Server – Client (theo thứ tự: server → client → server → client →…) bằng Console. Quá trình chat sẽ kết thúc khi một trong hai bên gõ Exit

1. Xác định kiến trúc mạng: **Client – Server**
2. Giao thức sử dụng tầng Transport: **TCP** (dữ liệu truyền giữa client và server là text)
3. Các port sử dụng ở Server và Client: Server - 1234
4. Giao thức tầng ứng dụng khi trao đổi dữ liệu giữa hai end-host:
   - server → client → server → client →…

- format thông điệp truyền giữa client và server: <Chiều dài thông điệp><Thông Điệp>

5. Lập trình: xem chi tiết mục 4.1 và 4.2

4.1 Server

Khai báo biến:

```
CSocket server, client;
```

Khởi tạo Windows Socket

```
AfxSocketInit(NULL);
```

Tạo socket lắng nghe kết nối

```
if (!server.Create(1234))
     {
          printf("Tao socket khong thanh cong");
          exit();
     }
server.Listen();
```

Chấp nhận kết nối của Client

```
server.Accept(client)
```

Trao đổi thông điệp với Client

```
do
{
     printf("\nServer: ");
     gets(s_str);
     len = strlen(s_str);
     client.Send(s_str,len,0);
     len = client.Receive(r_str,100,0);
     // gan ket thuc chuoi
     r_str[len] = 0;
```

```
            // hien thi chuoi nhan duoc ra man hinh
            printf("\nCleint: %s",r_str);
      }while(strcmp(r_str,"exit")&&strcmp(s_str,"exit"));
```

Đóng kết nối và socket của Server

```
      client.Close();
      server.Close();
```

4.2 Client

Khai báo biến:

```
      CSocket client;
```

Khởi tạo Windows Socket

```
      AfxSocketInit(NULL);
```

Tạo socket để kết nối đến server

```
      client.Create();
```

Connect đến Server

```
      client.Connect(svrAddr,1234)
```

Trao đổi thông điệp với Server

```
      do
      {
            len = client.Receive(r_str,100,0);
            r_str[len] = 0;
            printf("\n Server: %s",r_str);
            printf("\n Client: ");
            gets(s_str);
            client.Send(s_str,strlen(s_str),0);
      }while(strcmp(r_str,"exit")&&strcmp(s_str,"exit"));
```

Đóng kết nối

```
      client.Close();
```