Cơ chế lập trình non-blocking sử dụng WSAAsyncSelect

Như đã đề cập ở các bài trước, nhiều hàm trong lập trình socket hoạt động theo cơ chế blocking ví dụ như hàm accept, hàm recv. Khi server gọi hàm accept(), chương trình server sẽ treo cho đến khi có một client gọi hàm connect() để kết nối đến.

Ở thực tế, chúng ta ít khi nào chấp nhận để chương trình bị treo do đó phải sử dụng một số cơ chế lập trình non-blocking. Trong nội dung bài này, chúng ta sẽ nghiên cứu cơ chế xử lý non-blocking sử dụng WSAAsyncSelect

WSAAsyncSelect sử dụng cơ chế xử lý thông điệp của Windows. Khi windows kiểm tra và nhận thấy có một kết nối đến server sẽ gửi một thông điệp để báo cho server biết, lúc này server mới gọi hàm accept và không phải chờ đợi dẫn đến bị treo chương trình. Tương tự, khi có dữ liệu gửi đến, windows sẽ báo cho ứng dụng. Lúc này ứng dụng mới gọi hàm recv và nhận ngay dữ liệu.

Cú pháp hàm WSAAsyncSelect:

```
int WSAAsyncSelect(
  __in      SOCKET ,
  __in      HWND ,
  __in      unsigned int wMsg ,
  __in      long lEvent
);
```

Giải thích ý nghĩa:

s

socket cần Windows theo dõi giúp

hWnd

handle tới một cửa sổ, windows sẽ gửi các thông điệp báo hiệu đến cửa sổ này.

wMsg

Loại thông điệp mà Windows sẽ gửi khi có sự kiện xảy ra tại socket. Thông điệp này thường sẽ là thông điệp tự định nghĩa, có giá trị lớn hơn WM_USER

lEvent

Cho biết những sự kiện mà windows sẽ giám sát.

Giá trị lEvent thường là kết hợp của các giá trị FD_READ, FD_WRITE, FD_CLOSE, FD_ACCEPT.

Ví dụ lEvent có thể là FD_READ | FD_CLOSE | FD_ACCEPT

Các sự kiện chính:

Sự kiện	***Ý nghĩa***
FD_READ	Báo hiệu có dữ liệu gửi đến cho

Sự kiện	***Ý nghĩa***
	socket, sẵn sàng để gọi hàm recv
FD_ACCEPT	Báo hiệu cho socket server (đã gọi hàm listen) có một kết nối từ phía client
FD_CLOSE	Báo hiệu đầu còn lại của kết nối đã đóng socket

Một số đặc điểm cần lưu ý:

- Windows ngay lập tức đặt socket s vào trạng thái non-blocking, bất chấp giá trị của lEvent
- Các socket cha nếu đã được gọi hàm WSAAsyncSelect trước đó, socket con sinh ra bởi hàm accept sẽ có cùng đặc điểm với socket cha.
- Lời gọi hàm WSAAsyncSelect sau sẽ ghi đè lên lời gọi hàm trước, không có ý nghĩa kết hợp

Trong bất kỳ thông điệp (message) nào của Windows đều có 2 thông số quan trọng là wParam và lParam. Với sự kiện sinh ra bởi WSAAsyncSelect thì:
wParam: là socket có sự kiện phát sinh
lParam: hai byte thấp (low word) cho biết sự kiện phát sinh (FD_READ, FD_ACCEPT...), hai byte cao (high word) cho biết lỗi phát sinh nếu có.
Để thao tác dễ dang trên lParam, windows đã định nghĩa sẵn 2 macro:

```
#define WSAGETSELECTERROR(lParam)        HIWORD(lParam)
#define WSAGETSELECTEVENT(lParam)        LOWORD(lParam)
```

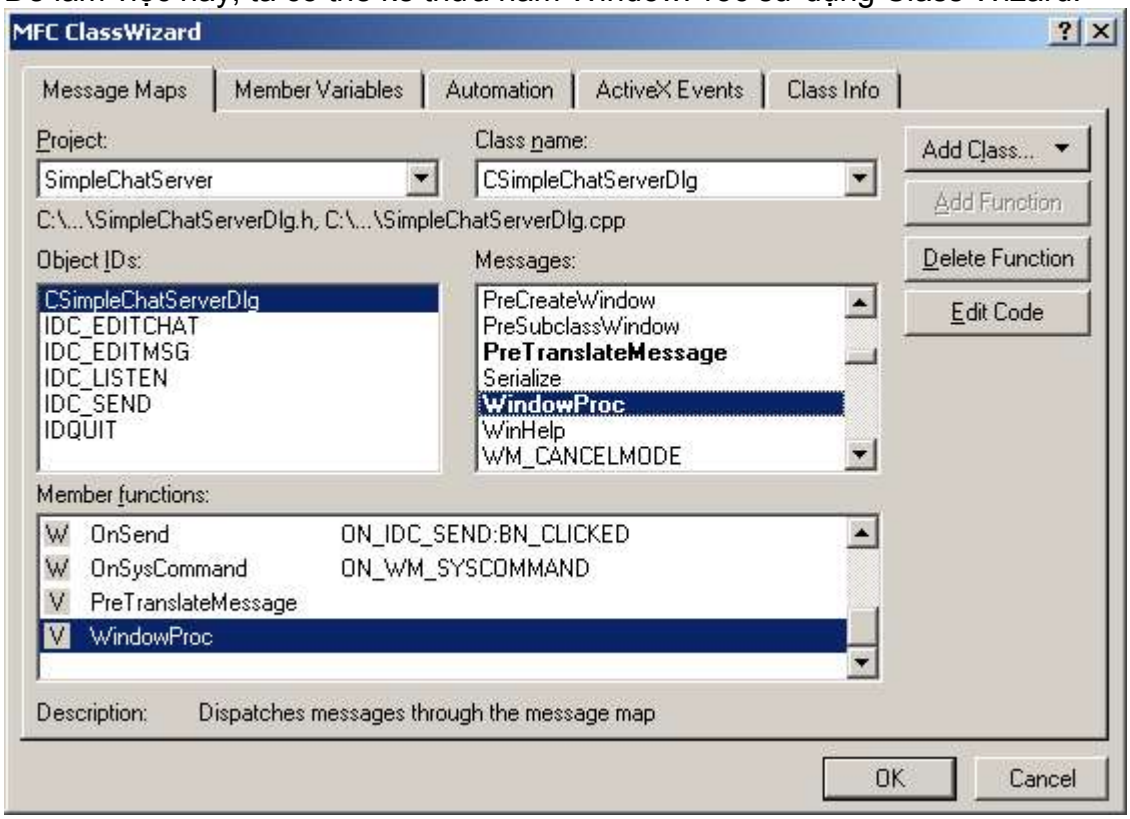
Nguyên tắc khai báo và xử lý sự kiện
Giả sử ta đặt tên sự kiện cần bắt là WM_SOCKET, đầu tiên ta sẽ định nghĩa
#define WM_SOCKET WM_USER+1
gọi hàm WSAAsyncSelect:
WSAAsyncSelect(socket,m_hWnd,WM_SOCKET,FD_ACCEPT|FD_READ|FD_CLOSE);
Có 2 cách để "bắt" và xử lý sự kiện này.
Cách 1:
Trước hết xây dựng hàm để xử lý sự kiện:

```
void CChatDlg::SockMsg(WPARAM wParam, LPARAM lParam)
{
        if (WSAGETSELECTERROR(lParam))
        {
                // Có lỗi xảy ra, xuất tb lỗi và ngắt socket
               closesocket(wParam);
        }
        switch(WSAGETSELECTEVENT(lParam))
        {
                case FD_ACCEPT:
                        //Có 1 kết nối đến, gọi hàm accept để tiếp nhận
                        sockClient = accept(wParam,NULL,NULL);
```

```
                break;
        case FD_READ:
                //Có dữ liệu gửi đến, gọi hàm recv
                break;
        case FD_CLOSE:
                //Đầu bên kia đã đóng socket
                closesocket(wParam);
                break;
    }
}
```

Vấn đề còn lại, phải báo hiệu cho windows biết khi có sự kiện WM_SOCKET sẽ gọi giúp hàm SockMsg.

Để làm việc này, ta có thể kế thừa hàm WindowProc sử dụng Class Wizard:



Hàm WindowProc sẽ như sau:

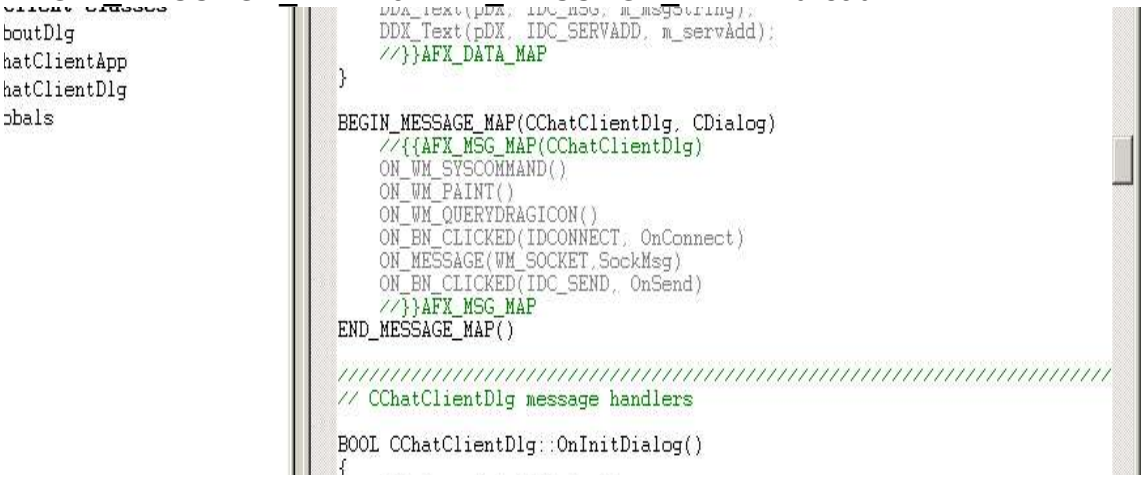
```
LRESULT CChatDlg::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
    // TODO: Add your specialized code here and/or call the base class
    if (message==WM_SOCKET){
        SockMsg(wParam,lParam);
    }
    return CDialog::WindowProc(message, wParam, lParam);
}
```

Như vậy, khi có sự kiện VM_SOCKET phát sinh, hàm SockMsg sẽ được gọi.

Cách 2:
Xây dựng hàm SockMsg như trên, nhưng kiểu trả về của hàm là LRESULT như sau:

```
LRESULT CChatClientDlg::SockMsg(WPARAM wParam, LPARAM lParam)
{
	if (WSAGETSELECTERROR(lParam))
	{
		// Display the error and close the socket
		closesocket(wParam);
	}
	switch(WSAGETSELECTEVENT(lParam))
	{
		case FD_READ:
			break;
		case FD_CLOSE:
			break;
		.....................
	}
	return 0;
}
```

Sau đó khai báo macro **ON_MESSAGE** nằm giữa đoạn BEGIN_MESSAGE_MAP và END_MESSAGE_MAP như sau:



Những phần trình bày ở trên là những nét chính của lập trình sử dụng WSAAsyncSelect. Đề nghị các bạn nên đọc thêm trong sách Network Programming và MSDN cũng như xem thêm ví dụ.

HẾT