ÁP DỤNG MẪU THIẾT KẾ HƯỚNG ĐỐI TƯỢNG TRONG XÂY DỰNG CÁC ỨNG DỤNG MẠNG THEO GIAO THỨC TCP/IP

Trần Đan Thư, Huỳnh Thụy Bảo Trân
Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM
(Bài nhận ngày 05 tháng 04 năm 2006, hoàn chỉnh sửa chữa ngày 20 tháng 07 năm 2006)

TÓM TẮT: *Giao thức TCP/IP đóng vai trò rất quan trọng trong việc phát triển các ứng dụng mạng. Các ứng dụng TCP/IP thường được viết bằng C/C++ và gọi các hàm thư viện được hỗ trợ sẵn. Tuy nhiên, do có nhiều phiên bản thư viện hàm trên Unix và Windows nên người lập trình gặp khó khăn trong việc tái sử dụng mã nguồn đã được viết. Mặt khác, bản thân giao tiếp lập trình của các hàm thư viện này làm phát sinh những đoạn mã nguồn rườm rà, trùng lặp và không rõ ràng. Trong bài báo này, chúng tôi áp dụng các mẫu thiết kế hướng đối tượng để phát triển hệ thống lớp hỗ trợ lập trình giao thức TCP/IP. Hệ thống lớp này giải quyết vấn đề tương thích trên nhiều môi trường, thuận lợi cho việc tái sử dụng mã nguồn, đồng thời làm rõ ràng ngữ nghĩa của các đối tượng trong các ứng dụng truyền thông theo giao thức TCP/IP.*

Từ Khoá: *Mẫu thiết kế, Giao thức TCP/IP, Lập trình hướng đối tượng*

1. GIỚI THIỆU

Các mẫu thiết kế hướng đối tượng [1,3,4,8] được sử dụng rộng rãi và có hiệu quả trong việc thiết kế các phần mềm trong nhiều lĩnh vực khác nhau. Những mẫu thiết kế này có thể áp dụng giải quyết nhiều vấn đề đa dạng trong quá trình phát triển phần mềm hướng đối tượng. Giao thức TCP/IP [2,5] là một trong những giao thức mạng quan trọng được sử dụng thông dụng nhất hiện nay để phát triển các phần mềm mạng và truyền thông. Tuy nhiên, khi viết các chương trình C/C++ với giao thức TCP/IP, mặc dù cùng một yêu cầu chức năng nhưng người lập trình thường phải viết những chương trình nguồn riêng cho từng môi trường như *Unix* và *Windows*. Mặt khác, mã nguồn C/C++ có sử dụng các hàm trong thư viện lập trình TCP/IP thường *rườm rà*, *trùng lặp*, và *khó tái sử dụng*. Bài báo này nhằm mục đích trình bày kết quả nghiên cứu của chúng tôi trong việc áp dụng các mẫu thiết kế kinh điển [4] để xây dựng một hệ thống lớp đối tượng giải quyết các vấn đề trong giao tiếp lập trình của các hàm trong thư viện lập trình TCP/IP. Trong phần 2 chúng tôi sẽ tóm tắt lại cơ chế lập trình TCP/IP cùng với những vấn đề nảy sinh. Phần 3 sẽ trình bày việc vận dụng mẫu thiết kế nhằm đề xuất một kiến trúc lớp giải quyết các đề đã được nêu. Việc cài đặt các lớp đối tượng cùng với một số chi tiết kỹ thuật sẽ được trình bày trong phần 4. Phần 5 sẽ điểm qua một số nghiên cứu liên quan và trình bày hướng phát triển.

2. GIAO THỨC LẬP TRÌNH TCP/IP

Phần này trình bày những điểm chính yếu về giao thức TCP/IP, phân tích những vấn đề trở ngại về giao tiếp lập trình mà chúng tôi rút ra qua kinh nghiệm lập trình và giảng dạy [5]. Chúng tôi cũng đề xuất hướng giải quyết nói chung để khắc phục các vấn đề nảy sinh.

2.1 Tổng quan về giao thức TCP/IP

Giao thức TCP/IP là giao thức truyền thông được thiết kế để kết nối các máy tính trên mạng Internet. Thư viện lập trình TCP/IP khởi đầu [2] được cung cấp sẵn trên hệ điều hành họ *Unix* và sau đó được phát triển trên *Windows* (kể từ *Windows 3.1* thì thư viện lập trình TCP/IP được cung cấp miễn phí). Khi *Windows NT* ra đời thì giao thức TCP/IP được tích hợp sẵn trong hệ điều hành đưới dạng các thư viện động DLL [5] và giao tiếp lập trình TCP/IP đã được đưa vào hệ thống hàm lập trình trên *Windows*, cũng như hệ thống lớp *MFC* (*Microsoft Foundation*

Classes [6]) của Microsoft. Hiện nay TCP/IP vẫn đang là giao thức truyền thông quan trọng, việc sử dụng hiệu quả giao thức TCP/IP trong quá trình xây dựng các ứng dụng mạng vẫn là một vấn đề mang tính ứng dụng thực tế, rất đáng được thảo luận [9]. Việc liên lạc giữa hai máy tính theo giao thức TCP được thiết lập dựa trên hai khái niệm: cổng truyền thông (port) và ổ cắm (socket). Đây chỉ là những khái niệm logic ở mức độ quan niệm của người lập trình:

- Cổng truyền thông là một số nguyên: các giao thức chuẩn qui ước dùng các số hiệu cổng từ 0 đến 999 (chẳng hạn giao thức SMTP dùng cổng 25 để chuyển phát thư tín điện tử; giao thức FTP dùng cổng 21 gởi lệnh và cổng 20 truyền dữ liệu), trong khi giao thức không chuẩn có thể dùng các số hiệu cổng từ 1000 đến 64000.
- Khái niệm socket tương tự như các thẻ tập tin (file handle) nhưng được dùng để gởi và nhận dữ liệu thay vì đọc ghi tập tin. Để hai chương trình có thể liên lạc được với nhau, mỗi chương trình trên một máy tính sẽ tạo ra một socket và liên kết socket này với cùng một cổng (thực hiện bằng cách gọi các hàm thư viện TCP/IP).

Chúng ta cũng có thể thiết kế để nhiều chương trình cùng giao tiếp với một chương trình thông qua cùng một cổng giao tiếp. Thông thường các ứng dụng giao tiếp TCP/IP được viết bằng ngôn ngữ lập trình C/C++ và gọi các hàm thư viện về socket được cung cấp sẵn. Các chương trình giao tiếp với nhau theo mô hình khách (client) – chủ (server): chương trình chủ tạo ra thẻ socket và kết buộc với cổng truyền thông và thông tin của máy chủ, chạy sẵn và chờ chương trình khách nối kết vào. Chương trình khách cũng tạo ra một thẻ socket và dùng thẻ socket này để kết nối vào máy chủ (nhờ vào địa chỉ IP của máy chủ – ví dụ như "192.158.68.1" – cùng với cổng truyền thông). Sau khi thiết lập được kết nối, các thẻ socket được dùng để làm "đường truyền" trao đổi dữ liệu giữa các chương trình. Hình 1 tóm tắt các hàm tiêu biểu của thư viện socket C/C++ theo dạng BSD (Berkeley-style, thường dùng trên các máy Unix, Linux...) và của thư viện trên Windows. Chúng ta có thể chú ý các điểm so sánh như sau về mặt lập trình:

- Đối với các thư viện trên Windows thì các thẻ socket được khai báo kiểu **SOCKET**, trong khi đối với BSD thì các thẻ socket có kiểu **int**. Ngoài ra các ứng dụng trên Windows cần phải nạp thư viện động DLL nhờ gọi hàm **WSAStartup()** ;
- Kiểu dữ liệu **sockaddr_in** của BSD có cấu trúc tương thích với kiểu **SOCKADDR_IN** và **SOCKADDR** (kiểu con trỏ **LPSOCKADDR**) trong thư viện của Windows. Đây là kiểu dữ liệu dùng để xác định địa chỉ IP của máy và số hiệu của cổng truyền thông ;
- Các cổng truyền thông (port) là các biến có kiểu **int** khi viết chương trình ;
- Các hàm được chia 3 nhóm chính: các hàm cho chương trình chủ (server), các hàm dùng trong chương trình khách (client), và các hàm chung ;
- Sự khác nhau về tên và thư mục chứa các tập tin khai báo (*.h) giữa hai thư viện.

Dạng BSD (họ Unix) **Tập tin header khai báo hàm (*.h):** < sys/type.h > < netinet/in.h > < sys/socket.h >	- **Kiểu dữ liệu:** **struct** sockaddr_in { **short** sin_family **;** **unsigned short** sin_port **;** **struct** in_addr sin_addr **;** **char** sin_zero[8] **;** } **;** - **Nhóm hàm chung:** **int** socket (**int** af**, int** type**, int** protocol) **;** **// tạo socket** **int** closesocket (**int** hSocket) **;** **// đóng socket** **int** send (**int** hSocket**, char** *buf**, int** len**, int** flags) **;** **…**

	- **Nhóm hàm cho chương trình chủ (server):** **int** bind (**int** hSocket**, struct** sockaddr_in* Addr**, int** AddrLen) **;** // kết buộc thẻ hSocket với địa chỉ IP máy chủ và cổng truyền thông **int** listen (**int** hSocket**, int** backlog) **;** **int** accept (**int** hSvr**, struct** sockaddr_in* Addr**, int*** AddrLen) **;** // chờ khách nối vào**,** hSvr**:** thẻ socket của chủ**,** trả về socket khách **…** - **Nhóm hàm cho chương trình khách (client):** **int** connect (**int** hSvr**, struct** sockaddr_in* Addr**, int** AddrLen) **;** **//** kết nối vào máy chủ**,** hSvr là thẻ socket của máy chủ **…**
Thư viện Windows - **Tập tin header khai báo hàm (*.h):** < winsock.h > - **Hàm nạp và đóng thư viện DLL:** **int** WSStartup(…) **;** **int** WSACleanup( ) **;**	- **Kiểu dữ liệu:** SOCKADDR_IN **và** SOCKADDR - **Nhóm hàm chung:** SOCKET socket (**int** af**, int** type**, int** protocol) **; // tạo socket** **int** closesocket (SOCKET hSocket) **; // đóng socket** **int** send (SOCKET hSocket**, char*** buf**, int** len**, int** flags) **;** **…** - **Nhóm hàm cho chương trình chủ (server):** **int** bind (SOCKET hSocket**,** SOCKADDR* Addr**, int** AddrLen) **;** // kết buộc thẻ hSocket với địa chỉ IP máy chủ và cổng truyền thông **int** listen (SOCKET hSocket**, int** backlog) **;** SOCKET accept (SOCKET hSvr**,** SOCKADDR* Addr**, int*** AddrLen) **;** // chờ khách nối vào**,** hSvr**:** thẻ socket của chủ**,** trả về socket khách **…** - **Nhóm hàm cho chương trình khách (client):** **int** connect (SOCKET hSvr**,** SOCKADDR* Addr**, int** AddrLen) **;** **//** kết nối vào máy chủ**,** hSvr là thẻ socket của máy chủ **…**

Hình 1. Giao tiếp lập trình TCP/IP theo 2 dạng: BSD (Berkeley-style) và Windows

Để viết các chương trình theo giao thức TCP/IP, người lập trình cần tuân theo các bước được qui định (có thể tham khảo chi tiết trong các tài liệu [2, 5, 9]). Trong phạm vi bài viết này, chúng tôi sẽ tập trung vào việc phân tích các vấn đề nảy sinh do điểm không tương đồng giữa các thư viện hàm và sự rườm rà trùng lắp trong mã nguồn của giao tiếp lập trình socket.

2.2 Các vấn đề nảy sinh và hướng giải quyết

Hình 2 minh họa các đoạn mã nguồn được trích ra từ ứng dụng khách viết trên Unix (theo thư viện socket dạng Berkeley) và ứng dụng chủ viết trên Windows (theo thư viện Windows

socket). Chắc chắn rằng chúng ta hoàn toàn có thể có nhu cầu ngược lại: ứng dụng khách trên Windows và ứng dụng chủ trên Unix. Do sự không tương đồng về giao tiếp lập trình: chúng ta phải sao chép mã nguồn và thực hiện một số sửa đổi nhất định. Điều này sẽ đưa đến sự xuất hiện nhiều đoạn mã nguồn cùng ý nghĩa với những sửa đổi vụn vặt phụ thuộc vào môi trường, công việc bảo trì phần mềm có những đoạn mã nguồn trùng lặp như vậy sẽ tốn kém nhiều chi phí và khó bảo đảm sự hoạt động chính xác của phần mềm. Mặt khác chúng ta cũng thấy sự xuất hiện lặp lại khá nhàm chán và lượm thuộm trong mã nguồn như: hằng số **AF_INET**, lời gọi **sizeof( )**, các lệnh đặt giá trị khởi tạo các trường của cấu trúc **SOCKADDR_IN** hay **sockaddr_in**…

Sau quá trình nghiên cứu và phân tích kỹ lưỡng các đoạn mã nguồn C/C++ trích từ các chương trình sử dụng trực tiếp thư viện socket trên Windows và Unix, chúng tôi nhận thấy có những vấn đề như sau cần phải được cải thiện:

- Cùng một chức năng truyền thông nhưng người lập trình phải viết những phiên bản mã nguồn khác nhau trên những môi trường khác nhau. Một số lập trình viên cũng cố gắng giải quyết vấn đề bằng cách sử dụng các chỉ thị biên dịch có điều kiện như **#ifdef**, **#ifndef**, ... và chèn trực tiếp các chỉ thị loại này vào nhiều chỗ trong mã nguồn. Đây là cách giải quyết không trọn vẹn, có thể làm mã nguồn rối rắm và khó bảo trì hơn ;
- Sự trùng lặp, lặp đi lặp lại của các chỉ thị mang tính chất "thủ tục hành chánh" xuất phát từ qui định các tham số trong giao tiếp lập trình của các thư viện hàm socket. Điều này làm phát sinh những đoạn mã nguồn luộm thuộm, khó hiểu, trùng lắp ;
- Có nhiều khó khăn trong việc tái sử dụng mã nguồn xuất phát từ các: sử dụng trực tiếp các hàm thư viện socket của Windows hay Unix, sử dụng các hằng số (đã định nghĩa) hay kiểu dữ liệu đặc thù của từng môi trường, sử dụng trực tiếp các tập tin khai báo hàm (*.h) của riêng từng môi trường ;
- Không trừu tượng hóa được ý nghĩa của các đối tượng truyền thông, điều này xuất phát từ việc trộn lẫn các chi thiết kỹ thuật của giao thức lập trình TCP/IP với mô hình, cơ chế hay thuật toán trao đổi dữ liệu giữa các ứng dụng, các tiến trình truyền thông.

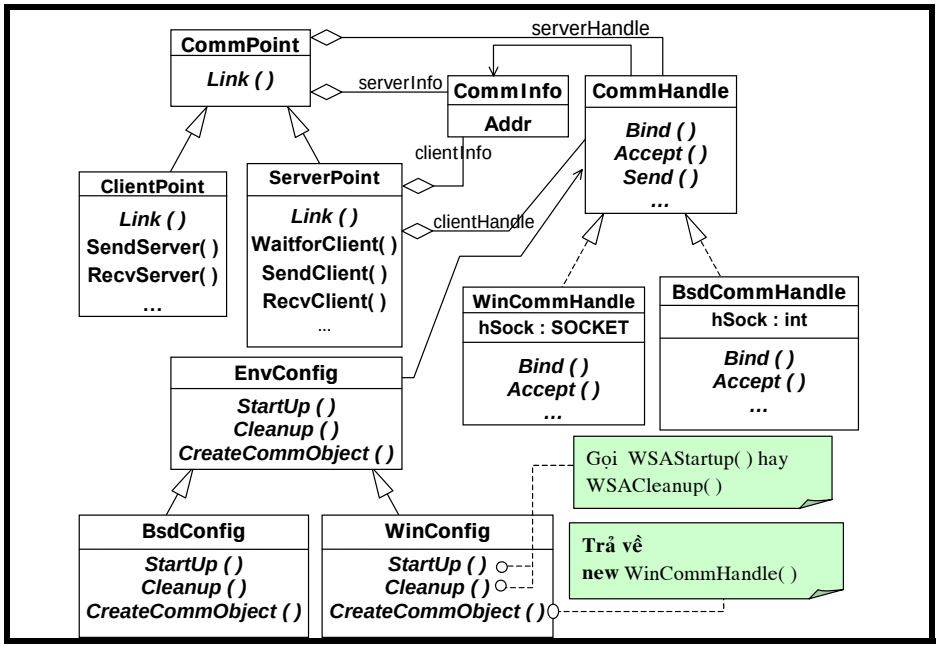
Để giải quyết các vấn đề trên, chúng tôi sẽ sử dụng kỹ thuật hướng đối tượng để thiết kế một kiến trúc lớp nhằm trừu tượng hóa các đối tượng truyền thông, tách bạch rõ ràng giữa chủ và khách, bao bọc và che dấu các chi tiết kỹ thuật TCP/IP, giải quyết vấn đề phụ thuộc cứng vào từng môi trường. Phần kế tiếp của bài báo sẽ trình bày chi tiết về giải pháp đã đề xuất và triển khai.

```
...
        struct sockaddr_in Addr ; unsigned short port=44966; // cổng tự định nghĩa
        char* ipAddr="192.168.58.1" ; int hSock ;
        memset(&Addr, 0, sizeof(Addr) );
        Addr.sin_port = htons(port) ;  Addr.sin_family = AF_INET ;
        Addr.sin_addr.s_addr = inet_addr(ipAddr);
        hSock = socket(AF_INET, SOCK_STREAM, 0) ;
        if (hSock != -1) {
              int status = connect(hSock, &Addr, sizeof(Addr));
              ...}
...
```

Đoạn mã nguồn viết trên Unix trích ra từ ứng dụng khách …

3. THIẾT KẾ ĐỐI TƯỢNG CHO GIAO THỨC TCP/IP

3.1 Kiến trúc hệ thống lớp bao bọc giao thức TCP/IP



Hình 3. Kiến trúc hệ thống bao bọc giao thức TCP/IP

Hình 3 trình bày kiến trúc hệ thống lớp được thiết kế để bao bọc thư viện lập trình với giao thức TCP/IP nhằm giải quyết, khắc phục các vấn đề mà chúng tôi đã nêu ra đối với hệ thống hàm thư viện có sẵn. Một số điểm chính cần thảo luận trong kiến trúc này:

- Lớp **CommHandle** (Communication Handle) là lớp trừu tượng, thực chất là một giao diện lớp (class interface) được sử dụng để bao bọc các hàm lập trình theo giao thức TCP/IP mà không phụ thuộc vào chi tiết kỹ thuật của giao tiếp lập trình trên từng môi trường cụ thể như Windows hay Unix. Lớp này được hiện thực bởi 2 lớp cụ thể phụ thuộc môi trường: **WinCommHandle** (communication handle of Windows) và **BsdCommHandle** (BSD communication handle) ;
- Lớp **CommInfo** (Communication Information) bao bọc các thông tin liên quan đến địa chỉ IP, cổng truyền thông và thông in khác chứa trong cấu trúc **sockaddr_in** có sẵn ;
- Lớp **CommPoint** (Communication Point) có mục đích trừu tượng hóa các đối tượng truyền thông, dùng làm cơ sở để giữ các đối tượng liên lạc giữa các máy tính với nhau. Lớp này được đặc biệt hóa thành 2 lớp **ServerPoint** và **ClientPoint** bên dưới ;
- Lớp **ServerPoint** (Communication Point for server side) định nghĩa các đối tượng truyền thông của ứng dụng chủ (server), đây là sự trừu tượng hóa của đối tượng truyền thông chủ mà không phụ thuộc vào chi tiết kỹ thuật của giao thức TCP/IP cụ thể ;
- Lớp **ClientPoint** (Communication point for client side) nhằm trừu tượng hóa các đối tượng truyền thông khách, dùng để tạo các đối tượng được bên trong ứng dụng khách ;
- Lớp trừu tượng **EnvConfig** (Environment Configuration) dùng làm cấu hình để tạo ra các đối tượng truyền thông trên một môi trường cụ thể, lớp này được đặc biệt hóa thành: lớp **WinConfig** cho Windows và lớp **BsdConfig** cho họ Unix.

3.2 Các mẫu thiết kế Gamma đã sử dụng

Trong giải pháp trình bày bên trên chúng tôi đã sử dụng phối hợp một số mẫu thiết kế của Gamma và cộng sự [4], mỗi mẫu được dùng đã giải quyết từng khía cạnh, từng vấn đề mà

chúng tôi đã nêu lên trong các phần trước. Có thể tóm tắt việc sử dụng các mẫu Gamma trong nghiên cứu này như sau :

- Mẫu ***Adapter*** được dùng làm tương thích hai thư viện socket khác nhau, điều này được thể hiện bởi lớp trừu tượng **CommHandle** định nghĩa một giao tiếp lập trình chung và được hiện thực bởi 2 lớp cụ thể là **WinCommHandle** và **BsdCommHandle** ;
- Mẫu ***Bridge*** được vận dụng để tách rời tính trừu tượng của các đối tượng truyền thông khỏi các yếu tố kỹ thuật liên quan đến giao thức TCP/IP: các lớp như **CommPoint**, **ServerPoint** và **ClientPoint** mô hình hóa các đối tượng truyền thông; trong khi đó thì các lớp **CommInfo**, **CommHandle**, **WinCommHandle** và **BsdCommHandle** quán xuyến các thông tin và các phương thức truyền thông ;
- Mẫu ***Abstract Factory*** được dùng để chọn thư viện socket của một môi trường cụ thể (Windows hay Unix) cho ứng dụng đang xây dựng, người lập trình chỉ cần tạo đối tượng thích hợp khi muốn phát sinh ứng dụng ứng với bộ thư viện nào ;
- Mẫu ***Singleton*** (điều này được thể hiện trong cài đặt của các lớp **WinConfig** và **BsdConfig**) được dùng để bảo đảm chỉ có duy nhất một đối tượng cấu hình được tạo ra: lớp **WinConfig** là một lớp đặc biệt mà chỉ có một thể hiện (lớp chỉ có 1 đối tượng duy nhất) và lớp **BsdConfig** cũng tương tự như vậy.

4. CÀI ĐẶT CÁC LỚP ĐỐI TƯỢNG VÀ ÁP DỤNG

Kiến trúc lớp bao bọc giao thức TCP/IP đã được cài đặt bằng ngôn ngữ C++ và chạy thử nghiệm trong một số ứng dụng trên Windows và Linux. Trong phần này, chúng tôi lưu ý một số điểm chính trong cài đặt hệ thống lớp và trình bày một ứng dụng đơn giản nhằm để minh họa.

4.1 Một số điểm chính trong cài đặt kiến trúc lớp

Các lớp đều được cài đặt theo đúng thiết kế trong hình 3. Chẳng hạn các lớp **CommPoint**, **ServerPoint** và **ClientPoint** được mô tả như sau:

```
#include "CommHandle.h"
#ifndef class_CommPoint
#define class_CommPoint
class CommPoint {
protected:
        CommInfo* serverInfo;
        CommHandle*
serverHandle;
public:
        int IsValid();
        virtual int Link()=0;} ;
#endif
```

```
#ifndef class_ClientPoint
#define class_ClientPoint
#include "CommPoint.h"
class ClientPoint:public CommPoint {
public:
    ClientPoint(CommHandle*  hServer,  unsigned
short port,
                char*
serverAddr="255.255.255.255" );
    virtual int Link();
    int SendServer(string buffer, int flag);
    int RecvServer(string& buffer, int flag);} ;
#endif
```

```
#ifndef class_ServerPoint
#define class_ServerPoint
#include "CommPoint.h"
class ServerPoint:public CommPoint {
        CommInfo* clientInfo;
        CommHandle* clientHandle;
        int AcceptClient();
public:
        ServerPoint(CommHandle* hServer, unsigned short port);
        virtual int Link();
        int WaitforClient(int backlog);
        int SendClient(string buffer, int flag);
        int RecvClient(string& buffer, int flag);};
#endif
```

Phương thức *Link( )* có ý nghĩa là kết nối socket với thông tin của máy chủ, được hiện thực bằng phương thức *Connect( )* đối với **ClientPoint**, nhưng đối với **ServerPoint** thì lại là *Bind( )*:

```
int ClientPoint::Link() {
    return serverHandle->Connect(serverInfo);}
```

```
int ServerPoint::Link() {
    return serverHandle->Bind(serverInfo);}
```

Phương thức *WaitforClient( )* của **ServerPoint** được dùng để chờ ứng dụng khách kết vào, được hiện thực bằng cả 2 phương thức *Listen( )* và *Accept( )*:

```
int ServerPoint::WaitforClients(int backlog) {
        int err=serverHandle->Listen(backlog);
        if ( err == 0 )
                return AcceptClient();
        else
                return err;}
```

Lớp cấu hình cho Windows là **WinConfig** (tương tự với **BsdConfig**) được cài đặt như sau. Chú ý rằng vì chúng ta dùng mẫu ***Singleton*** nên **WinConfig** có hàm tạo (constructor) được dấu trong lớp; việc tạo lập đối tượng cấu hình thực hiện nhờ phương thức tĩnh ("*class level method*") tên là *CreateEnvObject( )* chỉ tạo ra 1 đối tượng duy nhất, nếu đối tượng đã tạo thì chỉ việc trả nó về.

```
EnvConfig* WinConfig::envObj=NULL;
//...
class WinConfig:public EnvConfig {
  WinConfig();
public:
  virtual int Startup();
  virtual int Cleanup();

  static EnvConfig* envObj;
  static EnvConfig* CreateEnvObject();
  virtual CommHandle*
CreateCommObject(int);} ;
```

```
// Signgleton Pattern
EnvConfig* WinConfig::CreateEnvObject(){
    if (envObj == NULL){
        envObj = new WinConfig();}
    return envObj;}
CommHandle*
WinConfig::CreateCommObject(int type)
{return new WinCommHandle(type);}
...
```

4.2 Ứng dụng minh họa việc sử dụng các lớp đã xây dựng

Chúng ta xem một ứng dụng đơn giản truyền thông điệp giữa ứng dụng chủ và ứng dụng khách, mỗi lần ứng dụng khách truyền một từ (chuỗi ký tự) đến ứng dụng chủ. Trong hình 4,

đoạn mã nguồn phía trên được viết cho ứng dụng khách. Hàm *simpleClient( )* khai báo đối tượng truyền thông khách, tên là *client*, được tạo rất đơn giản nhờ dùng hàm tạo lập (constructor) của lớp **ClientPoint**; sau đó là thao tác nối kết *client.Link( )* và gởi liên tục dữ liệu đến ứng dụng chủ (đã chạy sẵn và chờ). Ta có thể thấy hàm *simpleClient( )* được viết tổng quát không phụ thuộc trực tiếp vào thư viện socket cụ thể, chỉ cần biết địa chỉ IP của máy kết nối và cổng truyền thông số mấy. Trong hàm *main( )*, phương thức *WinConfig::CreateEnvObject( )* tạo ra đối tượng cấu hình *cfg* trên Windows: nhờ đa xạ việc gọi *cfg->CreateCommObject(SOCK_STREAM)* và *cfg->Startup( )* sẽ kết buộc với các đối tượng ứng với thư viện socket của Windows.

Một cách tương tự, hàm *simpleServer( )* (xem đoạn mã nguồn phía dưới, trong hình 4) cũng được viết không phụ thuộc vào thư viện socket: hàm viết rất đơn giản bằng cách tạo ra một đối tượng truyền thông của lớp **ServerPoint**, chờ ứng dụng khách kết nối vào, sau đó nhận dữ liệu được gởi liên tục từ ứng dụng khách. Với mục đích minh họa, đoạn mã nguồn trong hình 4 được viết dưới dạng nhập xuất console. Tuy nhiên chúng ta hoàn toàn có thể sử dụng hệ thống lớp đã xây dựng để viết các ứng dụng chạy ẩn bên dưới (dạng các dịch vụ) hay tích hợp vào các ứng dụng có giao diện đồ họa.

```
#include "ClientPoint.h"
#include <string>
#include <iostream>
using namespace std ;
int simpleClient(CommHandle* hServer, char* serverIP, unsigned short port) {
      string data;
      ClientPoint client(hServer, port, serverIP );  client.Link();
      while(1) {
            cout<<"Enter data:";   cin >> data;
            int nByte=client.SendServer(data, 0);
            if(nByte != data.length()+1) {cout << "End connection..." << endl; return 1; }}}
#include "WinCommHandle.h"  // Phụ thuộc Windows ...
int main(int argc, char* argv[]) {
      EnvConfig *cfg=WinConfig::CreateEnvObject(); unsigned short port=4965;
      if(cfg==NULL) return 0;
      if(cfg->Startup() != 0) { cout << "Socket error..." << endl;  return 0; }
      CommHandle* hServer=cfg->CreateCommObject(SOCK_STREAM);
      simpleClient(hServer, argv[1], port);
      cfg->Cleanup();
      return 1;}
```

```
int simpleServer(CommHandle* hServer, unsigned short port) {
      ServerPoint svrPoint(hServer, port);
      svrPoint.Link(); svrPoint.WaitforClients(1);
      while(1) {
            string data;
            int nByte=svrPoint.RecvClient(data, 0);
            if (nByte == 0) {cout << "End connection..." << endl; return 1; }
            cout << "From client:" << data << endl;}}
```

Hình 4. Mã nguồn của ứng dụng khách truyền thông điệp đến ứng dụng chủ

5. CÁC NGHIÊN CỨU LIÊN QUAN VÀ KẾT LUẬN

Việc áp dụng ngôn ngữ mẫu (pattern language) vào các ứng dụng mạng và truyền thông cũng đã được quan tâm bởi nhóm nghiên cứu của Y. Byun [1], tuy nhiên nhóm tác giả này đặt

trọng tâm nhiều về các mẫu kiểm soát trạng thái của các đối tượng truyền thông. Tác giả P. E. Sevinç và cộng sự [8] cũng đề xuất việc áp dụng các mẫu của Gamma [4] vào việc phát triển các ứng dụng mạng, tuy nhiên kết quả của nhóm này chỉ mới bắt đầu ở mức độ nghiên cứu tổng quan. Kiến trúc MFC [6] của Microsoft cũng bao gồm các lớp bao bọc giao thức TCP/IP (chẳng hạn như *CAsyncSocket* và *CSocket*), tuy nhiên những lớp này chỉ bao bọc đơn thuần các hàm socket và gắn quá chặt vào MFC nên không thể dùng được trên các môi trường lập trình họ Unix. Môi trường ACE (Adaptive Communication Environment) [7] là môi trường hỗ trợ lập trình mạng trên ngôn ngữ C++ với một hệ thống lớp rất lớn, đáp ứng nhiều khía cạnh đa dạng của các ứng dụng truyền thông. Hiện tại chúng tôi đang nghiên cứu, phân tích kiến trúc lớp của ACE để có những định hướng thích hợp cho việc phát triển các ứng dụng mạng theo tiếp cận hướng đối tượng.

Trong bài báo này chúng tôi đã tóm tắt kết quả nghiên cứu liên quan tiếp cận vận dụng mẫu thiết kế để xây dựng hệ thống lớp hỗ trợ lập trình TCP/IP một cách dễ dàng và hiệu quả hơn trong việc tái sử dụng mã nguồn. Việc nghiên cứu ứng dụng mẫu thiết kế và kỹ thuật hướng đối tượng vào việc phát triển các ứng dụng mạng là một tiếp cận có nhiều ứng dụng thiết thực. Chúng tôi sẽ tiếp tục nghiên cứu để ứng dụng phương pháp hướng đối tượng vào nhiều khía cạnh, vấn đề khác trong việc xây dựng các ứng dụng mạng và truyền thông.

APPLYING OBJECT- ORIENTED DESIGN PATTERNS IN CONSTRUCTING TCP/IP NETWORK APPLICATIONS

Tran Dan Thu, Huynh Thuy Bao Tran
University of Natural Scineces, VNU-HCM

ABSTRACT: *TCP/IP protocol plays an important role in constructing network applications. Network applications which use TCP/IP protocol are usually implemented in C/C++ with available function libraries. However, it is difficult to reuse source codes because of the existence of several function libraries for Unix and Windows with different programming interfaces. Besides, these programming interfaces cause clumsy, ambiguous and duplicate fragments of source codes. In this paper, we apply design patterns to develop an architecture of classes encapsulating TCP/IP protocol. These classes help developers write compatible source codes which can be reused in different environments, and create communication objects with more clear semantics.*

Keywords: *Design Patterns, TCP/IP protocol, Object – Oriented Programming*

TÀI LIỆU THAM KHẢO

[1]. Y. Byun, B. Sanders, and K. Chung, *A Pattern Language for Communication Protocols*, Pattern Languages of Programs conference, 2002.

[2]. Douglas E. Comer and David L. Stevens, *Internetworking with TCP/IP: Client-server programming and applications BSD socket version*, Prentice Hall, 1993.

[3]. DONG T. B. Thuy and TRAN D. Thu, *User Interface Design by Applying Object – Oriented Design Patterns,* Addendum Contributions to the 4th IEEE International Conference on Computer Sciences Research, Innovation & Vision for the Future, February 12-16, Hochiminh City, Vietnam, 2006.

[4]. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Longman,1995.

[5]. Nguyễn Tiến Huy, Trần Đan Thư, Trần Hạnh Nhi, Chương 10 – Kỹ thuật TCP/IP, *Kỹ thuật lập trình trên môi trường Windows NT – Tập 2: Lập trình giao tiếp mạng*, Nhà xuất bản giáo dục , 1998.

[6]. R. Jones, *Introduction to MFC Programming*, Prentice Hall, 1999.

[7]. Douglas C. Schmidt and Stephen D.Huston, *C++ Network Programming: Mastering Complexity with ACE and Patterns*, Addison Wesley Longman, 2002.

[8]. P. E. Sevinç, J. P. Martin-Flatin, R. Guerraoui, *Patterns in SNMP-Based Network Management ,* Proc. 17th International Conference on Software and Systems Engineering and their Applications, November 2004 , Paris, France, 2004.

[9]. Jon C. Snader, *Effective TCP/IP Programming*, Addison-Wesley, 2000.